TÉCNICO LISBOA

Asynchronous Audio Sample Rate Converter

Pedro Miguel Portela Teixeira

Introduction to the Research in

Electrical and Computer Engineering

Supervisor(s): Prof. José João Henriques Teixeira de Sousa

Examination Committee

Supervisor: Prof. José João Henriques Teixeira de Sousa
Member of the Committee: Dr. Marco António da Mota Carvalho Silva Pereira

January 2020

Abstract

Currently, in digital audio processing, sampling rate convertions are common. While there solutions for this problem is the form of software and integrated circuit implementations, there is no viable solution in the form of intellectual property, existing only one (limited) solution.

A sample rate converter can be structured as: (1) an interpolation module, (2) a reconstruction and anti-aliasing filter (fractional delay filter), and (3) a decimation module. This structure can be optimized by joining these modules, avoiding computation of decimated samples. The two main concerns of this design are the design of the fractional delay filter, and the synchronization mechanisms used to guarantee that the filter is adjusted with changes on the conversion ratio.

The focus of this report is on the study of asynchronous sampling rate converter designs, as well as the digital signal processing techniques used in the current state-of-the-art. This study leads to a proposed design of a prototype multi-channel asynchronous sampling rate converter.

Keywords:Asynchronous Sample Rate Converter, Digital Signal Processing, FPGA, Fractional Delay Filter

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Topic Overview

In 1977, with the growing popularity of audio interfaces, the Audio Engineering Society (AES) founded the AES Digital Audio Standards Committee, creating some of the most used audio standards to this day. One of the most popular standards, the AES3 digital audio interface, is stil used in current audio equipment, like microphones and speakers with XLR connectors.

Since then, there has been a significant rise in AES standards, most of them demanding the conversion of an audio signal's sample rate. One of the classical examples is the the conversion from CD quality with a sampling rate $F_s = 44.1kHz$ to DVD quality with $F_s = 48kHz$. By consequence, there is a great demand for sample rate converters, both in the form of software algorithms and hardware designs.

To answer this need, some integrated circuit manufacturers developed multiple sample rate converters with varied specifications. The AD1896 [1], for instance, is an asynchronous sample rate converter made by Analog Devices, in 2003. It supports a stereo (2 channel) audio signal and converts its sampling rate from 7.75:1 to 1:8 ratios. Another example is the SRC4194 [2], manufactured by Texas Instruments since 2004, supporting up to 4 channel inputs, and a wider sampling rate ratio, ranging from 16:1 to 1:16 ratios. The first Intelectual Property (IP) was developed by Coreworks and called the CWda52 [3], a multi-channel sample rate converter, able to convert sampling rates using ratios from 8:1 to 1:8.

1.2 Motivation

While there are currently hardware implementations of asynchronous sample rate converters, most of them are integrated in a circuit (IC), while implementations as intelectual property (IP) are few and have limited specifications compared to their IC counterpart. This makes it difficult for some companies that manufacture devices like television sets to integrate a cheaper sample rate converter which fulfills the specifications of the respective chipsets.

The overall growth of the market for embedded audio systems which require the sample rate conversion function motivates the development of better sample rate converter IP cores, as a cheaper and more versatile alternative to discrete solutions.

1.3 Objectives

The main objective of this work is to design an asynchronous sample rate converter (ASRC) IP core using the Verilog hardware description language. This converter should support any input or output signal, sampled from 8 kHz to 192 kHz, with up to 24-bit samples. To ensure high audio quality, the sample rate converter should ensure a total harmonic distortion plus noise ratio (THD+N) of -130 dB or less. It should also support multi-channel signals, with a channel limit that depends on the computation time restraints.

The ASRC should be tested in simulation and field programmable array logic (FPGA) boards, using a PC to send and receive audio signals via Ethernet to the FPGA board; the audio signals originate from .wav files, and the result of sample rate conversion is stored back in another .wav file.

At the initial stages of this work some of the most used sample rates will be tested while the input sample rate is kept constant. Later, dynamically varying input sample rates will be tested. The work requires the study of digital signal processing techniques, with emphasis on digital signal filtering and interpolation techniques.

1.4 Author's Work

Part of the work reported here was done in a summer internship at IObundle. At the beginning of the internship, a non-functional Verilog prototype was already available, as well as an Octave model detailing the structure of the ASRC. This made it possible not only to gain knowledge of the dataflow of the sample rate conversion algorithm, but also to have a starting point for developing of the system. The work was focused on the reduction of the output signal's THD+N and optimization of the measurement of input and output sample rates for overall synchronization.

1.5 Report Outline

This report is composed of 4 more chapters. In the second chapter, the sample rate conversion algorithm will be generically explained and some sample rate converter designs will be presented, along with their advantages and disadvantages. In the third chapter, a hardware implementation is proposed, containing the architecture of the core along with its interfaces and submodules. In the fourth chapter, the process of simulation and implementation of the core is described and some preliminary results are presented. In the fifth and final chapter, some conclusions of the research needed for this thesis are drawn, and an outline of the work that still needs to be done is defined.

Chapter 2

Asynchronous Sample Rate Conversion

An ideal Asynchronous Sample Rate Converter (ASRC) is able to convert the sample rate of an input audio signal to a desired output sample rate without any loss of signal quality. As oppsed to a synchronous sample rate converter, the ASRC also needs to measure the value of the input and output sample rates continuously to compute the conversion ratio. The ideal ASRC converts a discrete time input signal $x[n]$, sampled at a rate F_{s1}, to a continuous time signal $x(t)$, with the use of a reconstruction filter. This signal is then filtered by an anti-aliasing filter which ensures that its output $y(t)$ has no components which would violate Nyquist's law. Signal $y(t)$ is then converted to a discrete time signal $y[m]$, sampled at the desired output rate F_{s2} [4].

A block diagram of this model is shown in Fig. 2.1. Note that the reconstruction filter and anti-aliasing filters can be combined in a single Low Pass Filter (LPF).

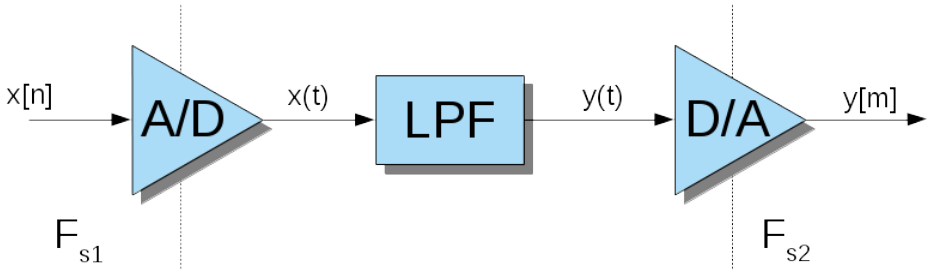

Figure 2.1: Analog interpretation of a sample rate converter

2.1 Sample Rate Converter's Structure

One of the challenges of creating a purely digital solution is the design of the LPF digital filter, which is at one time accurate and efficient. The classical approach to this problem consists in upsampling the

signal by a factor L (interpolation), doing the processing at the frequency $L \times F_{s1}$, and downsampling the result by a factor M (decimation), as a means to emulate a discrete to continuous and continuous to discreve signal convertion [5]. A simple block design of this algorithm is shown in Fig. 2.2.

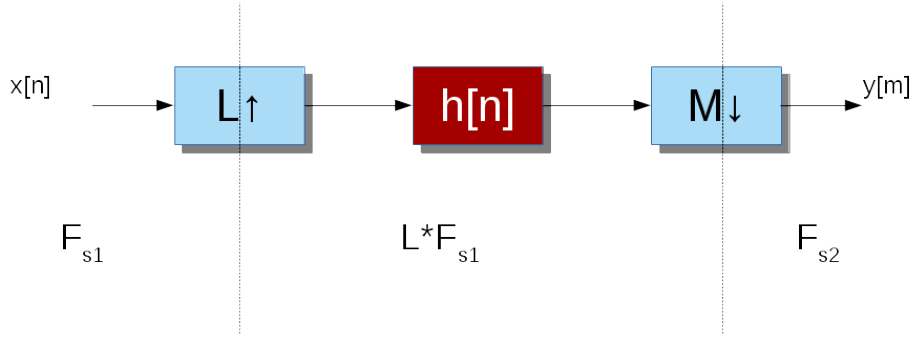

Figure 2.2: Classic digital sample rate conversion algorithm by factor L/M

In practice, an input signal $x[n]$, sampled at frequency F_{s1} is upsampled by insertion of $L-1$ samples with value zero between two consecutive input samples. The resultant signal is then inserted into a low pass filter $h[n]$, which will interpolate the inserted samples and avoid aliasing. Finally, the output of the filter is downsampled by taking only every M-th sample for the output signal $y[m]$. The relationship between L and M is such that

$$F_{s2} = \frac{L}{M} F_{s1}. \tag{2.1}$$

Regarding the filter $h[n]$, its cutoff frequency depends on the relationship described in Equation (2.1). In the upsampling case ($L \geq M$), there is the need to remove the resultant spectral images, using a filter with a normalized cutoff frequency ($\Omega_c \leq 0.5\pi/L$). In the case of downsampling ($L < M$), there is the need to filter signal frequencies that would cause aliasing, leading to a filter with a normalized cutoff frequency ($\Omega_c \leq 0.5\pi/M$). By joining the two conditions, the resultant filter should have a cutoff frequency

$$\Omega_c = min(\frac{\pi}{2L}, \frac{\pi}{2M})[rad]. \tag{2.2}$$

Note that the normalization considered is in relation to the upsampled frequency:

$$\Omega = \frac{2\pi f}{L f_{s1}}[rad]. \tag{2.3}$$

In the current state of the art, this algorithm is the basis of most synchronous and asynchronous sample rate converters.

For conversions which use small values of L and M the computational effort is modest. However, if the conversion involves sampling rates with a small difference, the factors L and M will increase significantly, increasing drastically the computational cost of the conversion, only to have most of the computed samples discarded. Note that in this case a small normalized cutoff frequency must be used for the filter.

Furthermore, there is the need to design the filter to both remove aliasing and interpolate the $L-1$ inserted samples. This is why design of the filter $h[n]$ is the main challenge of this architecture. The solution presented in this thesis is based on the use of a fractional delay filter.

Additionally, for asynchronous sample rate converters, the filter is not only time-varying, but also varies with the sample rate ratio. This means that there is the need to define a structure that computes the ratio and adapts the filter. For synchronous sample rate converters, the ratio stays constant. This can lead to a predictable filter, leading to the possibility to trade off storage space for computation time, by precomputing a finite set of filters [6].

2.2 Fractional Delay Filter

An efficient way of obtaining an interpolated value of a sample is to consider that the output samples needed correspond to the input samples, delayed or advanced by a certain value. Considering a discrete-time signal $y[n]$, obtained by delaying a signal $x[n]$,

$$y[n] = x[n - \tau_d] = x[n] * h_d[n], \tag{2.4}$$

where τ_d is the normalized delay. For continuous signals, a time shift in the frequency domain can be expressed as a product between an input $X(e^{j\omega})$ and a filter $H(e^{j\omega})$,

$$Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}), \tag{2.5}$$

where

$$H(e^{j\omega}) = e^{-j\omega\tau_d}. \tag{2.6}$$

By analysis of Equation (2.6), it is possible to note that a delay filter is an allpass filter with unitary gain and linear phase. However this is only the case for a continuous time domain $x(t)$ signal. For the digital signal $x[n]$, which needs to be reconstructed and aliasing-free, a low pass filter with a cuttoff frequency defined by Equation (2.2) is needed. Since both the reconstruction and anti-aliasing filters are linear systems, they can be combined together in a single filter whose frequency response $H_d(e^{j\Omega})$ is the product of the frequency responses of the two filters:

$$H_d(e^{j\Omega}) = \begin{cases} 1, |\Omega| < \Omega_c \\ 0, otherwise \end{cases}. \tag{2.7}$$

To obtain the impulse response of filter $h[n]$, defined in Equation (2.4), the inverse discrete-time Fourrier transform (IDTFT) of $H_d(e^{j\omega})$ is performed.

$$h[n] = IDTFT(H_d(e^{j\omega})) = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-j\Omega\tau_d}e^{j\Omega n}d\Omega. \tag{2.8}$$

Since the integrated function is non-zero only in the interval $-[-\Omega_c, \Omega_c]$ and $e^{-j\Omega\tau_d}e^{j\Omega n} = e^{j\Omega(n-\tau_d)}$, the integral can be solved yielding

$$h[n] = \frac{sin(\Omega_c(n-\tau_d))}{\pi(n-\tau_d)}. \tag{2.9}$$

The impulse response of the filter $h[n]$ is then defined by Equation (2.10), which is the normalized $sinc$ function.

$$h[n] = \frac{\Omega_c}{\pi} sinc\left[\frac{\Omega_c}{\pi}(n-\tau_d)\right]. \tag{2.10}$$

Fig. 2.3 represents this function for $\tau_d = 0$. Note that a variation of τ_d is equivalent to a translation of the figure in the n (discrete time) axis. By direct observation of the figure, it is possible to note that for integer values of τ_d, $h[n] = 0$ for all samples except for $n = \tau_d$ where $h[n] = 1$. This is the expected filter response for an integer delay. On the other hand, for a fractional value of τ_d, $h[n]$ is non-zero for all samples. For the ASRC algorithm, $0 \leq \tau_d \leq 1$, since the objective is to use this fractional delay filter as a way to obtain an interpolated sample using a finite number of input neighbour samples separated by an unitary normalized delay.

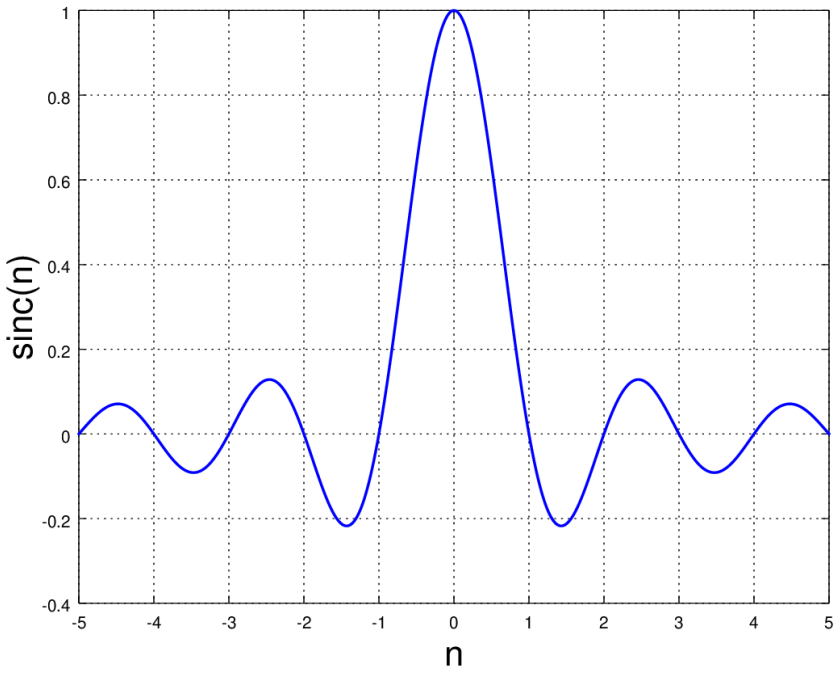

Figure 2.3: Graphical representation of $h[n]$ for $\tau_d = 0$

As this function is infinite and non-causal, an approximation must be done while retaining enough low-pass filtering capability to ensure quality, as explained in Section 2.1. This problem can be solved by applying a window to the ideal filter $h[n]$. The choice of the format of the window and bandwidth have an influence not only on the quality of ASRC's output signal, but also on the complexity of the computations done.

Using the truncation of the impulse response as an approximation technique, the resultant filter is a section of the sinc function which contains a certain number of zeroes. To exemplify, a filter with six zeroes is considered in Fig. 2.4, and Fig. 2.5.

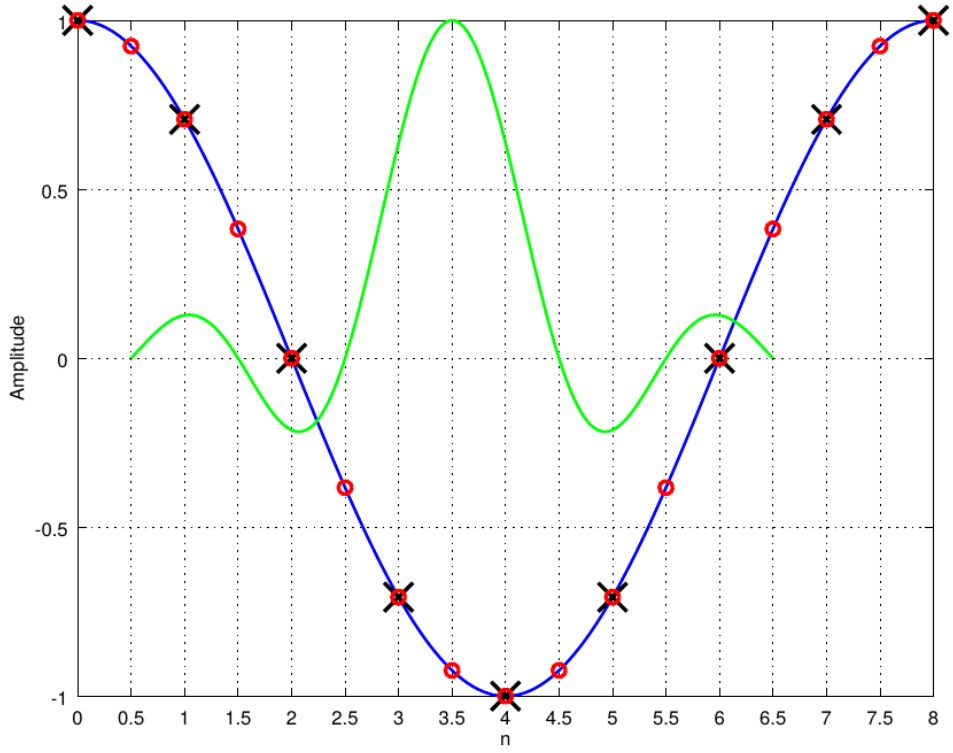

Figure 2.4: Illustration of upsampling (1:2) used to determine the output sample at n = 3.5

In Fig. 2.4, the digital input signal, whose samples are represented by crosses, is plotted along its continuous time representation. To upsample this signal by a factor of 2, the output samples, represented by circles, need to be computed. To interpolate these signals, the filter approximation is applied, centring the window at the desired output sample, multiplying each input sample by the corresponding filter value, and accumulate all obtained values. By analysis of this illustration, it is possible to note that a larger upsampling factor will decrease the distance between the output samples. According to Equation (2.2), the normalized cutoff frequency of the filter is $0.5/L$. The filter is always the same in the upsampling case and its number of accumulations is the same as the number of zeroes in the truncated sinc function.

Fig. 2.5 is analogous to Fig. 2.4, representing downsampling by a factor of 2 instead. In this case, according to Equation (2.2), the cutoff frequency of the filter is given by $0.5/M$. The number of accumulations is now M times the number of zeroes in the truncated sinc function.

Regarding the approximation of the filter itself, there is a great variety of techniques used to obtain a fractional delay filter which minimizes the discretization error: applying a window to the ideal filter, applying Lagrangian interpolation to compute an intermediate coefficient from a set of known filter samples, etc. These and other methods are explained in detail in [7–10].

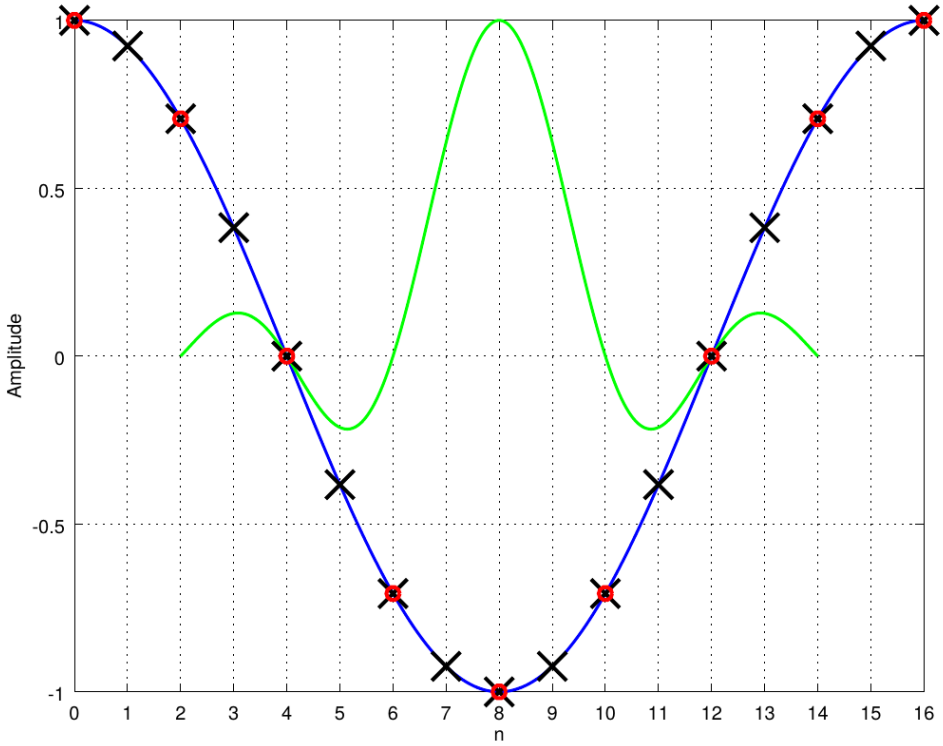

Figure 2.5: Illustration of downsampling (2:1) used to determine the output sample at n = 4

2.3 Output Sample Computation

As shown in Fig. 2.4 and 2.5, to obtain an output sample, the fractional delay filter is applied to the input signal, by centring it to the desired time of the output sample and computing the result of the (discrete) convolution, as expressed in Equation (2.4). Furthermore, it is important to note that, since this converter is applied to audio, the digital design of the filter should make sure that the filter has linear phase. This is achieved by using a Finite Impulse Response (FIR), which has a non-recursive structure. In an FIR, each output $y[n]$ is obtained by

$$y[n] = \sum_{i=0}^{N} a_i x[n-i], \tag{2.11}$$

where a_i are the coefficients of the filter and $x[n-i]$ is the input sample at discrete time $n-i$. It is important to note that the filter described in Section 2.2 is non causal. To fix this, the computation of the output is delayed by as many input samples as accumulations that involve future input samples.

With this implementation, the computation of an output sample is reduced to a multiply-accumulate (macc) operation. However, it requires a large amount of filter coefficients, which, in hardware, results in high memory usage, an undesirable result.

To guarantee that the sample rate converter supports multiple channels, the structure of the converter should be able to support the computation of multiple output samples. This can be done at the cost of hardware area, by having one output computation unit per channel, or at the cost of computation time,

by having one unit computing multiple outputs sequentially. To make this possible, there is the need to optimize this not only regarding hardware area, but also computation time.

2.4 Sample Rate Converter Implementations: An Analysis Of The State Of The Art

2.4.1 Cascaded Integrator Comb Filters (CIC)

As was explained in Section 2.1, sample rate conversion can be interpreted as signal upsampling with interpolation, followed by downsampling with decimation. Furthermore, this algorithm can be implemented using FIR filters as shown in the previous section. One of the designs comonly used for this purpose is a Cascaded Integrator Comb filter (CIC) [11]. This filter, originally designed by Hogenauer [12], is based on the implementation of simple integrators and differentiators. CIC filters are obtained by combining adders and delay registers, which consumes a reduced amount of memory and has the great advantage of dispensing with multipliers, which can consume a great amount of hardware resources. Due to the fact that the CIC decimator has a symetric structure in comparinson to a CIC interpolator, the combination of the two components leads to a highly efficient implementation in application specific integrated circuits (ASIC) and FPGA's. An interpolation filter using the CIC structure can be seen in Fig. 2.6. A decimation filter would have the same components, with the difference that the comb filters and integrator stages would swap positions.

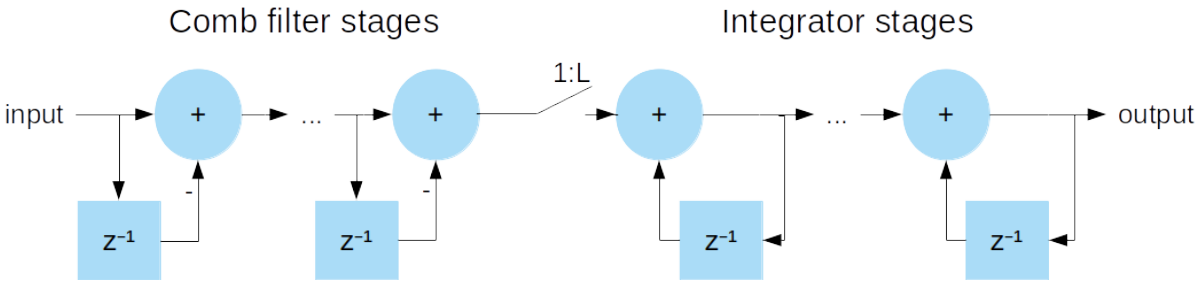

Figure 2.6: Example of cascaded integrator comb filter structure, used as an interpolation filter

While for one CIC structure, the frequency response does not fulfill the requirements, this problem can be solved by cascating multiple interpolation and integration units [11], increasing the attenuation in the filter's stopband. This implementation is elegant but has limited configurability, due to the fact that no coefficients are used, making it hard to adapt the filter to variations in the sample rates. Furthermore, this type of filter expects integer factors, which leads to the problem of making it able to convert fractional sample rate relations.

2.4.2 Approximation By Piecewise Quadratic Function

As explained in Section 2.2, a direct way to implement the sample rate converter is to consider that the algorithm can be reduced to simply applying a fractional delay filter to the input signal, with a sinc

impulse response as expressed by Equation (2.10). In practice this is impossible because the filter has an infinite number of taps and requires the use of future samples (non-causality).

The solution presented so far is to truncate and approximate the coefficients and delay the response to make it causal. Another way to address this problem is to split the sinc function into a piecewise function, and aproximate each piece by a quadractic function, leading to an interpolation filter which can be easily implemented [13, 14]. The approximation of the piecewise sinc function $h(x)$ into quadractic functions can be expressed as

$$h(t) = \begin{cases} a_{1,1}t^2 + b_{1,1}t + c_{1,1}, \left(0 \leq |t| \leq \frac{1}{N}\right) \\ \vdots \\ a_{1,n}t^2 + b_{1,n}t + c_{1,n}, \left(\frac{n-1}{N} \leq |t| \leq 1\right) \\ \vdots \\ a_{s,n}t^2 + b_{s,n}t + c_{s,n}, \left(s - 1 + \frac{n-1}{N} \leq |t| \leq s - 1 + \frac{n}{N}\right) \\ \vdots \\ a_{S,n}t^2 + b_{S,n}t + c_{S,n}, \left(S - 1 + \frac{n-1}{N} \leq |t| \leq S\right) \end{cases}, \tag{2.12}$$

where N is the number of quadratic functions used to represent a piecewise function, and S is the number of piecewise functions of the kernel. This technique leads to the implementation presented in Fig. 2.7.

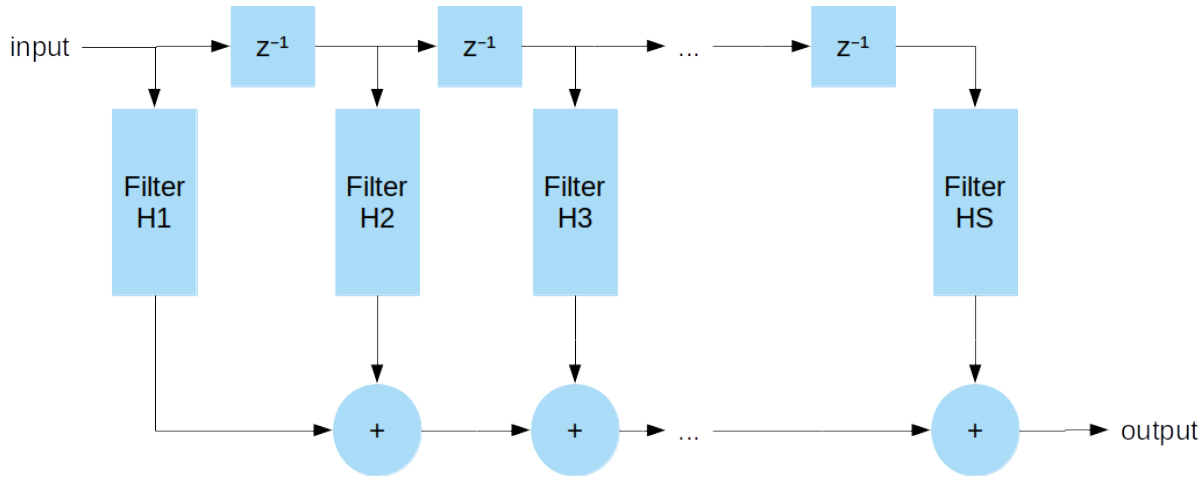

Figure 2.7: Example of a piecewise kernel filter structure

This design can be futher optimized, by considering that from a certain section onward, the polynomial remains the same, changing only in scale by a determined set of factors e_j. The block diagram of this structure is presented in Fig. 2.8. This leads to a structure where the polynomials used remain the same, while the factors e_j change with the sample rate convertion ratio. There are, however, some restrictions that need to be considered, as further explained in [13].

The main advantage of this design is the ability to change sample rate ratio without need to reconfigure the filter, leading to a kernel which not only is able to implement the function, but also does not need

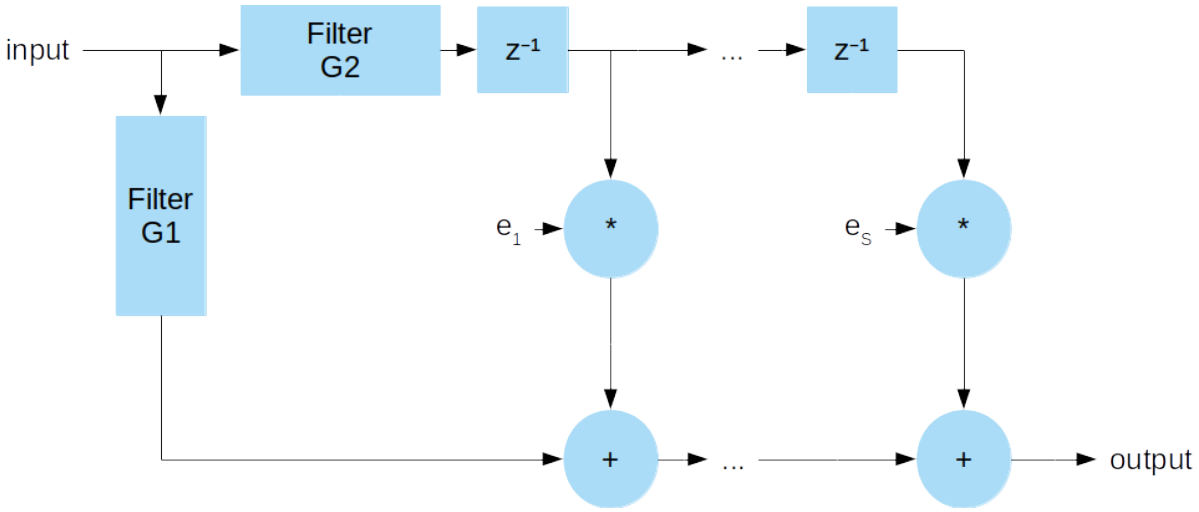

Figure 2.8: Example of optimized kernel filter structure

to change over time. However, as one needs to define a high number of pieces to obtain large attenuation in the stopband (to acceptably attenuate distortion and noise), it is needed to use a large number of quadratic functions, which significantly increases the computational cost. Apart from the structure itself, there is still the high cost of computing the coefficients e_j, which turns even more problematic with applications for which the sample rate convertion ratio varies in time.

2.4.3 Farrow Structure

The most common implementations of fractional delay filters, and, by consequence, sample rate converters, is one which uses the Farrow Structure [15]. This structure is based on the assumption that each sample of the filter impulse response $h[n]$ can be approximated by a polynomial of order q, which depends on the fractional delay d [16]:

$$h[n] = \sum_{m=0}^{q} c_m[n] d^m. \tag{2.13}$$

Similar to the structure presented in Section 2.4.2, a Farrow structure requires the implementation of a set of q FIR filters, which becomes a hardware-intensive design. Furthermore, a change in the fractional delay forces the change of all coefficients, leading to an increase of the computational cost. Over the years, this structure has been subjected to many changes and optimizations, with the objective to develop filters for different applications. For the specific application this thesis is concerned, the structure is optimized to design filters with a variable fractional delay. One possible optimization, further explained in [17], makes it possible to implement a low area Farrow structure, with the great advantage of having fixed coefficients and a single parameter μ that depends on the fractional delay. The block design of this structure is presented in Fig. 2.9, where the value of μ can be computed by

$$\mu = \frac{k}{M}, \tag{2.14}$$

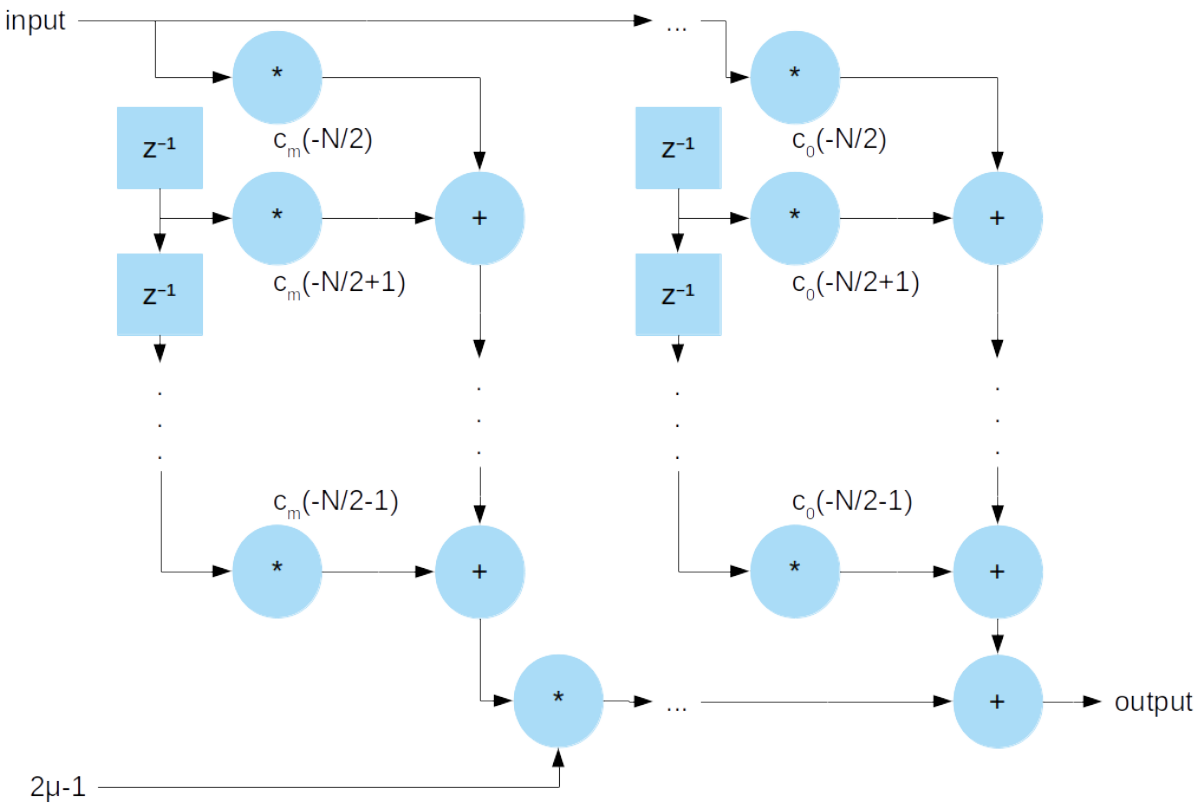

Figure 2.9: Example of farrow structure, optimized for variable fractinal delay

where M is the decimation factor and $k \in \{0, 1, 2, ...\}$, is a value which increments throughout the computation of the output. The transfer function of each FIR subfilter, $C_m(z)$ is expressed by

$$C_m(z) = \sum_{k=0}^{N-1} c_m\left(k - \frac{N}{2}\right) z^{-k}. \tag{2.15}$$

While the structure presented in Fig. 2.9 is optimized for interpolation, there are some modifications, explained in detail in [17] which apply to decimation. This structure has the disadvantage of having to include all subfilters to compute all coefficients c_n, it is one of the easiest methods to allow variations on sample rate conversion ratios, and occupies a low area, which is mainly taken by the delay elements.

Chapter 3

Circuit Design

The symbol and interface signals of the sample rate converter design is presented in Fig. 3.1. The data input and output signals are x and y, respectively, and the sample rate clocks are x_clk and y_clk, respectively. The definition of all interface signals is presented in Table 3.1.

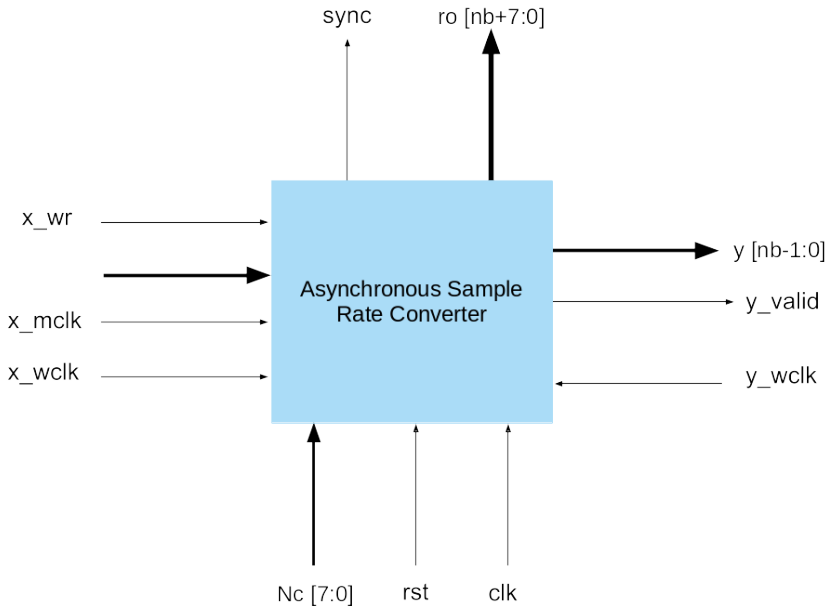

Figure 3.1: Asynchronous sample rate converter interfaces block diagram.

The ASRC is divided in three modules: the Input Data Memory, the Ratio Estimator and the Resampler. Additionally, a simple positive edge detector circuit is needed to start the execution of some modules. A block diagram of the ASRC with its three modules is presented in Fig. 3.2.

The core works in three different clock domains and all figures present the signals in three different colors, with green arrows indicating signals in x_mclk's domain, blue arrows for signals in y_wclk's domain, and black arrows for signals in clk's domain. This is important as wherever there is a clock domain crossing there needs to be a suitable synchronizer circuit.

Table 3.1: List of asynchronous sample rate converter interface signals.

Name	Direction	Description
clk	Input	System clock input.
rst	Input	Active high synchronous reset.
Nc [7:0]	Input	Number of input audio channels.
x [nb-1:0]	Input	Input sample.
x_wclk	Input	Input word clock, with frequency equal to the input sample rate.
x_mclk	Input	Input master clock. Should be 2^N times faster than x_wclk, where N is an integer greater than Nc.
x_wr	Input	Input sample write enable.
y [nb-1:0]	Output	Output sample.
y_wclk	Input	Output word clock, with frequency equal to the output sample rate.
y_valid	Output	Output sample valid. Should be used as an enable to an output sample register.
ro [nb+7:0]	Output	Sample rate convertion ratio signal. Represented in format 8Qnb.
sync	Output	Active high when the converter has stabilized.

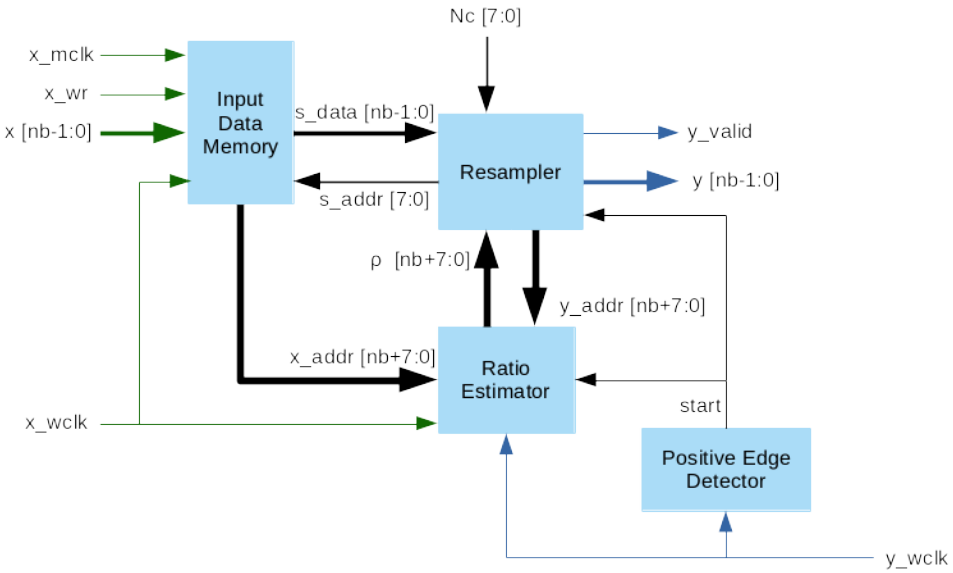

Figure 3.2: Asynchronous sample rate converter top block diagram.

3.1 Data Memory

To store the input samples, a data memory is used. This data memory should be a dual port Random Acess Memory (RAM) capable of working at two different clock domains. The block diagram of this memory and the auxiliary components to access is presented in Fig. 3.3.

In the input clock domain, the samples x are stored in the position given by x_waddr at the rate of x_mclk. The write address counter x_waddr is incremented at x_mclk's rate, whenever x_wr is active. In the time domain of the system clock, the samples are read by accessing the position given by s_addr. For each output clock pulse, the input samples needed to perform the filtering are read from the memory at the system clock rate. For a multiple channel input, to obtain the next sample of a certain channel,

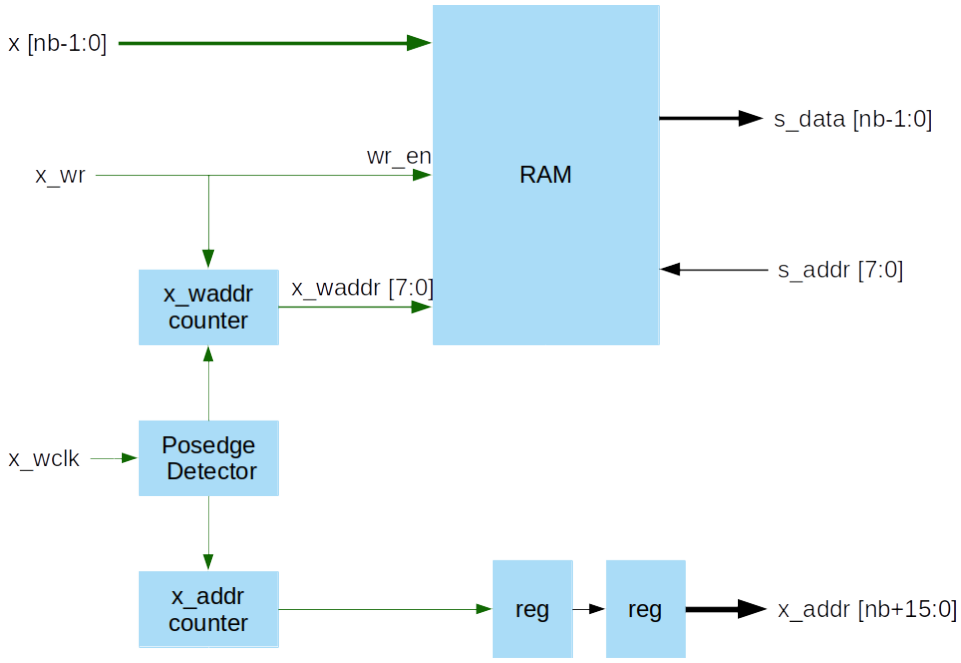

Figure 3.3: Data memory module block diagram.

the address $s\_addr[n+1]$ is given by

$$s\_addr[n+1] = s\_addr + Nc. \tag{3.1}$$

Since the accumulators do not stop or reset during the execution of the conversion, they wrap around creating a circular memory. When there is a stream of input samples, the newer ones will overwrite the older ones which will not be used anymore in the time-finite fractional delay filter. On average the write and read pointers increment at the same rate, which is guaranteed by the Ratio Estimator block.

This module also contains an additional counter, x_addr. This counter is similar to the x_waddr counter, with the difference that it is not controlled by x_wr. This counter is used by the ratio estimator described in Section 3.2, as an parameter used to adjust the sample rate conversion ratio.

3.2 Ratio Estimator

Since the sample rate converter is asynchronous, the frequency of the input and output clocks can change over time. Furthermore, their frequencies can also take any value in the supported range. The frequencies of the data clocks are unknown by the core but their ratio must be computed. The Ratio Estimator is the module that computes the ratio between the periods of the data clocks, and its block diagram is shown in Fig. 3.4.

The Period Meter block uses a counter to obtain a relative value T_{count} of the period of the data clocks. Since the counter is incremented at the system clock rate f_{sys_clk}, an approximate value of the data clock period T_d is given by

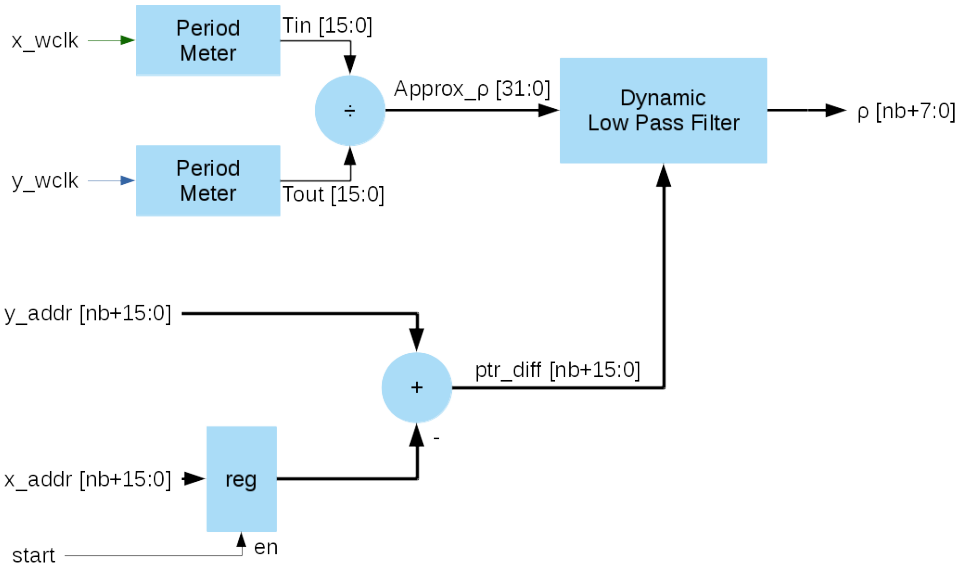

Figure 3.4: Ratio Estimator module block diagram.

$$T_d = \frac{T_{count}}{f_{sys\_clk}}. \tag{3.2}$$

The ratio ρ between the input and output clock frequencies f_{in} and f_{out} is defined by

$$\rho = \frac{f_{out}}{f_{in}}. \tag{3.3}$$

Combining Equation (3.2) and (3.3), an approximate value of ρ can be computed by

$$\rho = \frac{T_{count\_in}}{T_{count\_out}}. \tag{3.4}$$

The value of ρ can be computed in hardware without knowing any clock frequencies. However, since the counter is only able to count integer values, the periods of the input and output clocks are imprecise. For the same clock frequency the value T_{count} can vary each time it is determined, with an absolute error of 1 unit. It should be noted that the resolution of the counter increases with the frequency of the system clock: a higher system clock frequency leads to a lower error in T_d.

These errors lead to an imprecise value of ρ, leading not only to an error in the conversion done in the resampler module, but also to having the read and write pointers in the data memory module incrementing at different average speeds, which eventually may cause the buffer to underflow or overflow. The dynamic low pass filter is the block which solves this problem, as it stabilizes the deviations of the approximate result of ρ, obtained according to Equation (3.4), obtaining the mean value instead. The structure of the dynamic low pass filter is not given at this stage. It will simply be stated that when there is a high deviation of the approximate ρ, the filter's cuttoff frequency rises, as it assumes there was a change of the input and/or output clock frequencies. When the deviation lowers, the filter's cuttoff frequency lowers as well, being less sensitive to the small deviations of the approximate ρ caused by quantization and period counting errors. A correction parameter that takes into account the difference

between the read and write pointers of the input data memory adjusts the value of ρ. The read pointer increment rate is changed to keep the difference between the pointers constant, for example, ensuring that the read and write pointers have a distance equal to half of the memory's size.

3.3 Resampler

The resampler is the main module of the sample rate converter and executes the conversion algorithm. This complex module is split into three submodules: the *address generator*, the *coefficient memory* and the *multiply-accumulate* submodule, which are explained in the next subsections. A block diagram of the resampler is presented in Fig. 3.5.

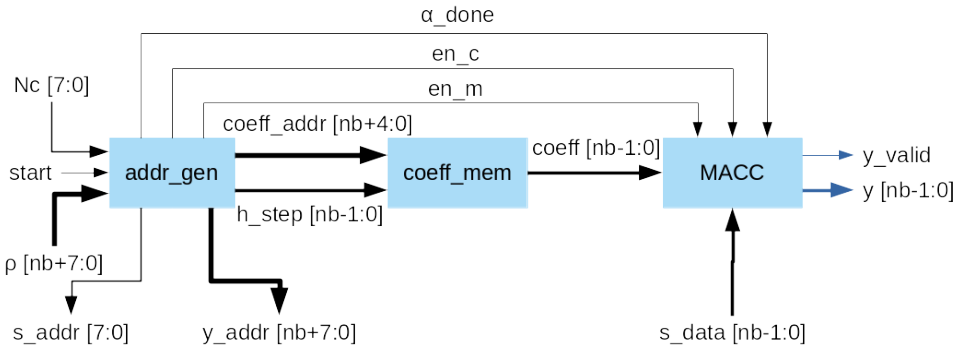

Figure 3.5: Resampler module block diagram.

3.3.1 Address Generator

The address generator is the submodule responsible for computing the addresses of the input samples needed to compute the output sample. As explained in Sections 2.2 and 2.3, there are two main steps needed to compute an output sample: (1) center the filter at the time of output sample to be computed; (2) compute the input samples and correspondent filter coefficients and perform the filter convolution operation.

To center the filter at the time of the output sample to be computed, an alpha computation submodule is used. This submodule computes the time of the current output sample to be computed in the form of a fractional address y_addr, which is incremented by a factor $1/\rho$ at the output sample rate.

Additionally, there is the need to know the distance in time (normalized to the input's sample rate) between the current output sample and the closest input sample, as this distance defines the first coefficient to be used for each side of the symmetric filter. This distance is called α and is computed by assuming that there is an input sample $x[n]$ for all integer values of n, and that $1 - \alpha$ is the fractional distance between the output sample and the closest input sample.

Finally, to adapt the filter's shape, ensuring that it has a cutoff frequency defined by Equation (2.2), a parameter h_step is defined by normalizing the cutoff frequecy to the input sample clock frequency:

$$h\_step = min(1, \rho). \tag{3.5}$$

With the value of h_step and α, as well as the integer part of y_addr, it is possible to determine all of the needed addresses for the input samples and correspondent coefficients.

To get the input samples' addresses a simple counter submodule (*s_addr computation*) is used. The integer part of y_addr is used as the base of the address computation. Knowing that consecutive input samples of the same channel have the distance defined by Equation (3.1), it suffices to accumulate this distance from the offset of the current channel to get all needed input sample addresses that come after (or to the right of) the output sample instant. When the right side is completed, the left side is computed by negative accumulations from the channel offset, getting the addresses of all input samples before the output sample instant. When both sides are done, the channel offset is incremented and the process is repeated for the next channel. This means that one channel is computed at a time, allowing the use of a single multiply-accumulate unit to compute the output samples for all channels.

To get the coefficients' addresses, another accumulator is used. The value of α is used as the initial value in order to align the input sample to its filter coefficient. Since an increment of one input sample corresponds to a increment of h_step in the coefficient function, h_step is used as the accumulation value. These accumulations will be done until the filter is no longer defined, which happens when the accumulator reaches or overcomes the final address of the coefficients memory. After these accumulations are done, there is a side switch, and the accumulations restart, with the complement $1 - \alpha$ as the initial value and the same increment of h_step. This module also produces two flags, en_m and en_c, to enable accumulations and reset the accumulator when the channel is done, respectively.

3.3.2 Coefficient Memory

To get the filter's coefficients, the simplest way is to have a lookup table with them already precomputed using a Read Only Memory (ROM). Because of the high precision needed the ROM would be too large. To solve this problem, a simple linear interpolator is used. Assuming that a certain coefficient $h[i + \Delta]$ is needed, $h[i]$ and $h[i + 1]$ exist in the lookup table, and Δ is a fractional positive distance to i, the coefficient is obtained by

$$h[i + \Delta] = h[i] + \Delta(h[i + 1] - h[i]). \tag{3.6}$$

3.3.3 Multiply-Accumulator

This submodule is a simple accumulator, done with a register and an adder. Its input is the product between the coefficient and the input sample as defined by Equation (2.11). Its initial value is set by directly loading (not accumulating) the first product.

The final register, which produces the output y, is enabled only when all accumulations have finished. This ensures that y never has an intermediate value. This output is updated at the output sample rate but lives in the system clock domain. Therefore a synchronizer is further needed to convert it to the y_wclk clock domain.

Chapter 4

Current Simulation and Implementation Status

Since this thesis consists on a hardware implementation of an algorithm, it requires a set of tools to properly test its functionalities. The first step is by simulation. As a prototype of the sample rate converter was already developed and simulated, some preliminary results related to this step are already shown in this chapter. The second step is the implementation of the core in a FPGA, including support for its use in a System-on-a-Chip.

4.1 Simulation

To simulate the hardware, Cadence's *NCsim* simulator is used. To test the algorithm, two scripts are devloped: one that generates the input signal and one that evaluates the converter's output signal.

4.1.1 Filter Coefficients And Input Signal Generation

To generate the output signal, the sample rate converter core needs the fractional delay filter coefficients and the input signal which are generated by a script described in this section.

To generate the coefficients, the script receives three values: $filter\_nzeros$, $filter\_nfrac$, and H_bits, which give the number of zeroes of the sinc function, the number of address bits to address the filter's memory, and the number of bits to quantify the value of the coefficients, respectively. With these values, the script generates an ideal sinc function and truncates it using window function. Since the result is a symmetric function, only half the window values are taken to fill the coefficient memory. A snippet of the *Octave* code of this script is shown in Listing 4.1.

Listing 4.1: Coefficients generation script

```
function filter_tab = getCoeffROM(filter_nzeros, filter_nfrac, H_bits)
    filter_t_res = 1/2**filter_nfrac;
```

```
    filter_t = 0: filter_t_res : filter_nzeros/2 -  filter_t_res; %
        normalized positive time axis
    window = kaiser(2*length(filter_t)-1, 12.4)';
    window = window(length(filter_t): 2*length(filter_t)-1); %take half
        window
    filter_tab = (2^(H_bits-1)-1)*sinc(filter_t) .* window; %build one
        sided impulse response filter table
    filter_tab = do_2s_comp(round(filter_tab), H_bits);
    return
    endfunction
```

To generate the input samples, the script generates a sinusoidal signal and samples it at the input sample rate. To do this, the script needs the sinusoid's frequency $Ftest$, the duration of the signal in input samples TD, the number of bits to quantize the samples H_bits, and the input signal's sample rate Fin. With these values a vector with all input samples is generated. Regarding the amplitude of the input signal, it is set slightly below unity to prevent overflows during processing. An example script is outlined in Listing 4.2.

Listing 4.2: Input generation script

```
function x = genInput(Ftest, TD, H_bits, Fin)
    %compute test duration in seconds
    td = TD/Fin;
    %compute input wave x
    t_in = 0: 1/Fin : td-1/Fin; %input time axis
    %create test wave with -1dB input gain
    x = sin( 2*pi*Ftest*t_in + pi/4) * 10^(-1/20);

    %quantize x to 1Q23
    x = round( x*2^(H_bits-1) );
    %do 2s complement
    x = do_2s_comp(x, H_bits);
    return
endfunction
```

To apply these signals in hardware simulation, they are written to a file using hexadecimal notation. The simulation testbench loads this file to read the input signal from it. The testbench also loads configuration values such as the number of channels or output signal sample rate.

4.1.2 Output Validation

To test the output signal, a script which computes its spectrum and the total harmonic distortion plus noise ratio is run. This script receives as inputs the array of output samples y, obtained by reading a file produced by the simulation, the original sinusoid's frequency $Ftest$, and the output sampling rate $Fout$.

To prevent spectrum artifacts, an integer number of periods is extracted from y and used by the script. To obtain the signal's spectrum, a fast Fourier transform is used. Then, the total harmonic distortion plus noise ratio is computed by summing the power value of every bin which is not deemed to belong to the original sinusoid signal. A small tolerance around the test frequency is allowed to compensate for the difficult to avoid scattering when computing spectra. An example script is presented in Listing 4.3.

Listing 4.3: Output analysis script

```
function thdn = getTHDN(y, Ftest, Fout)

    Nypts = length(y);
    f_axis = 0: Fout/Nypts :Fout/2-Fout/Nypts;

    % compute bandwidth surrounding the fundamental
    sig_bin_width = 8;

    Y=abs(fft(y))/Nypts;
    tone_idx = round(Ftest/Fout*Nypts)+1;
    signalBins = tone_idx-floor(sig_bin_width/2):tone_idx+floor(
        sig_bin_width/2);
    signalBins = signalBins(signalBins>0); %remove bins lower than 0
    signalBins = signalBins(signalBins<=Nypts/2); %remove bins over
        FFTpoints/2
    s = norm(Y(tone_idx));
    noiseBins = 1: Nypts/2;
    noiseBins(signalBins) = []; %excludes signal bins
    noiseBins(1: floor(sig_bin_width/2)) = []; %excludes DC
    n = norm(Y(noiseBins));
    thdn = 20 * log10(n/s)
    return
endfunction
```

4.1.3 Preliminary Results

A prototype of the design proposed in Chapter 3 was implemented during an internship at IObundle, generating preliminary results. This prototype consists in a sample rate converter with no dynamic ratio

estimator, that assumes that the sample clocks are ideal and have constant frequency. To obtain the ratio between the sample rates, the input and output sample clock periods are measured with the system clock and divided using a hardware divider. The division remainder is accumulated and injected in the quotient when its value equals the value quotient's least significant bit. This is done with the purpose of avoiding the propagation of quantization errors.

The system was simulated using the scripts and techniques explained in this chapter, generating the preliminary results shown in Table 4.1. While for most cases of upsampling and some cases of downsampling the -130 dB THD+N specification is already fulfilled, there are cases where the THD+N is too high, which means the core needs to be debugged and improved. These results tend to occur when the ratio between input and output sample rates is not an integer or the inverse of an integer. For the usual frequencies used in audio applications, this means that this prototype is unable to convert multiples of $8\,\text{kHz}$ to/from multiples of $11.025\,\text{kHz}$, which is of course unacceptable.

Table 4.1: Preliminary results obtained.

Convertion Ratio	**THD+N**
44.1 kHz:48 kHz	$-131.94\,\text{dB}$
48 kHz:44.1 kHz	$-53.45\,\text{dB}$
48 kHz:96 kHz	$-140.44\,\text{dB}$
44.1 kHz:192 kHz	$-131.82\,\text{dB}$
96 kHz:48 kHz	$-143.71\,\text{dB}$
192 kHz:32 kHz	$-46.21\,\text{dB}$

4.2 Implementation In FPGA

After all simulation results fulfill the required specifications, the sample rate converter will be implemented in FPGA.

For the testing an actual system testbench needs to be created to apply real signals to its input interfaces and collect responses from its output interfaces. This testbench is in fact another core called the tester core.

Furthermore, since one of the objective is to make the core commercially viable, the input/output interfaces should be standard, so that it can easily be connected to third party devices. For this purpose the core will feature Advanced eXtensible Interface (AXI) and Inter-IC Sound (I^2S) converters connected to it, and should be tested in a System-on-Chip (SoC).

4.2.1 Interfaces

The list of interface signals can be seen in Table 3.1. Since this core works in 3 different clock domains, there is at least one different interface for each domain. For control operations, an AXI-Lite interface is suitable, which will be running in the system's clock domain,

For the input and output audio samples, there are data streaming interfaces containing signals from which the input and output sample rates can be measured: native interface. The core will use simple

parallel bus interfaces for this purpose and, optionally, for greater interoperability input and output I^2S interfaces will be supported. The splitting of the signals and interface attribution is presented in Table 4.2.

As a simpler alternative to the I^2S interface, a native parallel interface is also developed and used. This interface uses First In, First Out (FIFO) structures to send data frames, converting bit streams into a native bus containing the data samples.

Table 4.2: Attribution of interfaces to asynchronous sample rate converter signals.

Interface	Name	Direction
General	clk	Input
	rst	Input
AXI-Lite	Nc [7:0]	Input
	ro [nb+7:0]	Output
	sync	Output
I2S (Input)	x [nb-1:0]	Input
	x_wclk	Input
	x_mclk	Input
	x_wr	Input
I2S (Output)	y [nb-1:0]	Output
	y_wclk	Input
	y_valid	Output

4.2.2 Implementation And Testing In A System-on-Chip

To test the versatility of the core, as well as to create a test environment in the FPGA, the sample rate converter will be implemented in a SoC which is currently being developed in another student's thesis, and uses a picoRV32 [18] RISC-V processor.

The SoC will include the core as a peripheral, stimulate its inputs and observe its outputs. The SoC will have an AXI-Lite master interface to control the core. Additionally, it will have two I^2S interfaces, one to send the input audio samples, and the other to receive the output audio samples.

To test the core in the SoC, a Personal Computer (PC) will be used. A simple program reads the samples from a .wav file, sends them to the FPGA board through an Ethernet interface. Using the same communication infrastructure, the PC will receive back the core output data from the board and run the analysis scripts on them.

The firmware in the SoC is also simple. It receives the data from the communication interface and forwards it to the core through its data output interface. To receive the core's output, it uses its input data interface, eventually buffering it. When all samples have been processed and received, the SoC will send them to the computer through the communication interface.

Chapter 5

Conclusions

In this report, a study of the most used audio Asynchronous Sample Rate Conversion (ASRC) algorithm and a preliminary hardware implementation is presented. The circuit is described in Verilog and will be prototyped in FPGA. The objective is to design a multi-channel sample rate converter Intellectual Property (IP) module which can be integrated in System on Chip. An ASRC is a rather complex circuit and only major semiconductor players such as Cyrrus Logic, Analog Devices and Texas Instruments have Integrated Circuit (IC) solutions available in the market. To the best of our knowledge the only existing ASRC IPs in the market, the CWda5x family, is made available by Coreworks, SA. According to the CWda5x documentation, there is much room for improvements, which are studied in detail in this thesis.

ASRC algorithms require high numeric precision to ensure the quality of the audio signal, given the high sensitivity of the human auditory system. There are two major sources of imprecision: (1) the precision of the time measurements needed to determine the sampling instant; (2) the precision of the filter coefficients, obtained from an ideal low pass filter impulse response (sinc function), since they vary with the exact instant when the output sample is computed.

Once the output sample issue time is determined, the computation of the output sample may proceed synchronously. As a consequence, synchronous sample rate converter IPs can easily be derived from the present work, since this type of converter is a sub part of the a full ASRC solution. Synchronous sample rate converters find many applications in digital audio to operate on stored signals for which the sample rate is known *a priori*.

The high precision demands of ASRC algorithms tends to undesirably increase the hardware area occupation and computation time. The solution which will be further explored in the following master's dissertation will need to compromise between signal fidelity and resources usage.

5.1 Achievements

A research about sample rate converters has been conducted, emphasizing the distinction between synchronous and asynchronous converters. In fact synchronous sample rate converters are a sub part

of asynchronous sample rate converters in the sense that they do not need to dynamically determine the sample rate of the input signal, nor do they need to perform dynamic computation as they could perform the calculation using predetermined filter coefficients. Note however, that the number of fixed coefficient sets may be rather large if the repetition period (in samples) of the sampling instant relative to the phase of the input signal is large.

Asynchronous sample rate converters involve a more complex algorithm, using more hardware resources to dynamically determine the output sample instants. However, this type of converters is very useful as they can work on input signals for which the sample rate is unknown. They are also able to compensate deviations on the input or output clocks while maintaining the same hardware.

To fully understand the principle on which the algorithm is based, some basic digital signal processing principles are reviewed, with emphasis on digital signal filtering, as well as the implications of sampling and aliasing. This study made it possible to derive the specifications of the digital resampling filter in terms of its type, cutoff frequency and phase response.

To implement the synchronous converter in hardware, some filter structures were analyzed, with emphasis on their advantages and disadvantages. While the Farrow structure is the hardest to implement, it is also the most versatile, and the one which may lead to a smaller total harmonic distortion plus noise ratio, while maintaining a low area and computational cost.

The asynchronous nature of the algorithm dictates the need to work with multiple clock domains, which is as complex as it can get in terms of digital hardware design. In the chosen implementation 3 different clock domains are used: the input, output and processing clock domains. Different types of hardware synchronizers using registers and dual-port memories are employed.

5.2 Future Work

The work to be developed in the thesis consists in creating a sample rate converter design which is able to dynamically convert sample rates which may vary over time. The core should occupy as less hardware resources as possible, and have an audio fidelity on par with the current integrated circuit implementations. Furthermore, it shall be possible to configure the core using the AXI-Lite interface, as well as sending and receiving audio samples using a parallel/I2S interface. With this implementation it should be possible to implement the core in FPGAs and SoCs. The work plan is shown in Table 5.1, together with the scheduling for each task.

The first step of this work is the improvement of the prototype, as it already contains the fractional delay filter implementation, as well as most of the modules defined in Chapter 3. This step is considered finished when the prototype is able to output a signal with a total harmonic distortion and noise ratio equal or lower than $-130\,\mathrm{dB}$ for all combinations involving the commonly used sample rates outlined in Table 5.2.

With a prototype capable of fulfilling the output fidelity specification for a synchronous ASRC, the next step consists in adding the ratio estimator, making it possible to convert sample rates dynamically. This means supporting sample rate changes over time, while maintaining the output fidelity. This step

Table 5.1: Planning of the work that will be performed in the thesis

Task	**Schedule**
Improve sample rate converter prototype, fulfilling specifications for main sample rates	17/02 - 15/03
Implement ratio estimator into sample rate converter	16/03 - 12/04
Insert pipelines and control units into core to increase its system clock frequency	13/04 - 19/04
Optimize core to decrease FPGA resources usage	20/04 - 03/05
Design and implement interfaces	04/05 - 10/05
Test and debug core in FPGA	11/05 - 31/05
Write the Thesis report	1/06 - 19/07

Table 5.2: Commonly used sample rates in audio applications and their particular uses

Sample Rate	**Use**
8.000 kHz	Telephones/walkie-talkies.
11.025 kHz	Lower-quality PCM, MPEG audio and analizers of subwoofer bandpasses.
16.000 kHz	VoIP and VVoIP applications.
22.050 kHz	Lower-quality PCM, MPEG audio and low frequency energy analizers.
32.000 kHz	miniDV camcorders, and some video tapes. Also used for high-quality wireless microphones.
44.100 kHz	Audio CDs, most used with MPEG-1 audio (VCD, MP3) covers the audible bandwidth (up to 20 kHz.
48.000 kHz	Professional digital video equipment and consumer video formats, like digital TV, DVD and films.
88.200 kHz	Some professional recording equipment that targets CD, such as mixers or equalizers.
96.000 kHz	High definition DVD and blu-ray audio tracks.
176.400 kHz	High definition CD recorders and other applications targeting CD.
192.000 kHz	Professional video equipment targeting high definition DVD and blu-ray audio tracks.

should also include the optimization of this module to ensure that it stabilizes as fast as possible. After this step, the prototype can be considered complete, although poorly optimized.

With an inefficient prototype, the next two steps are focused on optimization. Firstly, pipelines will be added, as well as other timing closure oriented optimizations, in order to enlarge the range of the system clock (processing) frequency as much as possible for maximum usage flexibility and capability to support multiple channels.

The second optimization step consists in optimizing the FPGA resource usage, which indirectly leads to a compact ASIC implementation. Firstly, the widths of the intermediate buses will be reduced as much as possible. This will create a trade-off with the output's harmonic distortion and noise ratio, since narrower buses will raise the quantization errors. The $-130\,\mathrm{dB}$ will be used as a reference to limit the reduction in the width of the buses. Afterwards, some further optimizations will be done, through resource sharing and Verilog code optimization. These optimizations should be done without impacting the algorithm's execution time, as computing less samples per output sample clock cycle leads to supporting less audio channels.

After all optimizations are done, the core needs to be implemented and tested in a FPGA. To properly conduct the tests, and also to facilitate the integration in any system, standard interfaces will be designed and implemented, using the AXI-Lite interface for control and the I^2S interfaces for the audio. With the standard interface in place, the core will be integrated in a System-on-Chip containing a RISC-V processor.

With the sample rate converter fully implemented and integrated, the final step is to conduct system-level tests using both test signals and audio files send from a PC. After system test, the sample rate converter IP core will be deemed complete, and its documentation including the thesis to be written will be finalized.

Bibliography

[1] *192 kHz Stereo Asynchronous Sample Rate Converter*. Analog Devices, March 2003. Rev. A.

[2] *4-Channel, Asynchronous Sample Rate Converter*. Burr-Brown Products from Texas Instruments, June 2004. Rev. B.

[3] *Multi-Channel Audio Sample Rate Converters*. coreworks, June 2016.

[4] R. Crochiere and L. Rabiner. *Multirate Digital Signal Processing*. Prentice-Hall Signal Processing Series: Advanced monographs. Prentice-Hall, 1983. ISBN 9780136051626.

[5] S. K. Mitra and J. F. Kaiser, editors. *Handbook for Digital Signal Processing*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1993. ISBN 0471619957.

[6] P. Beckman and T. Stilson. An efficient asynchronous sampling-rate conversion algorithm for multi-channel audio applications. In *AES Convention Papers Forum*, number 6553, October 2005.

[7] P. J. Kootsookos and R. C. Williamson. Fir approximation of fractional sample delay systems. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(3):269–271, March 1996. doi: 10.1109/82.486473.

[8] C. Tseng and S. Lee. Design of fir fractional delay filter based on maximum signal-to-noise ratio criterion. In *2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pages 1–8, Oct 2013. doi: 10.1109/APSIPA.2013.6694180.

[9] V. Valimaki and T. I. Laakso. Principles of fractional delay filters. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 6, pages 3870–3873 vol.6, June 2000. doi: 10.1109/ICASSP.2000.860248.

[10] A. Yardin, G. D. Cain, and A. Lavergne. Performance of fractional-delay filters using optimal offset windows. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 2233–2236 vol.3, April 1997. doi: 10.1109/ICASSP.1997.599495.

[11] S. CHARANJIT, M. Patterh, and S. Sharma. Efficient implementation of sample rate converter. *International Journal of Advanced Computer Sciences and Applications*, 1, 01 2011. doi: 10.14569/IJACSA.2010.010606.

[12] E. Hogenauer. An economical class of digital filters for decimation and interpolation. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 29:155 – 162, 05 1981. doi: 10.1109/TASSP.1981.1163535.

[13] Y. Mori and N. Aikawa. Kernel using piecewise nth polynomials for rate converter. In *Proceedings of the 6th Nordic Signal Processing Symposium, 2004. NORSIG 2004.*, pages 57–60, June 2004.

[14] N. Aikawa and Y. Mori. Kernel with block structure for sampling rate converter. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, volume 6, pages VI–269, April 2003. doi: 10.1109/ICASSP.2003.1201670.

[15] C. W. Farrow. A continuously variable digital delay element. In *1988., IEEE International Symposium on Circuits and Systems*, pages 2641–2645 vol.3, June 1988. doi: 10.1109/ISCAS.1988.15483.

[16] M. Blok and P. Drozda. Variable ratio sample rate conversion based on fractional delay filter. 2015.

[17] D. Babic, J. Vesma, T. Saramaki, and M. Renfors. Implementation of the transposed farrow structure. In *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, volume 4, pages IV–IV, May 2002. doi: 10.1109/ISCAS.2002.1010374.

[18] C. W. et al. Picorv32 - a size-optimized risc-v cpu. `https://github.com/cliffordwolf/picorv32`, 2015.