Master Thesis

Feng Wang

A Self-tuning Flexible Sample Rate Converter using a

Linear Interpolator on the

TI DSK 6713 board

Feng Wang

A Self-tuning Flexible Sample Rate Converter using a Linear Interpolator on the TI DSK 6713 board

Master thesis based on the examination and study regulations for the
Master of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner : Prof. Dr. Ulrich Sauvagerd
Second examiner : Prof. Dr. Hans Peter Kölzer

Feng Wang

Title of the Master Thesis

A Self-tuning Flexible Sample Rate Converter using a Linear Interpolator on the TI DSK 6713 board

Keywords

Digital Signal Processor, FIR lowpass filter, Interleaver, Lagrange Interpolator, Linear Interpolator, Matlab Simulation, Sampling Rate Converter, Window method

Abstract

This master thesis constructs a "Sampling Rate Converter" which can deal with arbitrary sampling rate conversion between input and output. This ratio can be rational, irrational or even be slowly time varying.

Linear interpolator is used to do the interpolation. FIR filters are used to remove the distortion introduced by SRD or SRI. These FIR filters are implemented by window function. Polyphase structure of a FIR filter is used to derive an efficient structure. All the real-time implementations are proved by the Matlab simulation results. A TI DSK C6713 board is used as the hardware to realize the real system. At last an alternative interpolation method Lagrange interpolation is explained in a theoretical level. A large number of figures which help the reader understand the complex theories involved in this thesis are the specialty of this thesis.

Feng Wang

Thema der Masterarbeit

Eine sich selbst einstellender flexibler Abtastratenumsetzer unter Verwendung eines linearen Interpolators auf einem TI DSK 6713(Entwicklungs-) board

Stichworte

Digitaler Signalprozessor, FIR Tiefpassfilter, Lagrange Interpolator, Linearer Interpolator, matlab Simulation, Abtastrantenumsetzer, Fenster Methode

Kurzzusammenfassung

Diese Masterarbeit behandelt (dis Konstruktion) eines "Sampling Rate Converters", welcher willk ü rliche Verhätnisse zwischen Ein- und Asugansabtastrate verarbeiten kann. Diese Verältnisse können rational, irrational order zeitlich veränderbar sein.

Zum Zwecke der Interpolation wird ein Linearer Interpolator verwendet. Störungen, hervorgerufen durch den SRD order SRI, werden durch FIR Filter entfernt. Die Implementierung der Filter basiert dabei auf einer Fensterfunction. Polyphase-Strukturen werder genutzt, um eine effiziente Struktur zu erreichen. Die gesamte Echtzeit-Implementierung wurde verifiziert mit den Matlab Simulationsergebnissen. Ein TI DSK C6713 (Entwicklungs-) Board wird benutzt um das System zu realisieren. Abschließend wird die Lagrange interpolationsmethode theoretisch als alternative erläutert. Eine Vielzahl von Abbildungen, die dem Leser helfen sollen, die komplexe Theorie zu verstehen, sind die Besonderheit dieser Ausarbeitung.

Content

Glossary

A/D—Analog to Digital

ALU—Arithmetic Logic Units

CD—Compact Disc

CCS —Code Composer Studio

D/A—Digital to Analog

DSP—Digital Signal processor

FIR—Finite Impulse Response

HiFi —HIgh-FIdelity

IDE—Integrated Development Environment

LI—Linear Interpolator

LPF—Lowpass Filter

LTD—Linear Time-Invariant

McBsp—Multi-channel Buffer serial port

SRC —Sampling Rate Converter

SRD—Sampling Rate Decreaser

SRI—Sampling Rate Increaser

SDRAM—Synchronous Dynamic Random Access Memory

THD—Total Harmonic Distortion

VOIP—Voice Over Internet Protocol

WLIW—Very Long Instruction Word

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Compare to an analog system, a digital system has a lot of advantages. They are small, cheap, programmable, reusable and capable of complicated processing. Since the last three decades, digital signal processing has been well established and developed. This can be proved by thousands of literatures concerning on that subject in both international organizations EURASIP and IEEE. Accompany with these theories, a large number of real products come to world such as PC, DVD, Digital TV and so on. These things make our world digital.

In the beginning 21th century, a lot of new applications based on digital signal processing in the range of audio and video are emerged. People can watch IPTV, listen to the Radio and make a VOIP call to a normal fixed user in one computer. One common problem of such applications is that the system components of different sampling frequencies have to communicate with each other. A real example may be helpful to understand this concept. Compact Disk (CD) is an audio signal digitally stored on a small optical disc. The stereo is sampled at a sampling frequency of 44.1 kHz. In a modern HiFi audio system, a sampling frequency of 48 kHz is required. If one wants to play CD in a HiFi system, a conversion of sampling frequency from 44.1 kHz to 48 kHz is needed. For this reason, a sample rate converter (SRC) will be used in a HiFi system. A SRC can change the sampling frequency of input signal and make it suitable for the system. It makes possible those digital systems which have different sampling rate to communicate. Interpolating technology and digital filters are the key points of it. Interpolating means use mathematical ways to predict unknown samples between the original samples. Filters are used to remove the image created by interpolating in frequency domain.

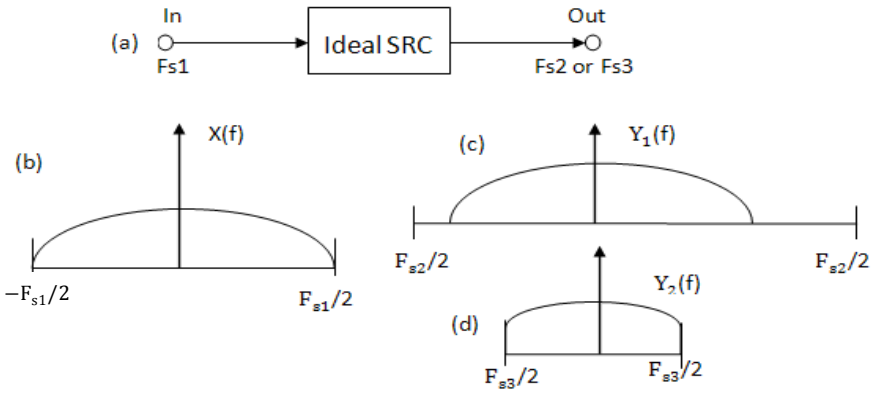


Fig 1.1 An ideal SRC with input spectrum X(f) and output spectrum $Y_1(f) or\ Y_2(f)$

The SRCs proposed in this thesis are all flexible SRCs. They are also needed in many other applications, for instance in the digital transmission systems. An ideal SRC is shown in Fig 1.1(a). Fig 1.1(b) shows the spectrum $X(f)$ of the input signal with sampling frequency F_{s1}. The ideal output spectrum $Y_1(f)$ with sampling frequency F_{s2} is shown in Fig.1.1(c) where $F_{s2} > F_{s1}$.This process is called interpolation. In Fig 1.1(d) the ideal output spectrum $Y_2(f)$ with sampling frequency F_{s3} is shown for $F_{s3} < F_{s1}$. This process is called decimation.

A digital signal processor (DSP) is a specialized microprocessor designed specifically for digital signal processing, generally in real-time computing. Today a DSP has usually a Floating-point unit integrated directly into the data-path and uses a pipelined architecture. This ensures the complex computation used in digital signal processing algorithms.

A few companies have developed perfect software to operate a DSP. Code Composer Studio (CCS) offered by Pacificxu TI is powerful software among them. It has an integrated development environment (IDE). It provides tools for code generation, such as a C compiler, an assembler and a linker. A few debugging features are available, including setting breakpoints, watching variables, viewing memory and registers, mixing C and assembly code, graphing results and so on.

1.2 Objectives

The goal of this master thesis is to develop a flexible self-tuning sampling rate converter with linear interpolator. A TI C6713 board will be used as the hardware. The develop platform is based on CCS Version 3.1. From its title a few main difficulties have been claimed.

Firstly, “flexible” means it can deal with arbitrary ratio between input and output sampling frequency. So the ratio can be either integer, rational or irrational. “Flexible” also means that the SRC should always work even if the given ratio is changed. So the ratio can be a function of time.

Secondly, it is a self-tuning SRC. The input and output sample frequency is unknown. The SRC should detect them automatically. It should also detect the new ratio, when the ratio changes. The on board clock in the DSP will be used to achieve this function.

Thirdly, the SRC uses the linear interpolating technology to predict the unknown samples. Normally in order to get high frequency bandwidth efficiency, a linear interpolator (LI) can’t work alone. A lowpass filter will be added to remove the distortion in the fundamental interval. The attenuation of the filter determines the quality of the SRC. As is known to all, the larger the stop band attenuation is, the narrower the transition

bandwidth will be(the passban and stopband frequency are supposed to be fixed). But narrow transition bandwidth means more filter coefficients are needed. More coefficients will take more time in computation. Hence an efficient structure should be implemented here to avoid time consumption cost by filter calculation.

1.3 Structure of the thesis

The remainder of the thesis is organized as follows:

Chapter 2 “State of Art” gives some basic background acknowledges in dealing with a multirate system. After this chapter one can know what a digital signal is, what are the elements used to increase or decrease the sampling rate of a system, how a lowpass filter looks like in frequency domain and how some system components in a multirate system are interchanged.

Chapter 3 “Interpolator and Decimator” tells how to increase or decrease the sampling frequency of a system by an integer number without effecting the original spectrum. The problems concerning in such processes will be listed and the solution to these problems will be discussed.

Chapter 4 “Rational SRC” shows the way to construct a rational SRC. The design of a rational interpolator in an efficient way will be explained first. The concept of an interleaver will be introduced. Followed by is the design of a rational decimator.

Chapter 5 “Flexible SRC” is based on the design of the previous chapter. A new concept linear interpolator will be given. The efficient implementation of an SRC is the combination of linear interpolator and the structure shown in Chapter 4. A software Matlab will be used to simulate the results.

Chapter 6 “Real-time Implementation” specifies the realization of flexible SRC in a DSP board. Some information about the hardware DSK C6713 board will be given first. Then the realization will be discussed through several parts. Finally the code structure will be explained.

Chapter 7 “Testing’ analyzes the behavior of the system through different testing methods. The results will be analyzed both in time and frequency domain. A new concept called Total Harmonic Distortion (THD) will be introduced here. It is a common standard used in audio analysis.

Chapter 8 “Lagrange Interpolator” describes an alternate way to implement a flexible SRC. It will use Lagrange polynomial to do the approximations which gives a better result than a linear interpolator.

Chapter 9 “Conclusion” makes the summary of the whole work. It describes whether the goal of this thesis is achieved based on the testing results shown in chapter 8. It also includes perspectives which tells how this thesis can be further developed.

2 State of the Art

2.1 Discrete-Time Signals

In digital signal processing, signals are always represented as sequences of numbers called samples. A sample value of a typical discrete-time signal or sequence is denoted as x[n] with argument n being an integer between -∞ and + ∞. It is clear that n is only for integer and not for the non-integer.

There are two ways to get a discrete-time signal. The first is to sample the continuous signal. That means taking a value of the continuous signal every T seconds. This T can be any number greater than zero. The sampling frequency $f_s = 1/T$. This is illustrated in Fig 2.1(a).

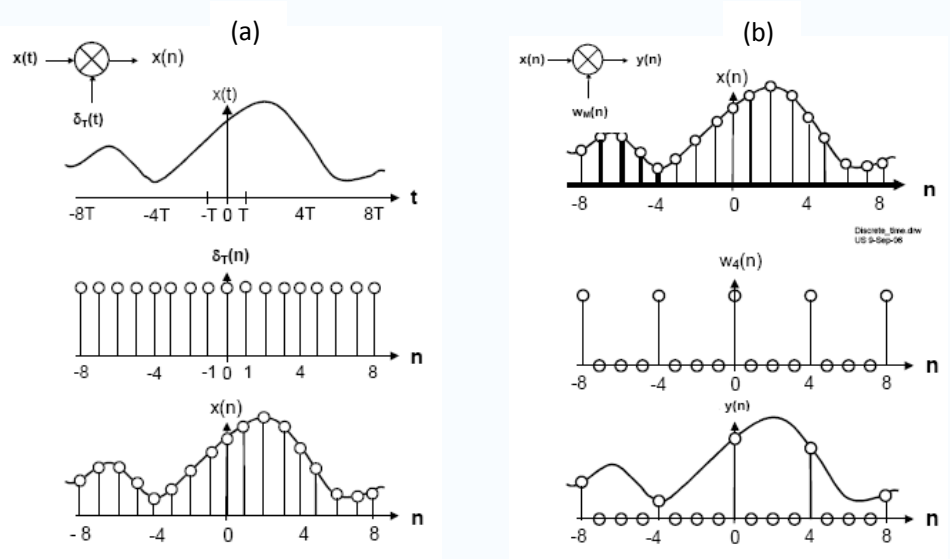


Fig 2.1 Methods to get discrete-time signal

(a) Sampling continuous signal (b) Sampling discrete signal

The second way is to sample a discrete-time signal. Suppose the original signal is x[n]. This is done through taking every M^{th} value of x[n] and setting other values to be zero. Fig 2.1(b) shows the situation where M=4. The original signals are shown on the upper position. A signal called y[n] take every 4^{th} value of it. Note the original sampling frequency has not been changed by the discrete sampling. This process can be represented by the following equation:

$$y[n]=x[n]*w[n]$$

where $$w[n] = \frac{1}{M}\sum_{i=0}^{M-1} e^{\frac{-j2\pi ni}{M}} = \begin{cases} 1, & n = mM \ \ m\ integer \\ 0, & otherwise \end{cases}$$

y[n] has non-zero values only if n is a multiple of M.

2.2 Increase and Decrease Sampling Frequency

Sampling-Rate Decreaser (SRD) and *Sampling-Rate Increaser* (SRI) are the two basic building blocks for changing the sampling frequency of a digital system. Fig 2.2 shows the example of SRD with factor L and SRI with factor K. The factor L and K here are integers. They give the direct information of relationship between input and output signal in time domain. Note they are periodical *time-varying* system, which means they can not be described by an impulse response or a system function. In this section, the input signal will be denoted by $x[nT_1]$ and the output signal will be denoted by $y[nT_2]$, where T_1 and T_2 are the sampling interval of them.

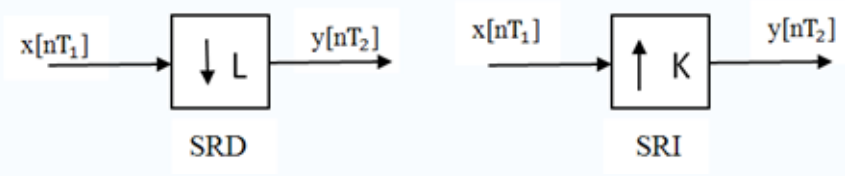


Fig2.2 Building blocks of Sampling-Rate Decreaser (SRD) and Sampling-Rate Increaser (SRI)

The sampling interval T_2 for the SRD is equal to LT_1. Taking out every L^{th} values of $x[nT_1]$ will form the output $y[nT_2]$ of SRD. This can be described as follows:

$$y[nT_2] = x[nLT_1] \tag{2.2}$$

The sampling interval T_2 for the SRI is equal to T_1/K. The SRI insert $(K-1)$ zeros between every two input samples. The SRI will have none-zero value only when n is a multiple of K. This relationship can be described as follows:

$$y[nT_2] = \begin{cases} x[nT_1/K], & n = an\ integer\ multiple\ of\ K \\ 0, & otherwise \end{cases} \tag{2.3}$$

The integer factor L and K that are used in Fig.2.2 are called decimation factor and interpolation factor respectively. An SRD and SRI can also be described in frequency domain.

The discrete-time Fourier transform of SRD can be found from equation (1.47) of [1]

$$Y\left(e^{j\omega T_2}\right) = \frac{1}{L}\sum_{k=0}^{L-1} X(e^{jw\omega T_2/L} e^{-j2\pi k/L}) \tag{2.4}$$

Replace $e^{j\omega T_2}$ with z and obtain the following equation

$$Y(z) = \frac{1}{L}\sum_{k=0}^{L-1} X(e^{-j2\pi k/L} z^{\frac{1}{L}}) \quad (2.5)$$

The k^{th} term in this summation corresponds to a shift version of the input spectrum with a frequency shift of $2\pi k/L$. That means it will repeat the spectrum of the original signal at frequency $2\pi k/L$. The terms for $k \neq 0$ are the aliasing terms, which are unwanted in most cases.

The relation of input and output signal of an SRI can also be described as in frequency domain.

$$Y(e^{j\omega T_2}) = X(e^{j\omega K T_2}) \quad (2.6)$$

Replace $e^{j\omega T_2}$ with z and obtain the following equation

$$Y(z) = X(z^K) \quad (2.7)$$

From equation (2.6) it is obviously that the spectrum of output signal can be found from the DFT transform of the input signal, by replacing T_1 to KT_2. So inserting zeros to the original signal will not change the spectrum. The only thing it has done is rescaling the frequency axis by a factor $1/K$ (The scaling factor of the magnitude is neglected here).

Fig 2.3 gives a clear idea of how SRD and SRI work in frequency domain. Both decimating and interpolating factor are 2. Fig 2.3(a) is the spectrum of the input signal. Fig 2.3(b) is the spectrum of output signal of SRD. The spectrum of the original signal is shifted to $n*Fs/2$, $n \in N$. The solid part in Fig 2.3(b) shows the aliasing effect. The spectrum of the output signal of SRI is shown in Fig 2.3(c). It is clear that frequency axis is scaled by a factor of $1/2$.This cause one image which is represented as a solid triangular in the fundamental interval.

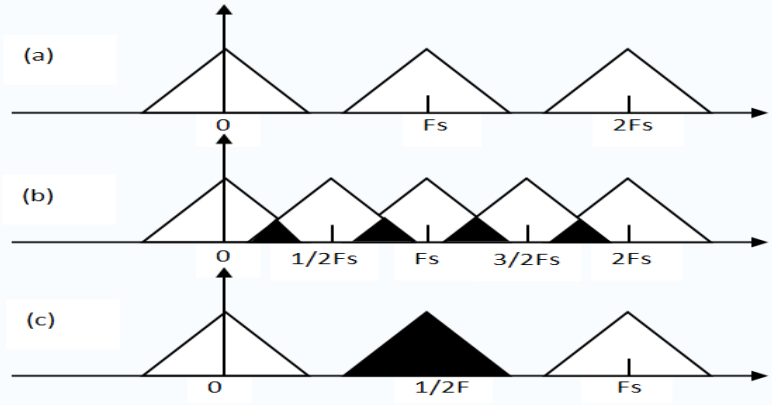


Fig 2.3 Spectrum of signals passing through SRD and SRI

2.3 Lowpass Filters

Filters are the most popular applications implemented in DSP. This introduction gives a clear idea on a theoretical level. A filter can be seen as a black box. It has the following functions:

- Accepting the input signal
- Blocking the pre-specified frequency components
- Passing the original signal minus the blocked frequency components specified above to the output

For example, the sampling rate of a telephone is *8KHz*. So a typical phone line acts as a filter that limits frequencies to a range smaller than *8KHz*. But the frequency range of the human ear is much larger. So audio quality of listening to CD-quality(sampling rate is *44.1KHz*) music over the phone is not as good as directly listening to it. The reason is a lot of high frequency components have been removed by the filter.

A *digital filter* takes a digital input sample and gives a digital output. In a typical digital filtering application, software running on a DSP, which reads input samples from an A/D converter, performs the mathematical manipulations dictated by theory for the required filter type, and outputs the result via a D/A converter.

An *analog filter*, by contrast, operates directly on the analog inputs and is built entirely with analog components, such as resistors, capacitors, and inductors.

Lowpass, highpass, bandpass, and bandstop are the most common filter types. In this thesis, the lowpass filter will be used to implement a SRC. Hence it will be discussed in detail. A lowpass filter passes only low frequency signals (below some specified cutoff frequency) to its output. Therefore it can be used to eliminate high frequencies.

Filters are usually defined by their responses to the individual frequency components that constitute the input signal. A filter's response to different frequencies is characterized as *passband*, *transition band*, or *stopband*.

The word “Pass” means pass them through. Frequency components in the passband are *almost* unchanged. Why “almost” is due to the *ripple* in the passband. Ripple is usually specified as a peak-to-peak level in decibels. It describes how large the filter's amplitude varies within a band. Smaller amounts of ripple represent more consistent response and are generally preferable.

Frequencies within a filter's stopband are, by contrast, highly attenuated.

The transition band represents frequencies in the middle, which may receive some attenuation but are not removed completely from the output signal. Bandwidth of transition band describes how quickly a filter transitions from the passband to the

stopband, or vice versa. The faster this transition transits, the smaller the transition bandwidth is. But this will make the design more difficult.

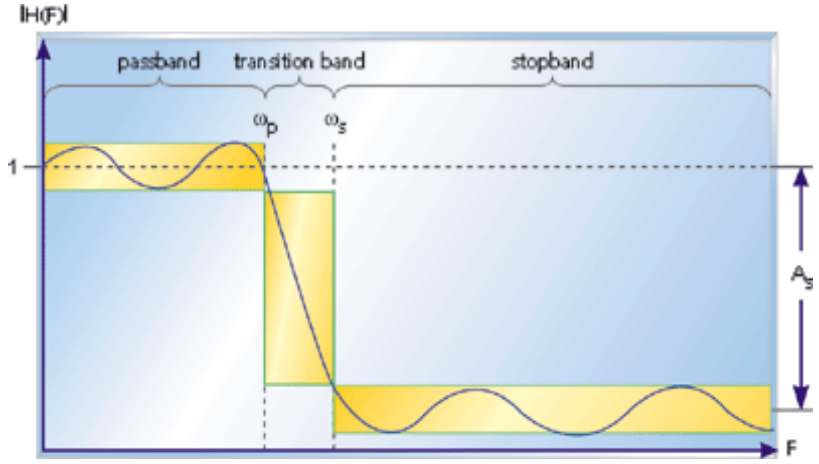


Fig 2.4 Frequency response of a lowpass filter

Fig 2.4 shows these three responses of a lowpass filter. ω_p represents the passband edge frequency, ω_s represents the stopband edge frequency, and A_s is the amount of attenuation in the stopband. Frequencies between ω_p and ω_s fall within the transition band and are attenuated to some lower extends. Both ripple exists within passband and stopband.

A finite impulse response (FIR) filter is a filter structure that can be used to implement almost any sort of frequency response digitally. An FIR filter is usually implemented by using a series of delays, multipliers, and adders to create the filter's output. An FIR filter simply produces a weighted average of its N most recent input samples. All of the magic is in the coefficients, which dictate the actual output for a given pattern of input samples. The real design of an FIR filter will be described later in chapter 6.

2.4 Polyphase decomposition

A straightforward way of implementation a multirate system is usually inefficient. For example, a Finite Impulse Response (FIR) filter with a large rang of coefficients takes a huge amount of time to calculate the filter output. The polyphase decomposition is often used. Assuming such a FIR filter of length N, the system function *H(z)* is given by:

$$H(z) = \sum_{i=0}^{N-1} a_i z^{-i}$$
$$= a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \cdots + a_{N-1} z^{-(N-1)}$$

This equation can be rewritten after *ploy*phase *decomposition* into M different polyphase components $H_{M,j}(z)$:

$$\begin{aligned} H(z) \quad = \quad & (a_0 + a_M z^{-M} + a_{2M} z^{-2M} + a_{3M} z^{-3M} + \cdots) \\ & + z^{-1}(a_1 + a_{M+1} z^{-M} + a_{2M+1} z^{-2M} + a_{3M+1} z^{-3M} + \cdots) \\ & + \cdots \\ & + z^{-(M-1)}(a_{M-1} + a_{2M-1} z^{-M} + a_{3M-1} z^{-2M} + a_{4M-1} z^{-3M} + \cdots) \\ = \; & H_{M,0}(z^M) + z^{-1} H_{M,1}(z^M) + \cdots + z^{-(M-1)} H_{M,M-1}(z^M) \\ = \; & \sum_{j=0}^{M-1} z^{-j} H_{M,j}(z^M) \text{ with } H_{M,j}(z^M) = \sum_j a_{Mj+j} z^{-i} \end{aligned} \tag{2.8}$$

The last summation in equation (2.8) contains at most *(1+N/M)* terms. The *M* functions $H_{M,j}(z^M)$ are called the polyphase components of $H(z)$. The example below shows how a system function $H(z)$ which has a length *N=16* is written into 4 polyphase components:

$$\begin{aligned} H(z) \quad = \quad & (a_0 + a_4 z^{-4} + a_8 z^{-8} + a_{12} z^{-12}) \\ & + z^{-1}(a_1 + a_5 z^{-4} + a_9 z^{8} + a_{13} z^{-12}) \\ & + z^{-2}(a_2 + a_6 z^{-4} + a_{10} z^{8} + a_{14} z^{-12}) + \\ & + z^{-3}(a_3 + a_7 z^{-4} + a_{11} z^{8} + a_{15} z^{-12}) \\ = \; & H_{4,0}(z^4) + z^{-1} H_{4,1}(z^4) + z^{-2} H_{4,2}(z^4) + z^{-3} H_{4,3}(z^4) \\ = \; & \sum_{j=0}^{3} z^{-j} H_{4,j}(z^4) \end{aligned}$$

In the example above, the system function $H(z)$ has been rewritten as the sum of four polynomials in z^4. These polynomials $H_{4,j}(z^4)$ represent so called *comb filters*, which have a fourfold periodic frequency response in their fundamental interval. In order to uniquely characterize them, one can replace z^4 by *z*. Then the so called *polyphase components* $H_{4,j}(z)$ are obtained.

$$H_{4,0}(z) = \sum_{i=0}^{3} b_{4i} z^{-i} \qquad H_{4,2}(z) = \sum_{i=0}^{3} b_{4i+2} z^{-i}$$

$$H_{4,1}(z) = \sum_{i=0}^{3} b_{4i+1} z^{-i} \qquad H_{4,3}(z) = \sum_{i=0}^{3} b_{4i+3} z^{-i}$$

The polyphase description of $H(z)$ as given in the example with M = 4 corresponds to the polyphase structure shown in Fig 2.5.

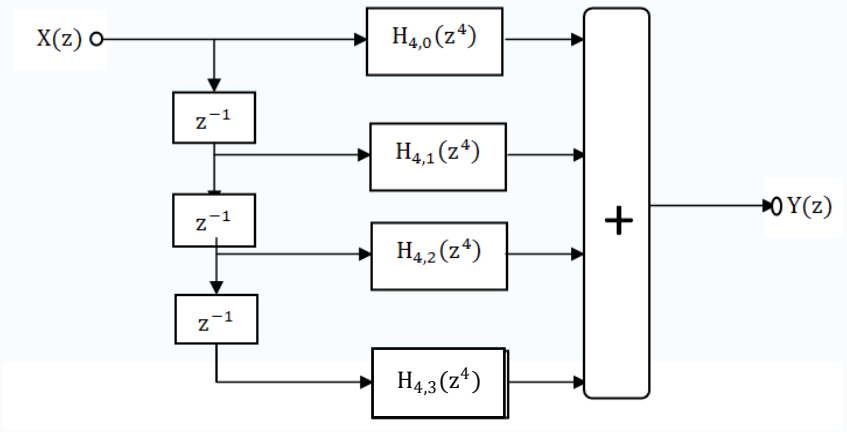


Fig 2.5 The first polyphase structure with M equal to 4

This circuit is called the *first polyphase structure*. How this structure can be used to derive efficient implementations of *decimating filters* will be discussed later. The polyphase description as given in the example for *M=4* also corresponds to an alternative structure, called the *second polyphase structure*. This structure is shown in Fig 2.5. How this structure can be used to derive efficient implementations of *interpolating filter* will also be discussed later.

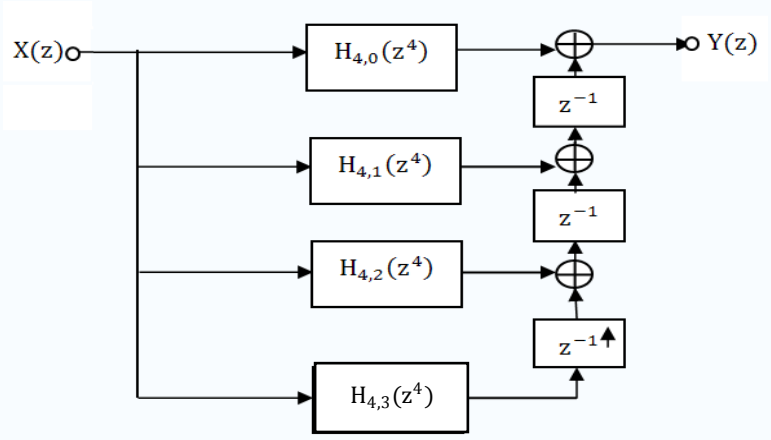


Fig 2.6 The second polyphase structure with M = 4

In fact, the second polyphase structure of Fig 2.6 is the transposed form of the first polyphase structure of Fig 2.5. This can be done by the following two steps.

- Reverse the signal flow,
- Interchange nodes and adders

2.5 Noble Identities

Identities are often used to analyze a complex system in a different view. The direct way of interchanging components of a system makes the whole system more readable. In this thesis two main identities called the *noble identities* will be used. They have been published in the literature [2]. These identities make the analysis of the multirate systems in a simple way. The noble identities describe the combination of a Linear Time-invariant Discrete (LTD) system and an SRD or a SDI.

The Noble identity for decimation can be depicted as in Fig 2.7. It reverses the order of SRD and filter. A SRD with decimation factor L, followed by a digital filter with system function $H(z)$ is identical to a digital filter with system function $H(z^L)$ that is preceded by the same SRD.

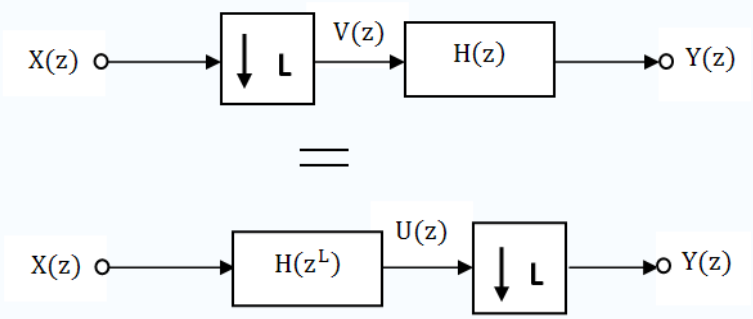


Fig 2.7 Nobel identity for decimation

This idenity can be proved through a mathematical way. For the upper part of Fig 2.7, V(z) can be obtained from equation (2.5),

$$V(z) = \frac{1}{L}\sum_{k=0}^{L-1} X(e^{-j2\pi k/L} z^{\frac{1}{L}})$$

$$Y(z) = H(z)V(z) = H(z)\left\{\frac{1}{L}\sum_{k=0}^{L-1} X(e^{-j2\pi k/L} z^{\frac{1}{L}})\right\}$$

For the lower part of Fig 2.7,

$$U(z) = X(z)H(z^L)$$

$$Y(z) = \frac{1}{L}\sum_{k=0}^{L-1} U\left(e^{-j2\pi k/L} z^{\frac{1}{L}}\right)$$

$$= \frac{1}{L}\sum_{k=0}^{L-1} X(e^{-j2\pi k/L} z^{\frac{1}{L}}) H(e^{-j2\pi kL/L} z^{\frac{L}{L}})$$

$$= \frac{1}{L}\sum_{k=0}^{L-1} X(e^{-j2\pi k/L} z^{\frac{1}{L}}) H(e^{-j2\pi k} z)$$

Because $e^{-j2\pi k} = 1$,

$$Y(z) = \frac{1}{L}\sum_{k=0}^{L-1} X\left(e^{-\frac{j2\pi k}{L}} z^{\frac{1}{L}}\right) H(z)$$

$$= H(z)\left\{\frac{1}{L}\sum_{k=0}^{L-1} X(e^{-j2\pi k/L} z^{\frac{1}{L}})\right\}$$

So the both block give the same output.

In the filter with system function $H(z^L)$, each delay element of the original filter $H(z)$ with delay LT_1 is replaced by a cascade of L delay elements, each with delay T_1. The frequency response of such a filter is periodically repeated L times in the fundamental interval $(-\pi \le \theta \le \pi)$.

The noble identity for interpolation is shown in Fig 2.8. An SRI with interpolation factor K, preceded by a digital filter with system function $H(z)$ is identical to a digital filter with system function $H(z^K)$ that is followed by the same SRI.

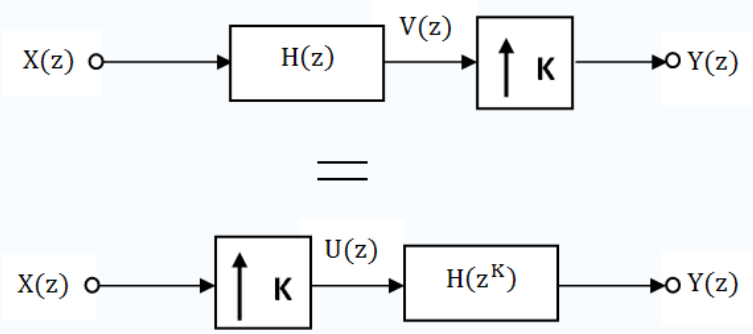


Fig 2.8 Nobel identity for interpolation

This identity can also be proved by mathematical calculation.

For the upper part in Fig 2.8, accoring to equation (2.7)

$$Y(z) = V(z^k)$$

$$V(z) = X(z)H(z)$$

$$\text{So } Y(z) = X(z^k)H(z^k)$$

For the lower part in Fig 2.8,

$$U(z) = X(z^k)$$

$$Y(z) = U(z)H(z^k) = X(z^k)H(z^k)$$

3 Interpolator and decimator

After learning the theoretical concepts of multirate system in the previous chapter, the real implementation of the interpolator and decimator will be described in this chapter. Section 3.1 deals with the interpolator which has a fixed interpolation factor and the efficient way of design such an interpolator. Section 3.2 deals with the decimator which has a fixed decimation factor. Like in section 3.1, an efficient design of decimator will also be explained.

3.1 Interpolator

As is described in section 2.2, an SRI will introduce image in the fundamental interval (see Fig 2.3(c)). These images will cause distortion in time domain. For an interpolator, a low pass filter will usually be used to remove the image. A straight forward way to implementing an interpolator is shown in Fig 3.1. In this figure, the interpolation factor is a fixed number equal to 6.

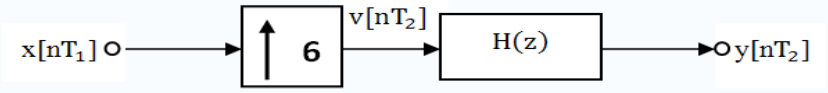


Fig 3.1 Straight forward way of design for Interpolator with interpolation factor 6

The SRI introduces 5 zeros between every two input signals $x[nT_1]$. The interpolated signal $v[nT_2]$ is filtered by a lowpass filter $H(z)$. In Figure 3.2, the spectrums occurring in this interpolating process are drawn schematically.

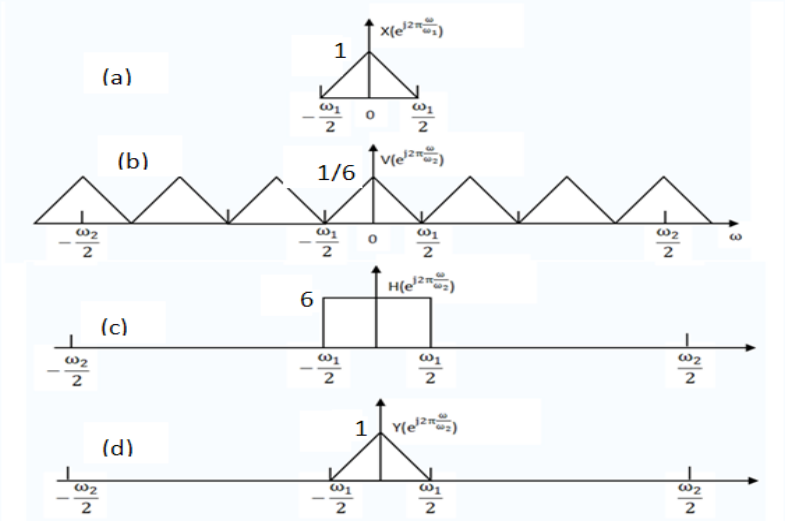


Figure 3.2 The spectrum of ideal interpolator with interpolation factor 4

The spectrum of the input signal is depicted in Fig 3.2(a). Here only the pecturm in fundamental interval is drawn. The magnitude of the spectrum is equal to 1. Fig 3.2(b) plots the spectrum of $V(e^{j2\pi\frac{\omega}{\omega_2}})$ with $\omega_2 = 4\omega_1 (\omega_1 = \frac{2\pi}{T_1})$. The 5 spectral images of the input spectrum that are created by the SRI in the fundamental interval $-\frac{\omega_2}{2} \leq \omega \leq \frac{\omega_2}{2}$ can clearly be seen. Because of the SRI, the magnitude here is scaled by 1/6. For a perfect interpolator, the filter $H(z)$ should be an ideal lowpass filter with cut-off frequency equal to $\frac{\omega_1}{2}$. This is shown in Fig 3.2(c). Note the passband gain should not be 1. It should be equal to the interpolation factor. (More explanation on this point can be found in section 9.4 of [3]). Hence the gain here is equal to 6. This gain will ensure the same magnitude for the input signal $x[nT_1]$ and the output signal $y[nT_2]$. The output spectrum of a perfect interpolator is shown in Fig 3.2(d). All the five images in the fundamental interval have been removed.

In practice an ideal lowpass filter can not be realized. As described in session 2.3, a filter's frequency response has been divided into three parts, passband, transitionband and stopband. The bandwidth of the input signal should not exceed the passband of the filter. This will insure almost no distortion. A stopband should have a large attenuation. The attenuation will only suppress the spectral of the images to some certain extent, but will not remove them. Depends on the purpose of each application, different attenuation in the stop band will be chosen. For example, a HiFi system needs a stopband attenuation larger than $90dB$.

The straight forward way of design a SRI is not sufficient in reality for the following two reasons,

- If the interpolation factor is K, then the sampling frequency of the filter is K times larger than the input signal.
- A large attenuation in stopband and a rapid transition band need a huge amount of filter coefficients which will take longer time in the calculation of filter output.

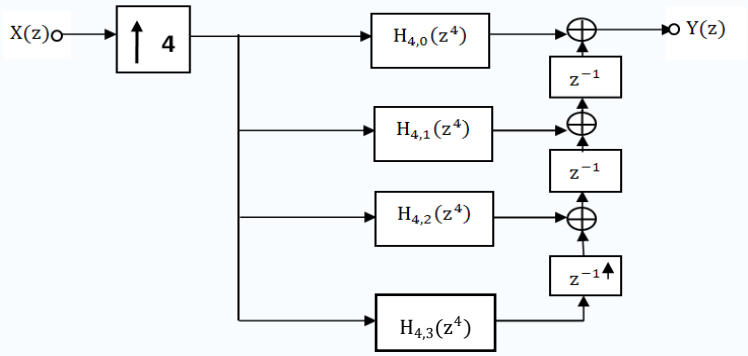


Fig 3.3 Second polyphase structure for interpolator by a factor 4

Assuming the interpolation factor is 4 and the interpolation filter is an FIR filter. An efficient structure will be introduced here by using the acknowledgments of polyphase decomposition in section (2.4). The filter has been decomposed into four ployphase components $H_{4,i}(z^4)$. Note that each component has a delay z^4 instead of delay z.

This is howed in Fig 3.3.

Noble identity for interpolation shown in Fig 2.8 will be used now to find a more efficient structure. The cascade connection of an SRI and polyphase components $H_{4,i}(z^4)$ can be interchanged. A resulting circuit is depicted in Fig 3.4 by slightly modifying the representation of delay and adder.

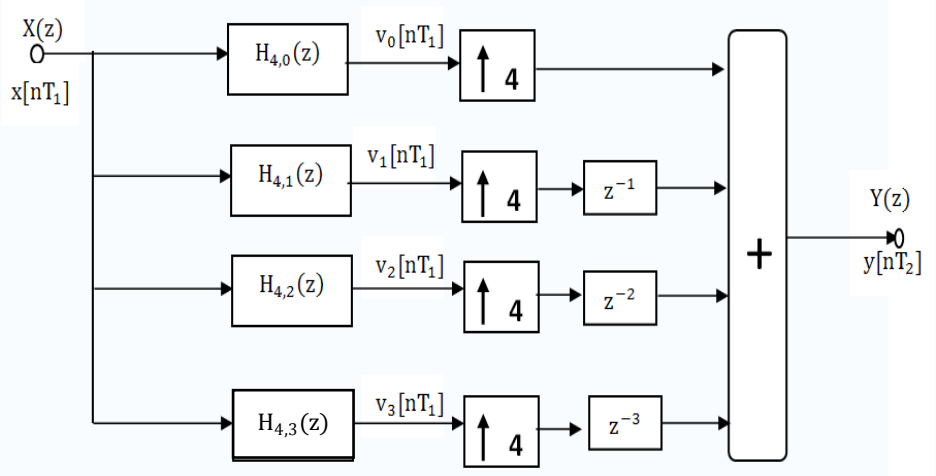


Fig 3.4 Resultig structure for interpolator by a factor 4; $T_1 = 4T_2$

Suppose the polyphase decompositions have the following outputs,

$v_0[nT_1]=\{0,4,8\}$ $v_1[nT_1]=\{1,5,9\}$
$v_2[nT_1]=\{2,6,10\}$ $v_3[nT_1]=\{3,7,11\}$

Then the output samples will be obtained by the following way. The red zeros are introduced by the delay elements and the green zeros are introduced by the SRI.

0	0	0	0	4	0	0	0	8	0	0	0
0	1	0	0	0	5	0	0	0	9	0	0
0	0	2	0	0	0	6	0	0	0	10	0
0	0	0	3	0	0	0	7	0	0	0	11
0	1	2	3	4	5	6	7	8	9	10	11

So the output $y[nT_2]$={0,1,2,3,4,5,6,7,8,9,10,11}

- The samples $y[4nT_2]$ are equal to $v_0[nT_1]$
- The samples $y[(4n+1)T_2]$ are equal to $v_1[nT_1]$
- The samples $y[(4n+2)T_2]$ are equal to $v_2[nT_1]$
- The samples $y[(4n+3)T_2]$ are equal to $v_3[nT_1]$

It can be easily seen that the four SRIs, together with the adder and delay elsments, simply interleave the outputs of four polyphase filters. Now a new symbol *interleaver*, is shown in Fig 3.5.

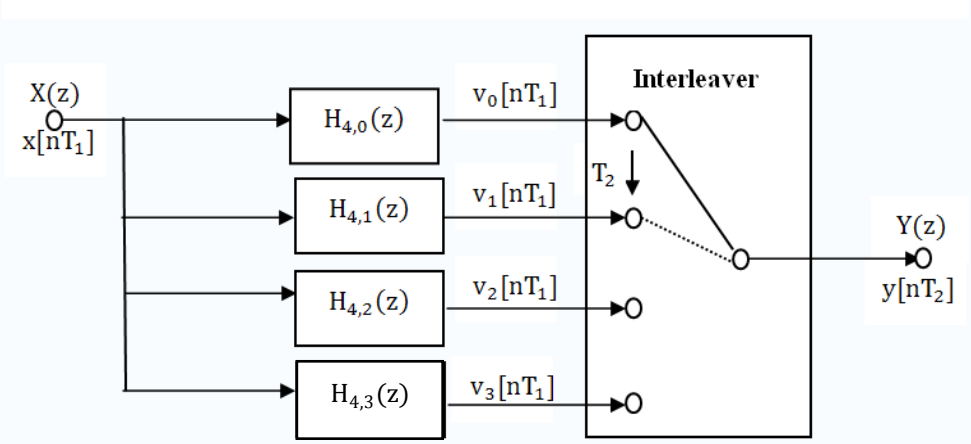


Fig 3.5 Realization of fixed interpolator using interleaver

This interleaver doesn't show its individual components. The switch repeatedly rotates itself with a time interval equal to T_2 and the arrow in the picture shows the direction. The position it will jump to is the corresponding signal path without extra delay. The interleaver for a general K is shown in Fig 3.6

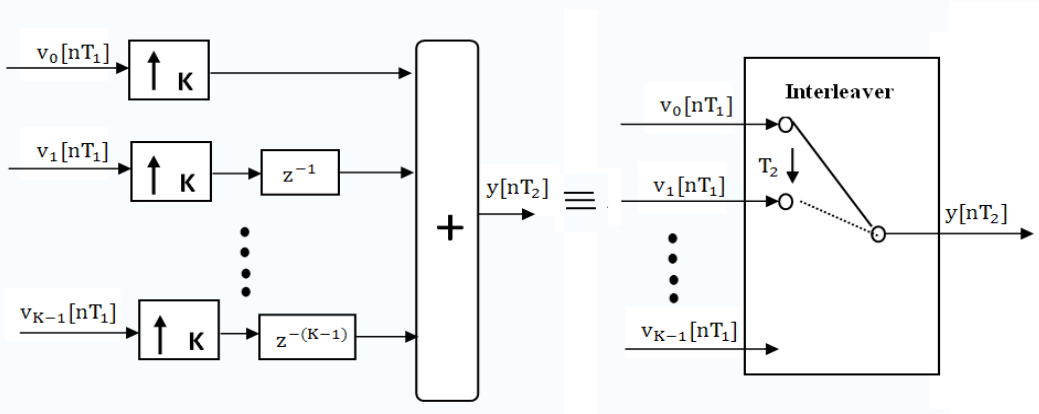


Fig 3.6 General interleaver with $T_2 = T_1/K$

For every input sample $x[nT_1]$, K output samples ,which are indicated by $v_0[nT_1]$ up to $v_{K-1}[nT_1]$ will be determined by the K polyphase filters. These samples are interleaved by the circuit shown in the left side of Fig 3.6. This circuit is symbolized by the switch in the 'interleaver' shown in the right part of this figure. This switch makes K steps in one sampling interval T_1 of the input signal.

One interesting question may be asked here. What are the advantages of such an efficient structure? To answer this question it is better to make a real example. Supposing *H(z)* is the system function of an FIR filter that has a length 8, then the direct structure can be realized as shown in Fig 3.7. The sampling frequency of the filter is $1/T_2$. At each time interval $4nT_2$, 8 multiplications and 7 additions should be used to calculate the filter output. Altogether 8 delays are needed to store the previous samples.

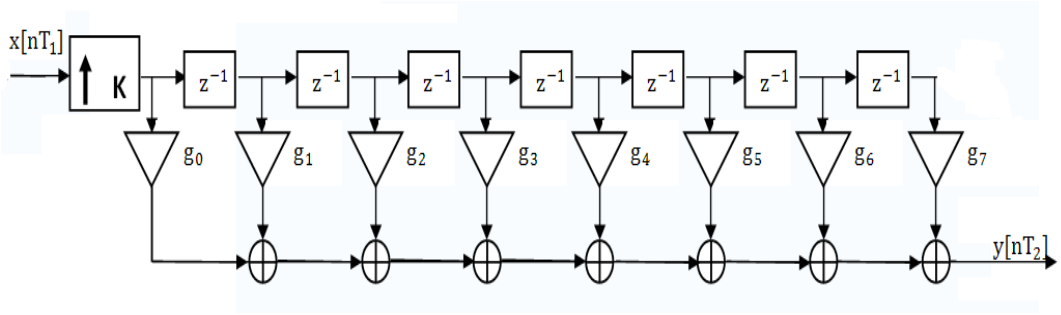
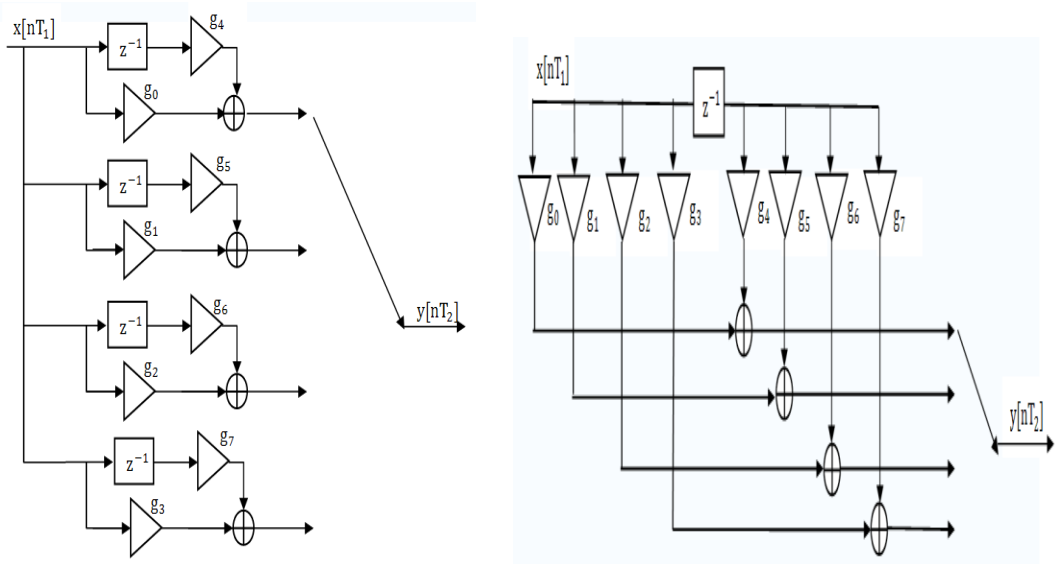


Fig 3.7 Realization of interpolator with direct structure



(a) (b)

Fig 3.8 Realization of interpolator with efficient structure

Fig 3.8(a) shows the realization of the efficient structure. 4 polyphase filters are

implemented by the transversal structure. An interleaver determined the output sample. Slightly change position of the delay in Fig 3.8(a), a final implementation is shown in Fig 3.8(b). Every T_2 seconds, an output of one polyphase filter will be calculated and will be choosen as the final output of the system. This needs 2 multipulations and 1 addition. Comparing to the direct structure, 6 multiplications and 6 additions are saved. Also this structure only needs 1 delay instead of 7. So the benefits are obviously through this example. In fact the computational consumption of final structure is a factor K less compare to the direct structure.

3.2 Decimator

As described in section 2.2, an SRD will cause aliasing effect in the fundamental interval (see Fig 2.3(d)). For an decimator, a lowpass filter also called anti-aliasing filter will usually be used to remove the aliasing effect. A straight forward way to implementing an decimator is shown in Fig 3.9. In this figure, the decimation factor is a fixed number equal to 4.

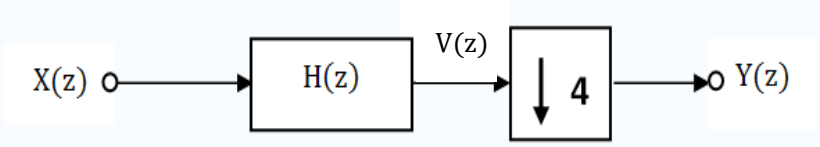


Fig 3.9 Decimator with decimation factor L=4

The forward lowpass filter limits the input signal to a small band. This insure no aliasing after decimating. The spectrum concerning in this process are drawn in Fig 3.10. Fig 3.10(a) is the spectrum of input signal. It has sampling frequency $\omega_1 = 4\omega_2$. Then the spectrum of an ideal lowpass filter is shown in Fig 3.10(b). The spectrum of the band limited signal $v[nT_1]$ is shown in Fig 3.10(c). The bandwidth of this signal is $-\frac{\omega_2}{2} \leq \omega \leq \frac{\omega_2}{2}$. All the frequency components with frequency $\omega \geq \frac{\omega_2}{2}$ or $\omega \leq -\frac{\omega_2}{2}$ will be removed. Fig 3.10(d) gives the spectrum of the output signal which has no aliasing at all. The spectrum of filtered signal $v[nT_1]$ *is* repeated at frequency $n\omega_2$. Of course, this process destroys the original spectrum. If the original signal has a lot of frequency components larger than $\frac{\omega_2}{2}$, then the decimator will give a bad result. Since decimating will also get a scaling factor equal to 1/L, the filter should have an amplitude equal to L to compensate the scaling factor. This is not drawn in the figure.

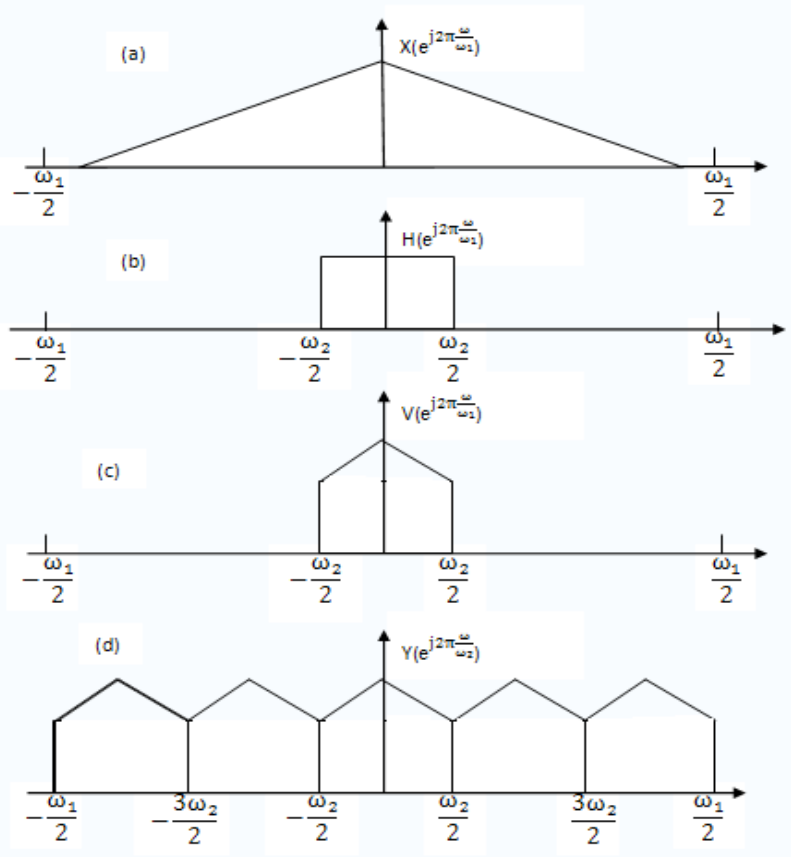


Fig 3.10 Spectrum of ideal decimator with factor L=4

The same method as described in Section 3.1will be applied here to derive an efficient structure for decimator. At first, the first polyphase structure shown in Fig 2.5 will be used. Four polyphase decompositions of *H(z)* will be created. The new structure is shown in Fig 3.11. Decimating the output signal of *H(z)* is equivalent to decimating the 4 polyphase outputs. The SRD is allowed to be shifted from the output of the system to the outputs of the four polyphase filters.

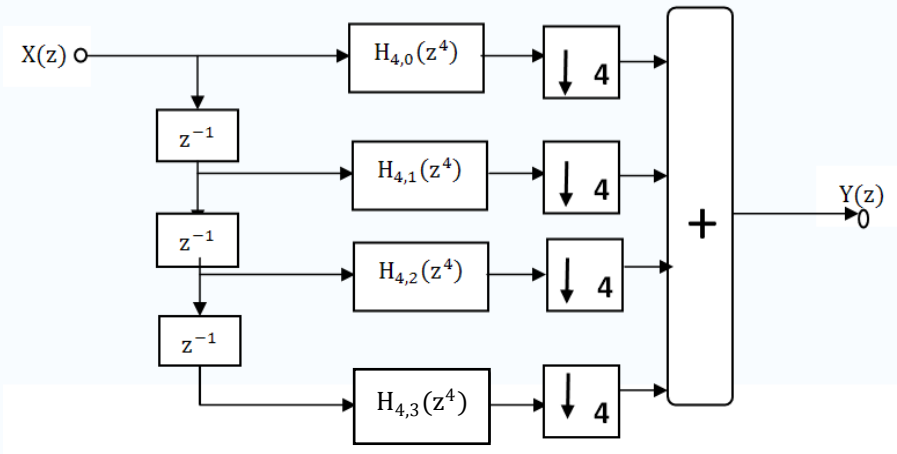


Fig 3.11 Decimator with factor 4 using first polyphase structure

Again use the noble identity for decimation, as shown in Fig 2.8, to find a more efficient structure which is drawn in Fig 3.12. The four SRDs have been shifted to the left side of the polyphase filters. In this figure, the delays have also been redrawn at the input side.

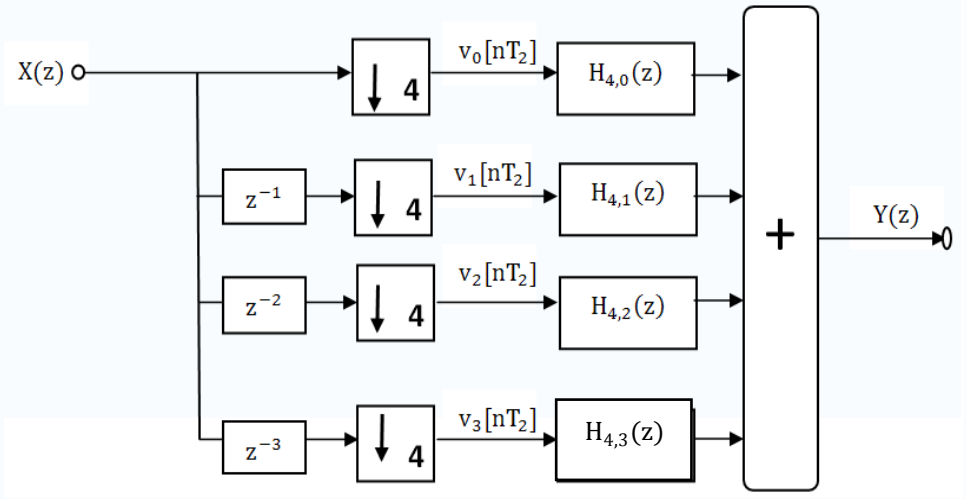


Fig 3.12 Efficient structure for decimator by a factor 4

Suppose the input sequences are *x[n]={0,1,2,3,4,5,6,7,8,9,10,11}.* Then the signals after delays and SRDs will have the following sequences

$v_0[nT_1]$*={0,4,8}* $v_1[nT_1]$*={1,5,9}*
$v_2[nT_1]$*={2,6,10}* $v_3[nT_1]$*={3,7,11}*

The result verifies the following conclusion in a straightforward way.

- The samples $x[4nT_1]$ are equal to $v_0[nT_2]$
- The samples $x[(4n+1)T_1]$ are equal to $v_1[nT_2]$
- The samples $x[(4n+2)T_1$ are equal to $v_2[nT_2]$
- The samples $x[(4n+3)T_1$ are equal to $v_3[nT_2]$

The input stage (the delay elements and the SRDs) acts as a switch. Again like the previous section, we now introduce a new special symbol for the de-interleaver, as depicted in Fig 3.13.

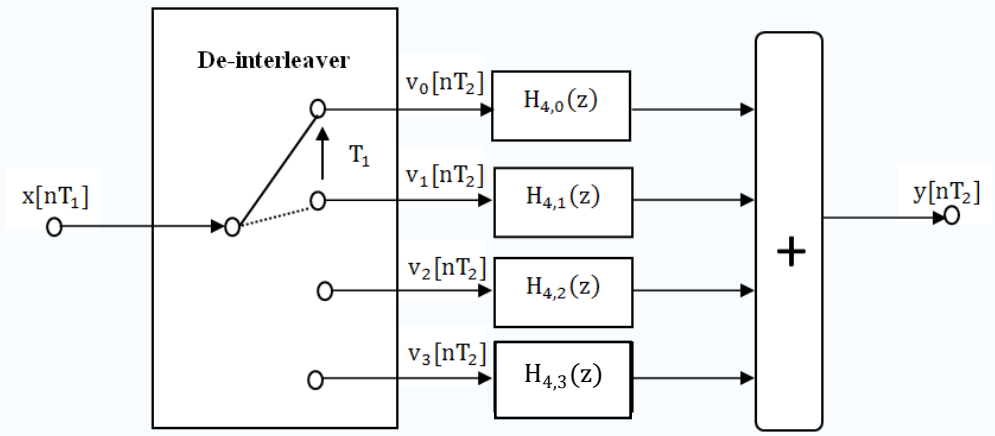


Fig 3.13 Representation of decimator with factor by de-interleaver

Each time a new input sample comes, the de-interleaver will pass it to the corresponding polyphase filter. After four input sampling intervals in Fig3.13, the switch return to the upper position and a new output sample $y[nT_2]$ is obtained by adding the outputs of the four polyphase filters together.

The de-interleaver is the transposed structure of the interleaver. Like the interleaver, the de-interlever acts as a black box without showing its individual components. The switch repeatedly rotates itself with a time interval equal to T_1 and the arrow in the picture shows the direction. The position it will jump to is the corresponding signal path without extra delay. The de-interleaver for a general L is shown in Fig 3.14.

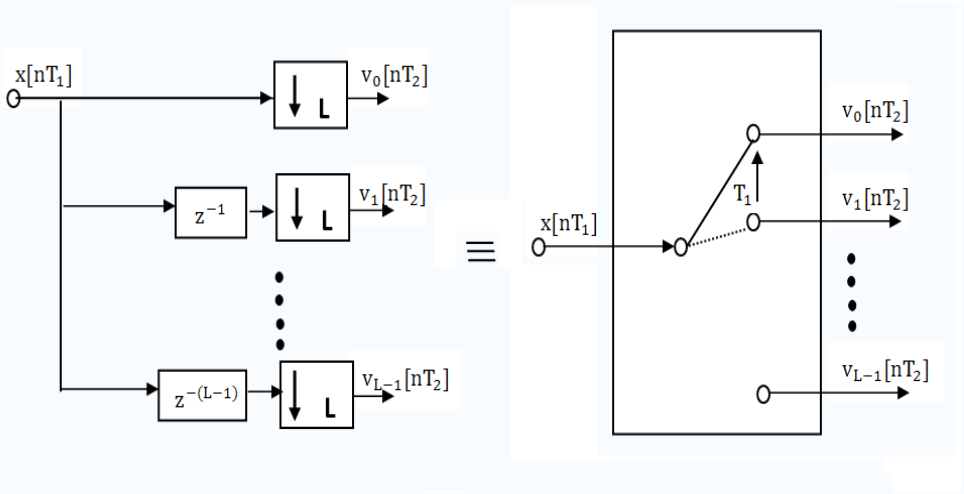


Fig 3.14 General de-interleaver with $T_2 = LT_1$

Now assuming the *H(z)* is the frequency response of an FIR filter with length N=8. The realization of the system can be shown in Fig 3.15.

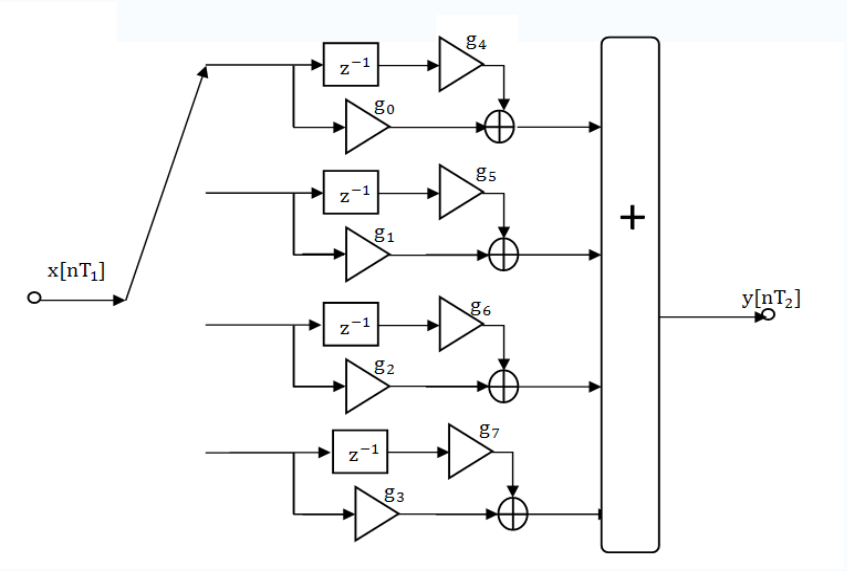


Fig 3.15 Realization of Decimator by a factor 4 using a standard transversal filter

From Fig 3.15 it is clear that the number of delay elements is 4. Compared with the direct structure, 3 delay are reduced. Each time interval $4T_1$, an output of a polyphase should be calculated. This means the sampling frequency of the polyphase filters is $1/4T_1$. The sampling frequency of each polyphase filter is reduced by 4 times. The calculation for each filter needs only two multiplications and one addition. Generally, the calculation efforts can be reduced L times less, too.

4 Rational SRC

In chapter 3, decimation and interpolation by an integer factor, has been introduced. However, in a large range of applications sampling-rate conversion by a rational factor K/L, with K and L integers, has to be performed. In this chapter, such a sampling-rate converter will be explained in detail. The basic system block is shown in Fig 4.1.

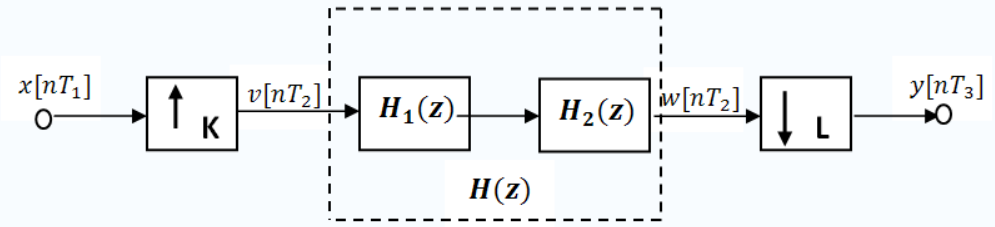


Fig 4.1 Rational sampling–rate conversion by a factor K/L

In this circuit, it is a cascade connection of an interpolator followed by a decimator. $H_1(z)$ is an anti-image filter and $H_2(z)$ is an anti-aliasing filter. In most cases, the position of SRI and SRD can not be interchanged. The reason is that this reduces the risk of introducing aliasing by the SRD. The two filters can be reduced to one with frequency response equal to $H(z)$. If the interpolation factor K and the decimation factor L are fixed integers, the passband of the filter $H(z)$ is determined by the lowest sampling frequency either $\mathrm{F_{s1}} = 1/T_1$ at the input side or $\mathrm{F_{s3}} = 1/T_3$ at the output side. The even interesting solution is that for more than one ration K/L the digital filter $H(z)$ needs not to be changed.
In the following section, a SRC which has a constant K and a variable decimation factor L is explained first. The structure used to implement it is called the *original* structure. After that another SRC which has the variable interpolation factor K and a constant L will be designed using a *transposed* structure.

4.1 SRC with variable decimation factor

At first, assuming the K and L in the system shown in Fig 4.1 are fixed integer. The signal $v[nT_2]$ is obtained by inserting *(K-1)* zeros between every pair of the input samples. So the sampling frequency of $v[nT_2]$ is K times larger than the input sampling frequency. Then this signal is passing through the digital filter $H(z)$. Finally the sampling frequency is decreased by selecting only every L^{th} samples of $w[nT_2]$.

Now take K=4 and L=3 for example. Fig 4.2 shows the spectra of the signals $x[nT_1]$, $v[nT_2]$ $w[nT_2], y[nT_3]$ and the frequency response of the filter. As said before, this is the original structure, the filter acts as an anti-image filter. Hence the cut-off frequency is equal to $\omega_1/2$. The spectrum of $w[nT_2]$ is equal to the spectrum of $v[nT_2]$, with the spectral image between $-\frac{\omega_2}{2} \leq \omega \leq \frac{\omega_1}{2}$ and $\frac{\omega_1}{2} \leq \omega \leq \frac{\omega_2}{2}$ suppressed. For a fixed integer interpolation factor *K*, the value of the decimation factor L in Fig 4.1 can be changed without changing the filter $H(z)$ as long as $L \leq K$. Hence it actually acts as an interpolator with a rational factor.

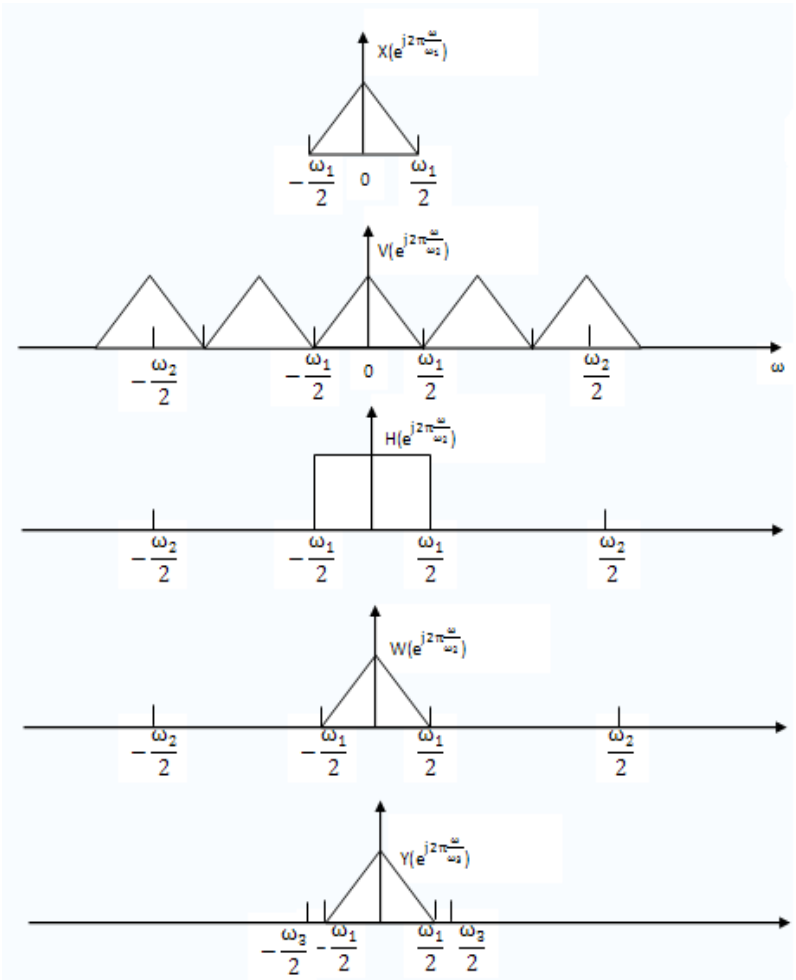


Fig 4.2 The spectra for ideal rational interpolator that increase the sampling frequency by a factor 4/3

Fig 4.3 shows the output spectrum of the system for an input sampling frequency of 9kHz, *K*=4 and for the integer *L* ranging from *L*=1 up to 4. The sampling frequency of the *H(z)* is 36kHz. It is clear that there are no unwanted aliasing for these 4 cases. The scaling factor of the lowpass filter is equal to K. This is not drawn in the figure. For some certain kind of applications, however, the factor *L* has to be larger than *K*. In this case, the system acts as a decimator with a rational factor instead. If the cut-off frequency of thefilter has not been adapted, the system will give an output

with unwanted aliasing. This is shown in Fig 4.4 for K=4 and L=5 with an input frequency equal to 9kHz.

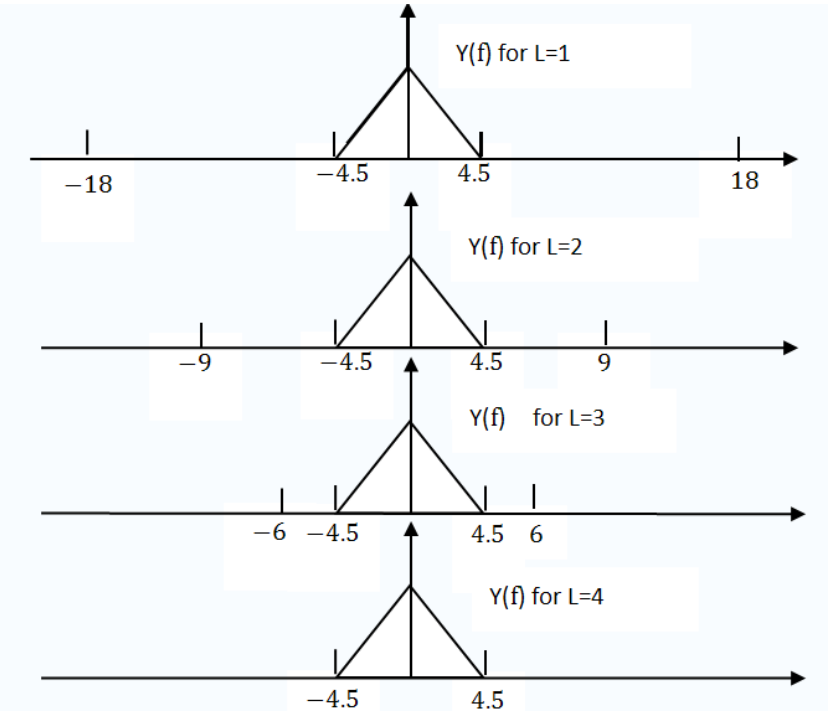


Fig 4.3 The output spectra Y(f) for interpolation by 4/L, with L=1 up to 4

The spectrum of the signal $w[nT_2]$ is shown in Fig 4.4(a). The fundamental interval of the output signal $y[nT_3]$ is represented by FI in that figure. Fig 4.4(b) gives the output spectrum. It can be seen from this figure that $y[nT_3]$ is corrupted by aliasing (hatched region) form frequency 2.7kHz to 3.6kHz.

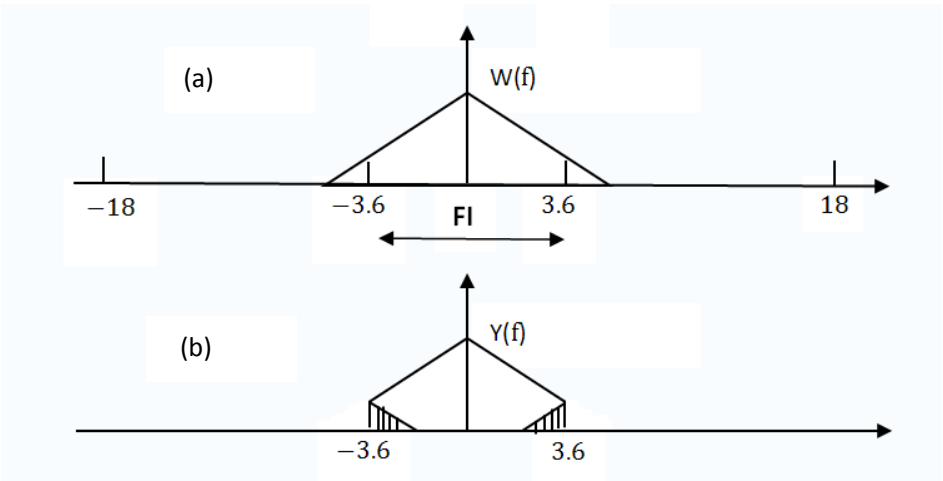


Fig 4.4 The output spectrum of fixed SRC with interpolation factor K=4 and decimation factor L=5 which has an input sampling frequency $Fs_{in} = 9kHz$

This problem can be solved by adapting the bandwidth of filter $H(z)$. The cut-off

frequency of the ideal lowpass filter should be equal to $\frac{9\text{kHz} \times 4 \div 5}{2} = 3.6\text{kHz}$.

Now the scaling problem will be discussed here. If the decimation factor L has to be larger than K and if L has to be a variable factor, the gain of the lowpass filter still has to be fixed and equal to K.

From these examples it is clear that the system of Fig 4.1 is a sampling rate converter with the following features:

- A fixed integer interpolator K and a fixed filter $H(z)$ which has a cut-off frequency equal to half of the input sampling frequency
- A variable integer decimation factor L where $L \leq K$

This means the system is an interpolator with a variable interpolation factor. This factor is equal to K/L where $L \leq K$.

4.2 SRC with variable interpolation factor

In order to derive a rational SRC with variable interpolation factor, the decimation factor L in Fig 4.1 will be assumed to be a fixed number. This time the variable interpolation factor K has a range $K \leq L$. In this section it will also proved that the resulting system does not introduce any unwanted phenomena as long as $K \leq L$, despite the fixed filter $H(z)$. Assuming the interpolation factor K is equal to 3 and the decimation factor L is equal to 4. Fig 4.5 shows the spectra of the signals $x[nT_1]$, $v[nT_2]$ $w[nT_2]$, $y[nT_3]$ and the frequency response of the filter (only the spectrums in the fundamental intervals are drawn). In this figure $\omega_2 = 3\omega_1$, $\omega_2 = 4\omega_3$. Now the filter acts as a anti-aliasing filter. Hence the cut-off frequency is now equal to $\omega_3/2$. The spectrum of $w[nT_2]$ is equal to the spectrum of $v[nT_2]$, with the spectral image between $-\frac{\omega_2}{2} \leq \omega \leq \frac{\omega_3}{2}$ and $\frac{\omega_3}{2} \leq \omega \leq \frac{\omega_2}{2}$ suppressed.

Be careful, the interpolator factor is equal to 3 to ensure that the magnitude of $w[nT_2]$ is not changed. In the general situation, the filter gain is equal to the interpolation factor K. The output signal should be scaled by an additional factor K/L. These scaling factors are not displayed in the figure. In the same way as shown in Fig 4.3, it can be proved that there are no unwanted distortions if $K \leq L$.

For K=5 and L=4, the spectral of $v[nT_2], y[nT_3]$ are shown in Fig 4.6. In Fig 4.6(a) the dotted line is the frequency response of the filter. The frequency components of $v[nT_2]$ which are inside the window of the filter are the spectrum of $w[nT_2]$. Fig 4.6(b) is the output of the total system. This figure tells that if K is larger than L, the spectral images created in the interpolation process are not completely suppressed

by the filter. Unwanted spectral images will be introduced at the system output. These images are the hatched area in Fig 4.5(b).

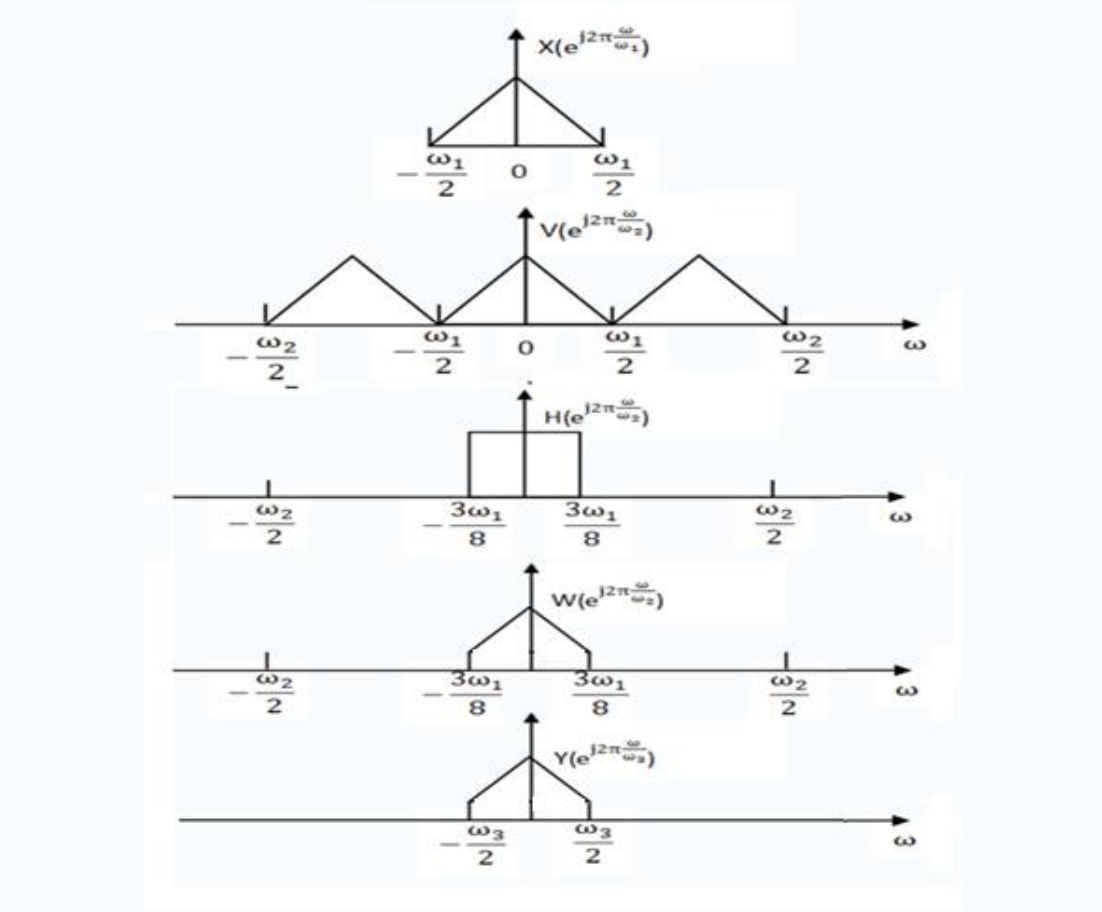


Fig 4.5 The spectra for ideal rational decimator that decreases the sampling frequency by a factor 4/3

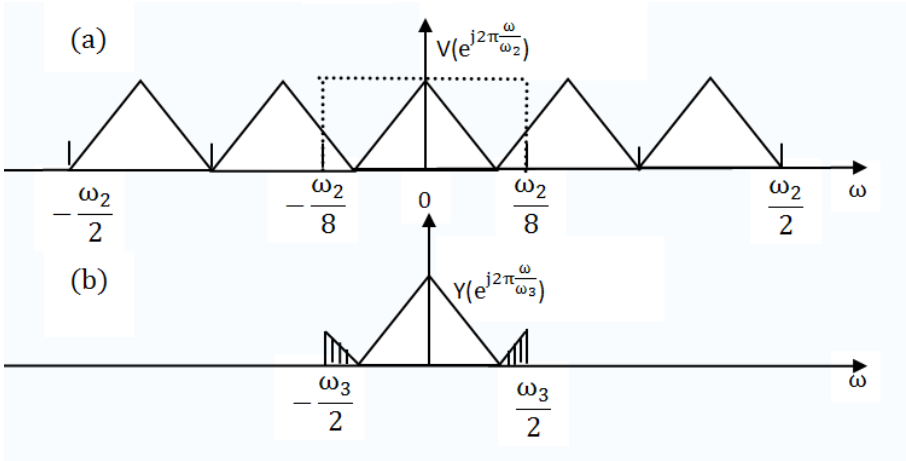


Fig 4.6 The spectra for 'transposed' interpolator with factor 4/5

From these examples it is clear that the system of Fig 4.1 is a sampling rate converter with the following features:

- A fixed integer decimator L and a fixed filter $H(z)$ which has a cut-off frequency equal to have of the output sampling frequency
- A variable integer decimation factor K where $K \leq L$

This means the system is a decimator with a variable decimation factor. This factor is equal to L/K, where $K \leq L$.

4.3 Efficient structure for SRC with variable decimation factor

A practical implementation of the system specified in section 4.1 will be described here. The same example as used in section 4.1 will also be used here. The rational interpolator system with fixed $K=4$ and variable L within range $L \leq K$. In Fig 4.7, the polyphase decomposition for the interpolating part of the system is drawn. The system with interleaver , which is shown in Fig 3.5, is used here. In this circuit, the interpolating part has already been realized in an efficient way by exploiting the zero-valued input samples. The calculation time has been reduced by a factor of 4.

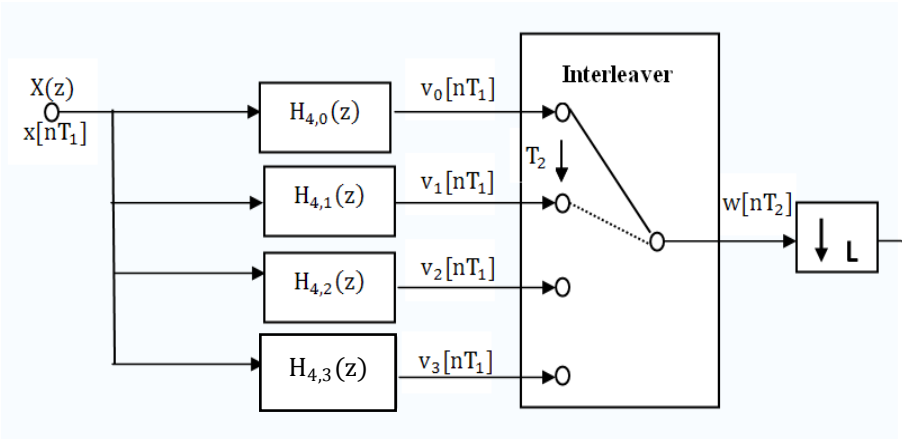


Fig 4.7 First step in the efficient realization of a rational interpolator by a factor 4/L; $T_2 = T_3/L = T_1/4$

It is clear that the system calculates all samples of $w[nT_2]$. But only 1/L sample sof them are used as an output sample. The selection of the final output samples is performed by the SRD. In order to get a more efficient structure, the interleaver should be modified properly. This is done by setting the rotating interval to be T_3 and the distance it rotates to be L instead of 1. Therefore the final implementation of the system is shown in Fig 4.8 for L=3. An alternative circuit realization, using the polyphase decomposition and the identities, can be found in Appendix C of [4]. Computational efforts for both systems are the same.

In this example, the switch in the interleaver rotates anti-clockwise. Each time interval T_3 it rotates for 3 positions. In a general system with a fixed interpolation K and a decimation factor $L \leq K$, there are K different polyphase filters. The switch at output rotates anti-clockwise and step over L positions between every two output

samples. The polyphase filter which determines the output signal $y[nT_3]$ is represented by $H_{K,i}(z)$, where i is calculated by :

$$i=(Ln) \bmod K \tag{4.1}$$

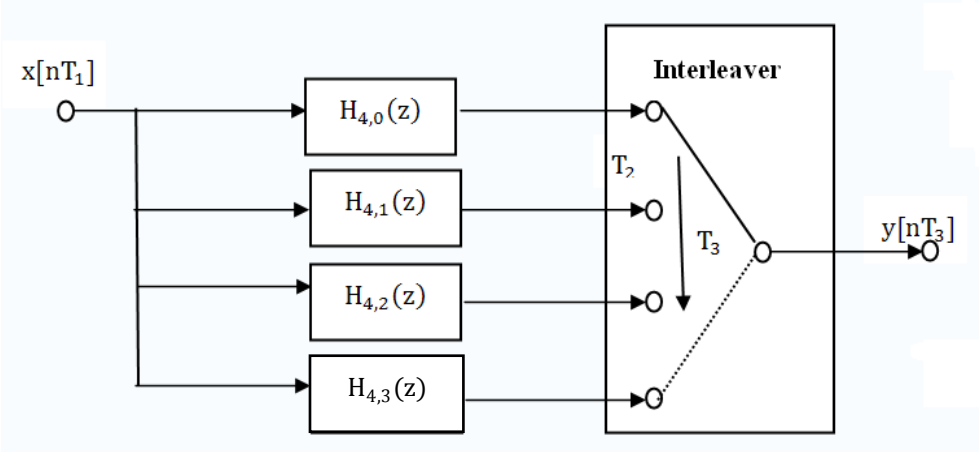


Figure 4.8 Final efficient structure for interpolation by 4/3

4.4 Efficient structure for SRC with variable interpolation factor

In this section, the system described in section 4.2 will be realized in an efficient structure. The system has a fixed decimation factor L and a variable interpolation factor *K*. The system works as long as $K \leq L$. As said before, it acts as a rational decimator. The filter *H(z)* works as a anti-aliasing filter. The first step is to realize the efficient structure for a filter combined with a decimator. This can be done through the circuit described in Fig 3.13. The resulting circuit is shown in Fig 4.9 for *L=4*.

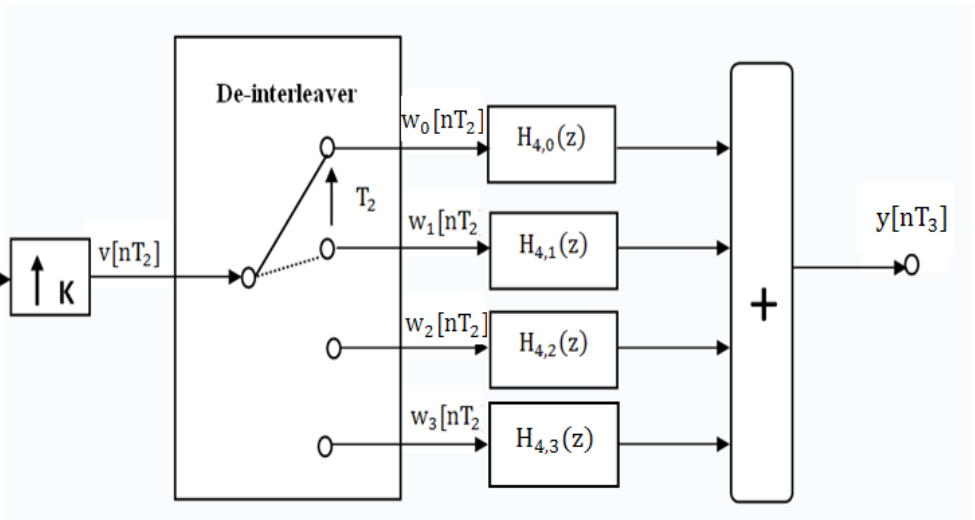


Fig 4.9 First step of realizing a decimator by 4/K

The switch rotates anti-clockwise and distributes the samples of $v[nT_2]$ over the polyphase filters. Howerver, every T_1 seconds the SRI introduces *(K-1)* zeros in $v[nT_2]$. These do not have to be distributed. For K=3, this leads to the structure of Fig 4.10.

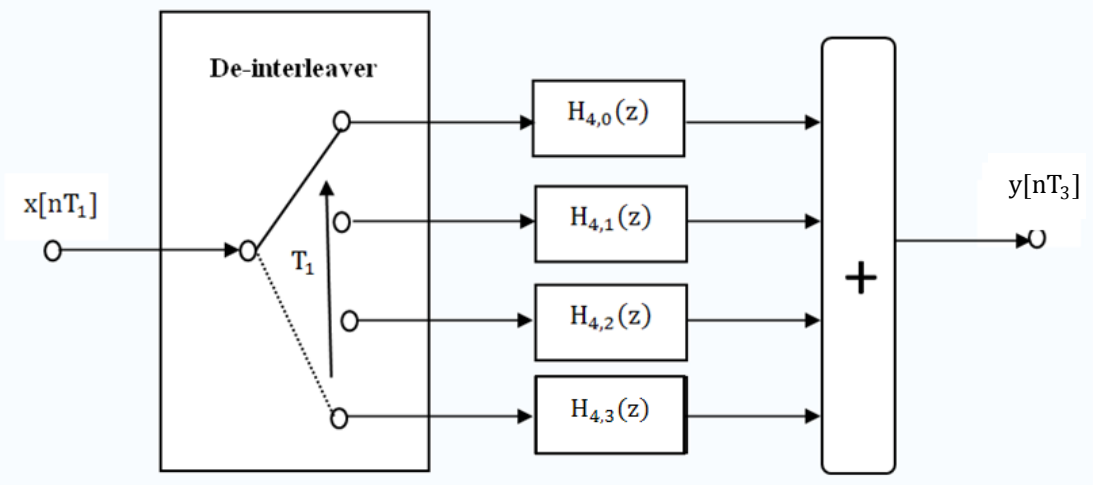


Fig 4.10 Final Efficient structure for decimator by a factor 4/3

For variable *K* and $K \leq L$, the SRC can be generalized easily. There are altogether *L* polyphase filters and switch steps over *K* positions in an anti-clockwise direction.

For small K and L, the two structures described in this chapter lead to an efficient solution. But if K and L are larger, then these two circuits become very complicated. It has a large number of coefficients and a lot of delays which will take more memory space. The implementation shown in the next chapter will solve the problem above.

5 Flexible SRC

Efficient flexible sampling-rate converters will be described in this chapter. As already mentioned in the requirements of this project, the actual ratio between input and output frequency doesn't have to be known in advance. The two sampling frequencies might even vary as a function of time. In order to design a suitable filter to remove the spectral disturbances caused by SRI and SRD, the relative bandwidth, the maximum passband ripple and the required attenuation should be known. The flexible interpolator will be described first. After that a flexible decimator will be implemented using the transposed theory.

In order to avoid misinterpreting the symbols in the following sections, it is necessary to redefine them first. Digital filters used in the flexible converters are generally denoted by $G(z)$ and the polyphase filters are denoted by $G_i(z)$. The system function of a *linear interpolator* (LI) which will be used in this chapter is represented by $H(z)$. The sampling frequencies from the input side to the output side are represented by F_{s1}, F_{s2}, F_{s3}... and the corresponding sampling intervals are given by T_1, T_2, T_3 ....

5.1 Flexible interpolator using linear interpolator

An impractical way to design a system used for conversion between two arbitrary sampling frequencies is to use an analog way. This is shown in Fig 5.1. The D/A converter converts the digital signal into an analog signal. An analog *LowPass Filter* (LPF) is used to remove the unwanted spectral images. Finally, re-sample the analog signal with a new sampling frequency equal to F_{s2} by an A/D converter. The cut-off frequency is equal to $F_{s1}/2$ for interpolation and $F_{s2}/2$ for decimation.

Fig 5.1 SRC using analog signal processing

The circuit shown in Fig 5.1 is expensive analog block. The bandwidth of the LPF is fixed. Once it is build, it is impossible to change. Furthermore, the frequency of F_{s1} and F_{s2} is unknown. This makes difficult to build such a LPF. Usually a flexible all digital solution shown in Fig 5.2 will be used here. Unfortunately this structure has a main disadvantage. It can deal with the ratio only for small integer division.

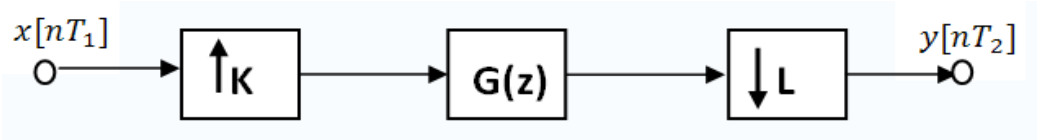


Fig 5.2 All-digital solution structure for a flexible SRC (not practical)

Supposing the K is equal to 128 and L is equal to 2. The timing diagram for the whole process is shown in Fig 5.3. T_1 is the input sampling interval, T_2 is 128 times smaller than T_1. T_3 is the time interval of output. Fig 5.3 (a) and (b) show the signals of $x[nT_1]$, $w[nT_2]$ and $y[nT_3]$. It is clear that T_3 equals to $2T_2$ and the output sample at that time is equal to sample C indicated in the figure. Fig 5.3 (c) shows the situation when L is equal to 2.13. The time interval ΔT indicated in Fig 5.3 (c) is equal to $0.13T_2$. Because there is no actual sample at that time, the nearest sample will be used to approximate the output sample. In this case, sample C will be used. Generally if $\Delta T \leq T_2/2$, the output is determined by the sample at time interval ΔT before. If $\Delta T \geq T_2/2$, the sample at time interval $T_2 - \Delta T$ later will be used as the output sample. This introduces am amplitude error when $\Delta T \neq 0$. Because ΔT is not fixed, the amplitude deviation is a function of time. This makes it almost impossible to adjust the error later.

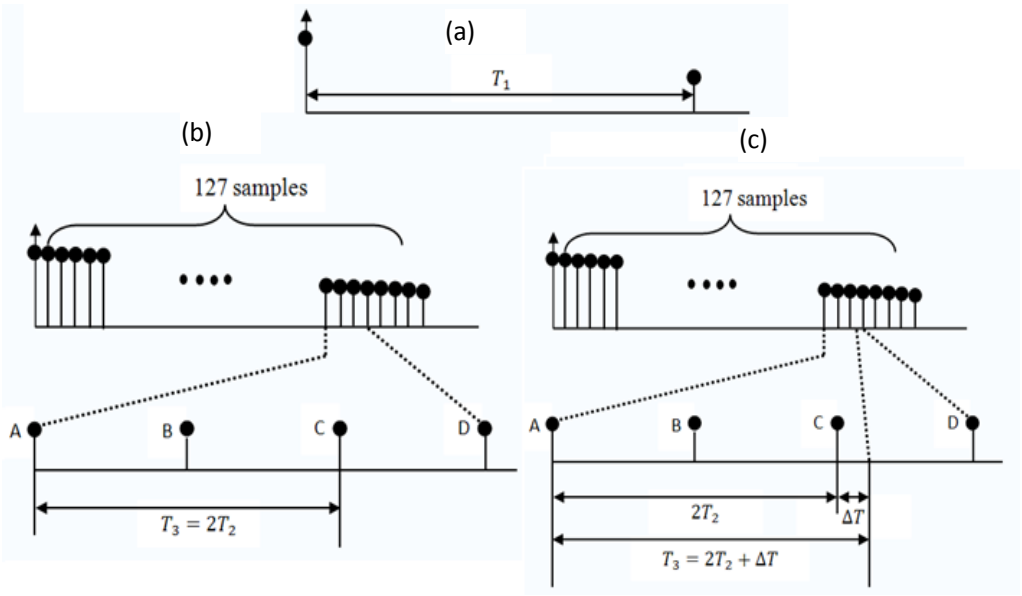


Fig 5.3 Timing diagram for flexible interpolation

One possible way to solve this problem is to make the time interval of T_2 extremely samll. This can be done through increasing the interpolation factor K. for example, if the value of K is increased to 1024, then the ΔT will be 8 times smaller. But this

method causes the extension of filter coefficients and it is almost impossible to design such a large filter.

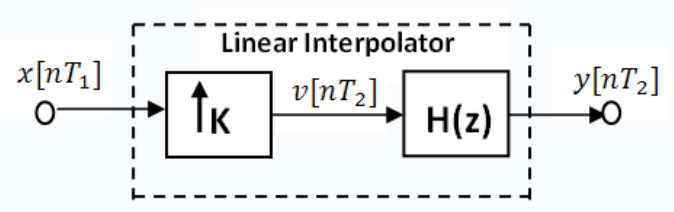


Fig 5.4 Structure of a linear interpolator

Before introducing a much more practical approach, a new building block called *linear interpolator* will be explained first. A linear interpolator with interpolation factor K is shown in Fig 5.4. It is a cascade connection of a SRI followed by a filter $H(z)$. The function of a linear interpolator is to linearly interpolate K-1 values between every pair of the input samples. The impulse response of the filter is a triangular impulse response with a length equal to $2K$-1. Fig 5.5 shows how a linear interpolator with an interpolator factor K=4 works in time domain. $x[nT_1]$ is shown as the solid circles in Fig 5.5 (a). The open circles in Fig 5.5(a) are the zeros introduced by the SDI. For simplicity, the filter is assumed to be a non-causal filter. For causality, an additional delay of at least 3 samples is needed. The corresponding output can be calculated by the following equation.

$$y[n] = \int v[n]h[t-n]dt$$

These output samples are shown in Fig 5.5(c). In this figure, the six open circles are the interpolated samples.

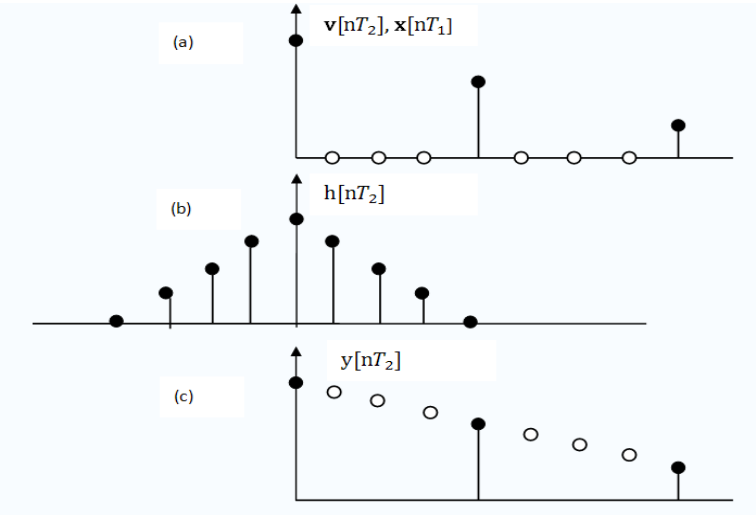


Fig 5.5 Linear interpolator in time domain

For a general linear interpolation factor K, it can easily be proved that the amplitude response for filter H(z) is equal to equation (5.1).

$$A\left(e^{\frac{j2\pi w}{w_2}}\right) = K\left(\frac{\sin(K\frac{\pi\omega}{\omega_2})}{K\sin(\frac{\pi\omega}{\omega_2})}\right)^2 \qquad (5.1)$$

From this equation, it is clear that the DC gain of linear interpolator is equal to *K*. If $A\left(e^{\frac{j2\pi w}{w_2}}\right)$ wants to have zero values, $\frac{K\omega}{\omega_2} = \frac{\omega}{\omega_1}$ $(\omega_2 = K\omega_1)$ should be an integer. That means the frequency components equal to a multiple of ω_1 will be strongly attenuated. As shown in section 2.2 tell that the spectrum of $v[nT_2]$ is just repeating the spectra components of $x[nT_1]$ in the fundamental interval with a period equal to ω_1. The position of image created by SRI coincides with the zeros in the amplitude response $A\left(e^{\frac{j2\pi w}{w_2}}\right)$ of the linear interpolator. Hence the image will be strongly attenuated.

Fig 5.6 shows how a linear interpolator works. The repeating triangles with a period of ω_2 are the spectrum of $v[nT_2]$. Because K is equal to 4, there are only three images in the fundamental interval. The bandwidth of the input signal is equal to $2\omega_a$. The behavior of the linear interpolator is shown by the dotted curve in this figure. The three images created by the SRI are strongly attenuated. The largest passband attenuation can be found at frequency ω_a and the smallest stopband attenuation can be found at frequency $\omega_b = \omega_1 - \omega_a$.

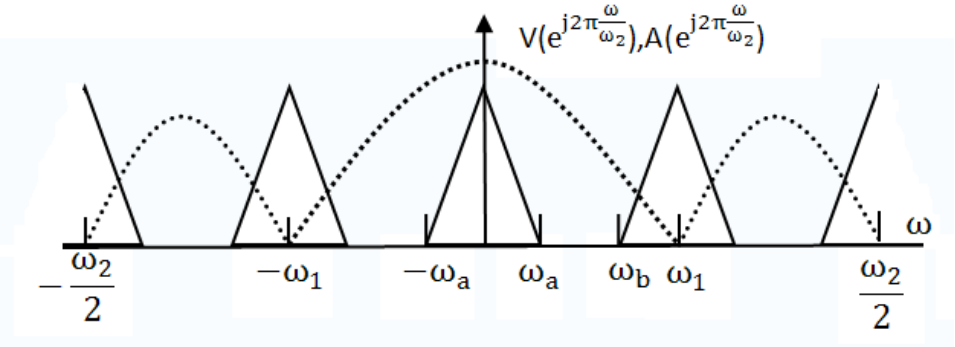


Fig 5.6 Spectra diagram of linear interpolator by a factor 4

Now a new structure for a rational interpolator which uses a linear interpolator is shown in Fig 5.7. It divides the original interpolator into two different parts. The first part is a normal interpolator with fixed factor K_0. The transition band of the anti-image filter is narrow. This leads to a high order digital filter G(z). The second part is a linear interpolator with a factor K_1. The transition band of the filter H(z) is wide. This leads to a low order filter. Actually the linear interpolator combined with the SRD acts as an "Analog Resampler". It re-samples the signal from the output of the filter G(z) at an arbitrary sampling frequency. The linear interpolator is the easiest type of analog resampler. Another one called *Lagrange interpolator* will be discussed

later in chapter 8. The value of L here should be smaller than K_0K_1.

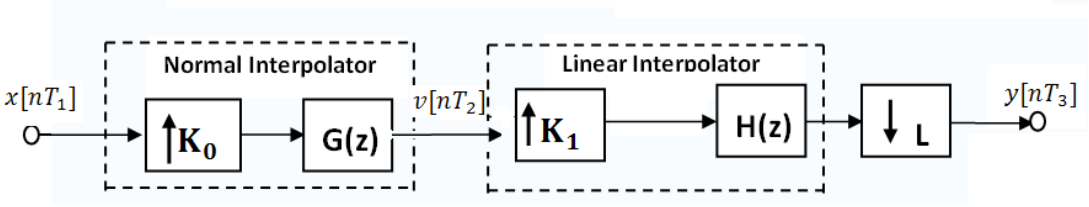


Fig 5.7 Practical two stage flexible interpolator

After the linear interpolator there is an SRD with a factor of L. That means among L output samples of the linear interpolator, only one will be used. This is not an efficient structure. The solution to this problem will be shown in Fig 5.8.

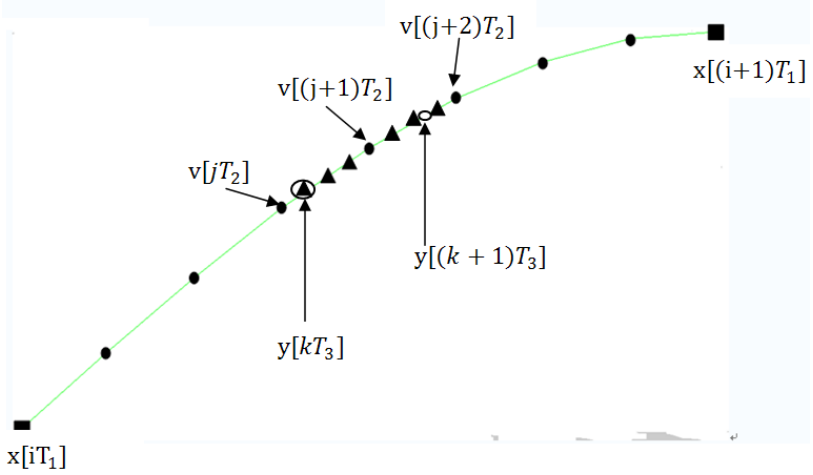


Fig 5.8 Timing diagram for interpolator by 8 followed by a linear interpolator with factor 4

The values of K_0 and K_1 are supposed to be 8 and 4 for simplicity. Two successive samples x[iT_1] and x[(i+1)T_1] of the input signal x[nT_1] are shown by the rectangular first, together with the new sampling grid after interpolation by a factor 8. Three adjacent samples, v[jT_2] v[(j+1) T_2] and v[(j+2) T_2] of v[nT_2] are shown by solid circles in Fig 5.8. y[kT_3] and y[(k+1)T_3] are the two output samples. They are shown by open circles in the figure. The triangulars in the figure are the samples interpolated by the linear interpolator. There are two ways to get the value of y[kT_3].

1) Using the output samples of the linear interpolator to calculate the output samples. In this figure, y[kT_3] is equal to first output value of the linear interpolator.
2) Using the output samples of the normal interpolator. Supposing the time distance from v[jT_2] to y[kT_3] is δT_2, where $0 \leq \delta < 1$. δ is called the linear-interpolation coefficient. Then the output can be calculated by the following equations,

$$y[kT_3]=(1-\delta)\, v[jT_2]+ \delta v[(j+1)\, T_2]. \qquad (5.2)$$

From Fig 5.8, the value of δ for y[kT$_3$] is equal to 0.25. It can be proved that the two methods give the same result.

Compare with the two methods, the second has two main advantages.

- It avoids the computation of output samples from the linear interpolator. It uses directly the output samples of filter G(z) to derive an output.
- When the final output samples y[nT$_3$] are at the middle of the two output samples of linear interpolator, for example y[(k+1)T$_3$] shown in the figure, the first method can't give a correct result. But for the second method, just the value of δ is changed.

The two equivalent circuits of calculation the output of the whole system are shown in Fig 5.9. The left-hand circuit performs two multiplications and one addition per output sample while the right-hand circuit requires only one multiplication, but two additions.

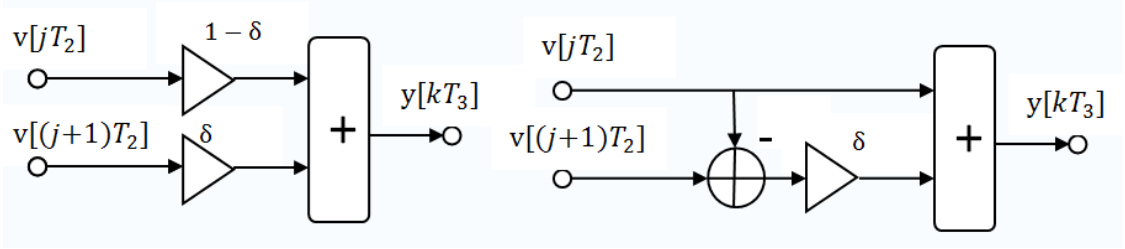


Fig 5.9 Two equivalent circuits for efficient realization of a linear interpolator followed by an SRD

A new symbol for the two circuits in Fig 5.9 will be introduced here to make further analysis simple. This is shown in Fig 5.10, where A = v[jT$_2$] and B= v[(j+1) T$_2$] and y[kT$_3$] =C.

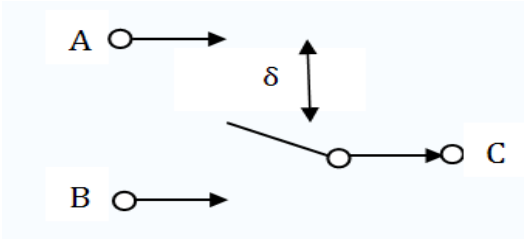


Fig 5.10 The symbol for calculation of C=(1-δ)A+δB The distance between upper position and the switch is δ, and the distance between switch and lower position is 1-δ.

Now a complete practical realization of the total flexible interpolation system can be derived by using the following steps:

1. Using figure 5.7 as the starting point. The input signal $x[nT_1]$ first passes through a normal interpolator with factor K_0=8. The output signal is $v[nT_2]$
2. One output sample has to calculated from a corresponding pair of adjacent samples of $v[jT_2]$ and $v[(j+1)T_2]$ (see equation (5.2))
3. Supposing the whole system gives an output sampling frequency $F_{s3} = 3.3F_{s1}$
4. The implementation of G(z) uses the circuits shown in Fig 3.5. The 8 polyphase filters create 8 output samples $v_j[nT_1]$ with j in a range from 0 to 7. The interleaver rotates anti-clockwise with a particular distance to find the correct pair of $v[nT_2]$.

Fig 5.11 shows the efficient implementation. The interleaver of Fig 5.11 has been extended to a linear-interpolator interleaver. Compare to the interleaver in Fig 4.8, the switch can take a position at a distance δ in between two particular polyphase filters. In this efficient structure, the output samples of polyphase filters which do not contribute to the final output are neglected. Therefore, calculation efforts for those polyphase phase filters are saved. Also the value of K_1 and L are not used in this structure. Only the ratio of them is used here.

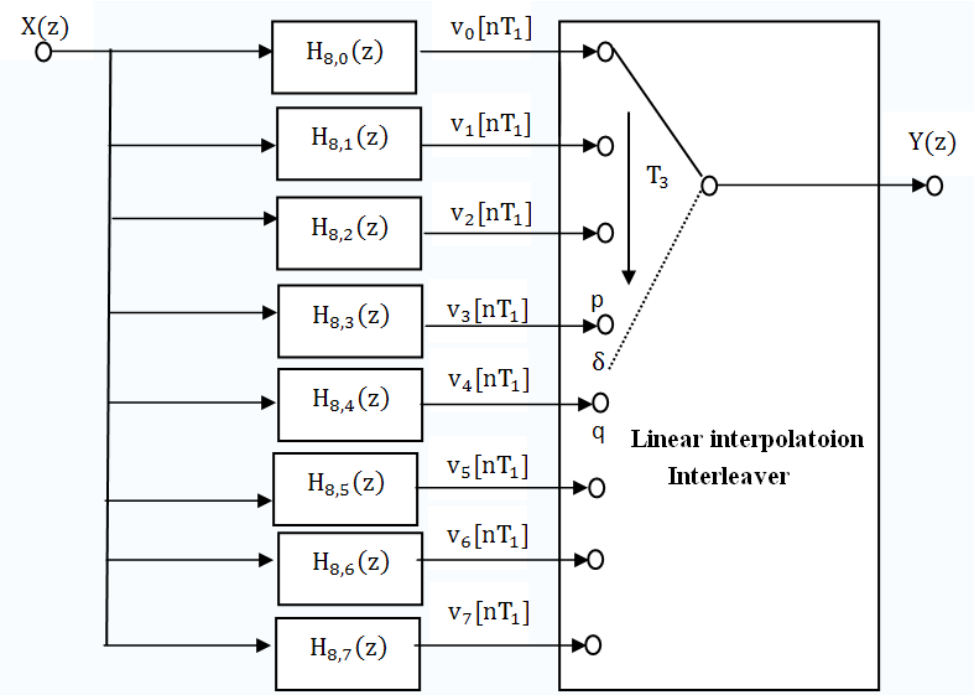


Fig 5.11 Implementation of the flexible interpolating system

The remaining problems are how to determine the positions p and q of the switch, the value of δ and the actual times at which the polyphase outputs of Fig 5.11 have to be calculated. The solution will be shown in Fig 5.12.

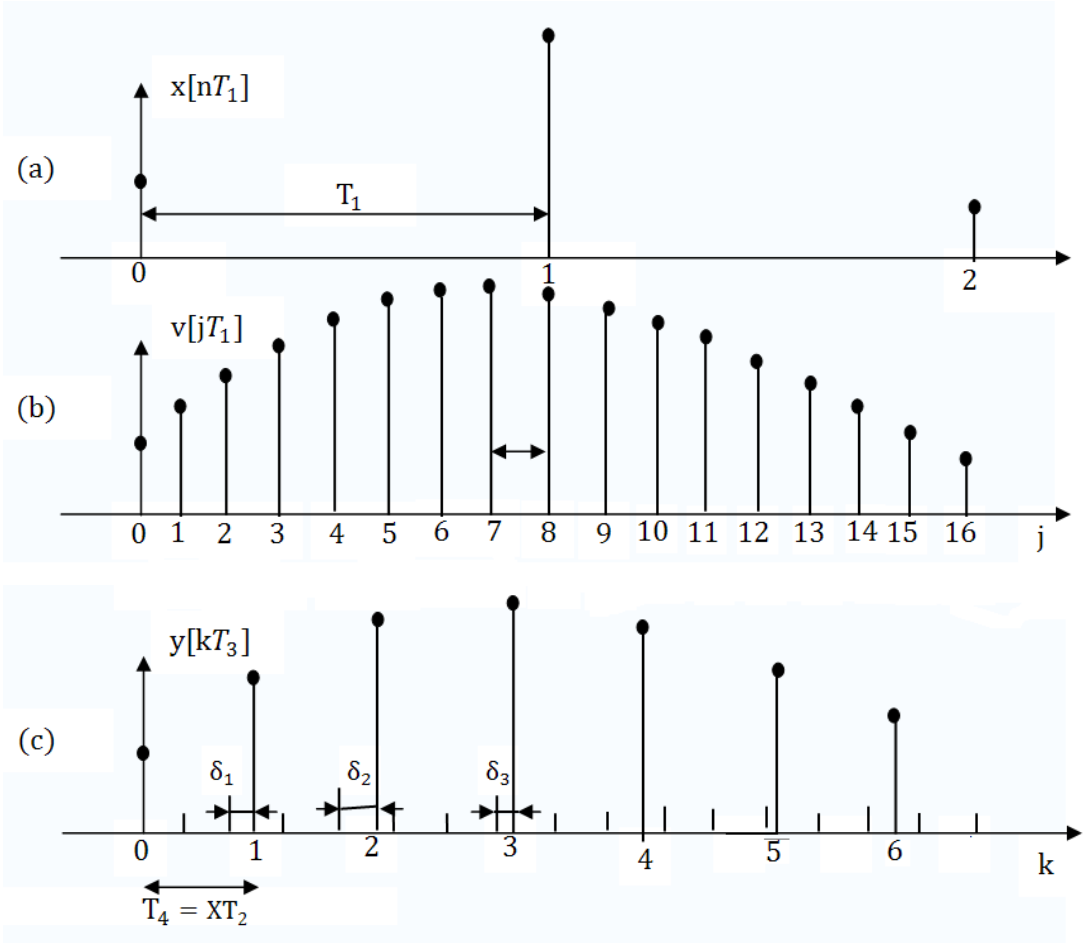


Fig 5.12 Time diagram for flexible interpolator

In Fig 5.12(a), the input signal are shown for n=0 to 2. Fig. 5.12(b) shows all the interpolated samples calculated by the 8 polyphase filters. The output samples are shown in Fig 5.12(c) for k between 0 and 6. The output sampling interval T_3 is equal to XT_2

$$T_3 = XT_2 = \frac{XT_1}{K_0}.$$

Gernally in this equation, X is a non-integer number. In this example, X≈ 2.42. The value of X can be calculated from the ratio of the two sampling frequency. From previous equation:

$$X = \frac{K_0 T_3}{T_1} = \frac{K_0 F_{s1}}{F_{s3}} \qquad (5.3)$$

The factor X is equal to the non-integer number of the step size that the switch of Fig 5.11 has to rotate between two output samples. The k^{th} output $y[kT_3]$ can be calculated by using the outputs of the polypohase filter p and q=(p+1) at time instant

nT_1. Note there is a special case, when the value of p is equal to K_0-1. In this situation, the output should be calculated by the output of filter p=(K_0-1) at $n = n_1$ and the output of the filter q=0 at time $n = n_1 + 1$. Also the value of δ is needed to calculate the output. This value is equal to the non-integer part of kX. So the variables of p, q, n, δ are function of time index k. The linear interpolation can be represented by the following equation,

$$y[kT_3] = (1 - \delta)v_p[nT_1] + \delta v_q[nT_1] \quad (5.3)$$

or in the special case

$$y[kT_3] = (1 - \delta)v_{K_0-1}[nT_1] + \delta v_0[nT_1] \quad (5.4)$$

In Fig 5.12(c), we have

- for k=1: p=2; q=3; and n=0
- for k=2: p=4; q=5; and n=0
- for k=3: p=7 at n=0; q=0 at n=1 (special case).

More calculation are shown in table 5.1.

Output index k	kX	p	q	δ	n	Special case
0	0	0	1	0.000	0	no
1	2.424	2	3	0.424	0	no
2	4.848	4	5	0.848	0	no
3	7.273	7	0	0.273	1	yes
4	9.697	1	2	0.697	1	no
5	12.121	4	5	0.121	1	no
6	14.545	6	7	0.545	1	no
7	16.970	0	1	0.970	2	no

Table5.1: The control of the switch for the flexible interpolator with an interpolation factor equal to 3.3; (8 polyphase filters are used here for the first interpolation stage)

The control variables $p, q, n\ and\ \delta$ can formally be calculated from kX, with k increase from 0 to infinite,

$X = K_0 F_{s1}/F_{s3}$

p = (integer part of kX) mod K_0

q = (p + 1) mod K_0

δ = non-integer part of kX

n = (kX) div K_0 n is the index of input samples
(i.e. largest integer smaller or equal to kX/K_0)

5.2 Matlab simulation for Flexible interpolator

In this section the implementation of a flexible interpolator specified in the previous section will be done in Matlab. The implementation has the following steps:

1) Set the input and output sampling frequency.
2) Set the value of K_0, the default value is 8.
3) Use Matlab command "remezord " to get coefficients of the filter G(z) according the input and output sampling frequency. The cut-off frequency is equal to half of the input sampling frequency. The sampling frequency of the filter is equal to K_0 times the input sampling frequency. In order to make the future calculation simple, the number of the filter coefficients should be a multiple of K_0.
4) Create a sine wave with a frequency smaller than the passband frequency of the filter.
5) Create an array called "delay" to store the delayed value of the input samples. The length of this array is equal to $(\frac{\text{the number of filter coefficients}}{K_0}+1)$. Because the existence of the special case, the length should be increased by 1 here. Note because all the polyphase filters have the same input samples, only one array will be used here instead of K_0.
6) Calculate the control variable $p, q, n\ and\ \delta$.
7) If n increase by 1, one new input sample should be added to the array "delay" and the earliest value in delay should be removed.
8) Calculate the output of the polyphase filter specified by p and q.
9) Calculate the system output by using equation (5.3) and (5.4). If the value of p is not equal to $K_0 - 1$, use the equation (5.3). Otherwise, use equation (5.4).

The Matlab file "flexible_interpolator_li.m" is stored in CD .

Figure 5.13 shows the simulation result for $K_0 = 8$ and the interpolation factor is 3.3. The input signal is a sine wave with frequency f=50Hz. The input sampling frequency F_{s1} is 1000 Hz.

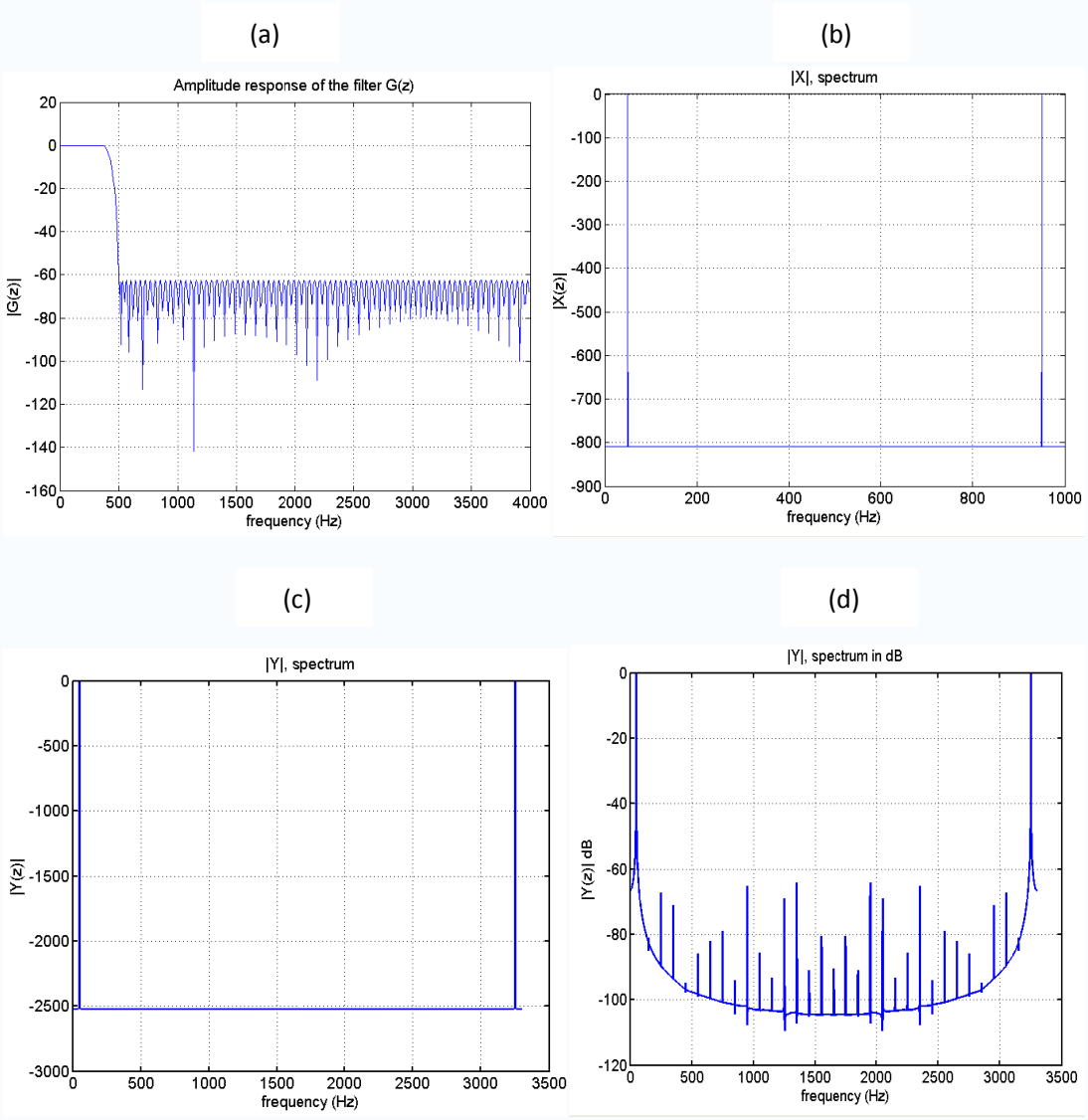


Fig 5.13 Matlab simulation result for flexible interpolator with factor 3.3, ($K_0 = 8$)

Fig 513(a) is the amplitude response of the interpolation filter G(z). The cutoff frequency is equal to half of the input sampling frequency. The attenuation in the stopband is set to be 60dB. So all the images created by the flexible interpolator should have attenuation at least more than 60dB. Fig 5.13(b) is the spectrum of the input signal plus a hamming widow. From this figure it is clear that the signal frequency is 50Hz and the sampling frequency is 1000Hz. Fig 5.13(c) and (d) are the spectrum of the output signal. Maximum values occur at 50Hz and 3250Hz tells that

new sampling frequency is 50Hz+3250Hz=3300Hz which is 3.3 times larger than the original non. All the images in between have been attenuated by 60dB. This accords with what has been set in the filter before.

Vary the input and output sampling frequency, the system still works. Hence the simulation result in matlab proved that the structure described in Fig 5.11 can be realized. The real-time implementation will be discussed in the next chapter.

5.3 Flexible decimator using transposed linear interpolator

As described in the previous chapter, a flexible decimator can be derived from a flexible interpolator using the transposition theorem. The transposed system corresponding to the circuit of Fig 5.7 is shown in Fig 5.14. The factors K_0 and K_1 are fixed integers and L is arbitrary integer in the range of $L \leq K_0K_1$.

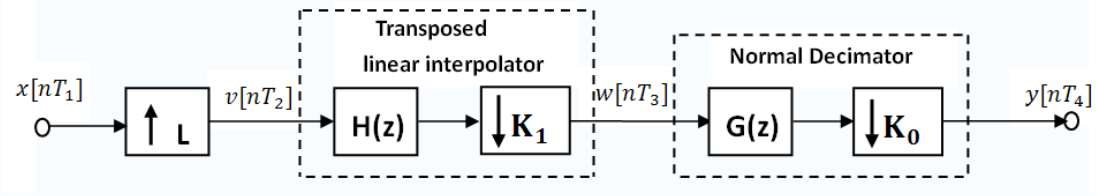


Fig 5.14 Flexible decimator using transposed structure

The transposed linear interpolator consists of a filter H(z) and an SRD with factor K_1. As described before, H(z) has a triangular impulse response with a length related to the factor K_1. Note the length here has no relationship with the interpolation factor L. The length is $2K_1 - 1$.

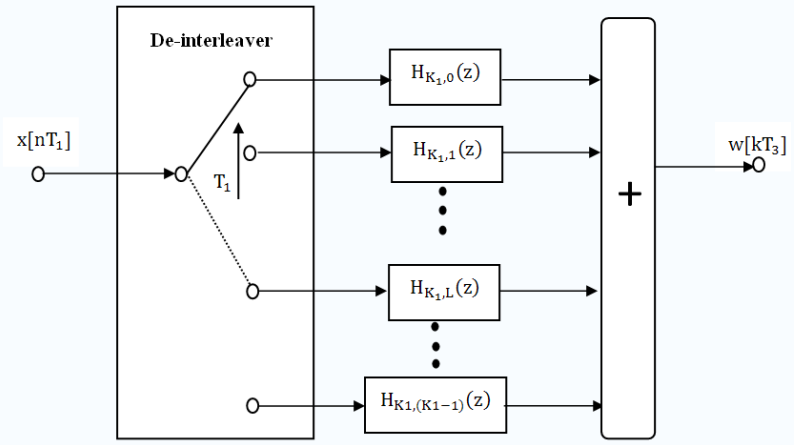


Fig 5.15 Efficient implementation of the input part of flexible decimator

To derive an efficient implementation of such a transposed system, it is necessary to separate the whole system into two parts. The SRI combined with a transpose linear interpolator will be called the input part. The rest will be called the output part. The analysis will start at the first part.

According to Fig 4.10, an efficient implementation of the input part is drawn in Fig 5.15. There are altogether K_1 polyphase filters. The de-interleaver rotate anti-clockwise with a time interval equal to $T_1 = LT_2$. The step size it rotates is equal to L. Filter H(z) is a causal linear-phase interpolating filter with a triangular impulse response of length($2K_1$-1). The i^{th} polyphase filter $H_{K_1,i}(z)$ has only two coefficients and can be represented as:

$$H_{K_1,i}(z) = \frac{i}{K_1} + \left(1 - \frac{i}{K_1}\right) z^{-1}$$
$$= \delta + (1 - \delta)\, z^{-1}$$
where $\delta = \frac{i}{K_1}$.

From the equation above it is clear that the first polyphase filter $H_{K_1,0}(z)$ has fixed coefficients equal to [0,1]. The condition shown in figure 4.10 is that $L \leq K_1 - 1$. If $L \geq 2K_1$, each *non-zero* sample is determined by only one input sample:

$$w[kT_3] = \delta x[nT_1]$$
$$w[(k+1)T_3] = (1 - \delta)x[nT_1] \qquad (5.5)$$

An example will be given here to illustrate the conclusion above. Suppose the following situation, L=7, $K_1 = 3$, The input sample $x[nT_1]$=[1,3,6] and the start point of the switch is at polyphase filter $H_{K_1,0}(z)$. So the step size of the switch is equal to L=7. Each time when the switch steps over the first polyphase filter $H_{K_1,0}(z)$, a zero value should be distributed to the polyphase filters which do not have an input. Therefore each polyphase filter will get the following input samples:

[1,0,0,0,0,0] for the first polyphase filter
[0,0,0,0,0,6] for the second polyphase filter
[0,0,0,3,0,0,] for the third polyphase filter

The impulse response of the three filters are $0+z^{-1}$, $1/3+2/3z^{-1}$, $2/3+1/3z^{-1}$,
So the outputs of the three filters are

[0,1,0,0,0,0,0] for the first polyphase filter
[0,0,0,0,0,2,4] for the second polyphase filter
[0,0,0,2,1,0,0] for the third polyphase filter

The final output is the summation of the outputs of three polyphase filters,

$w[kT_3]=[0,1,0,2,1,2,4]$

It is clear that $w[0]= 0*x[0]$, $w[1]= (1-0)*x[0]$

$w[3]=2/3 *x[1]$, $w[4]=(1-2/3) *x[1]$

$w[5]= 1/3*x[2]$, $w[6]=(1-1/3) *x[2]$

The three δ values here are 0 ,2/3 and 1/3.

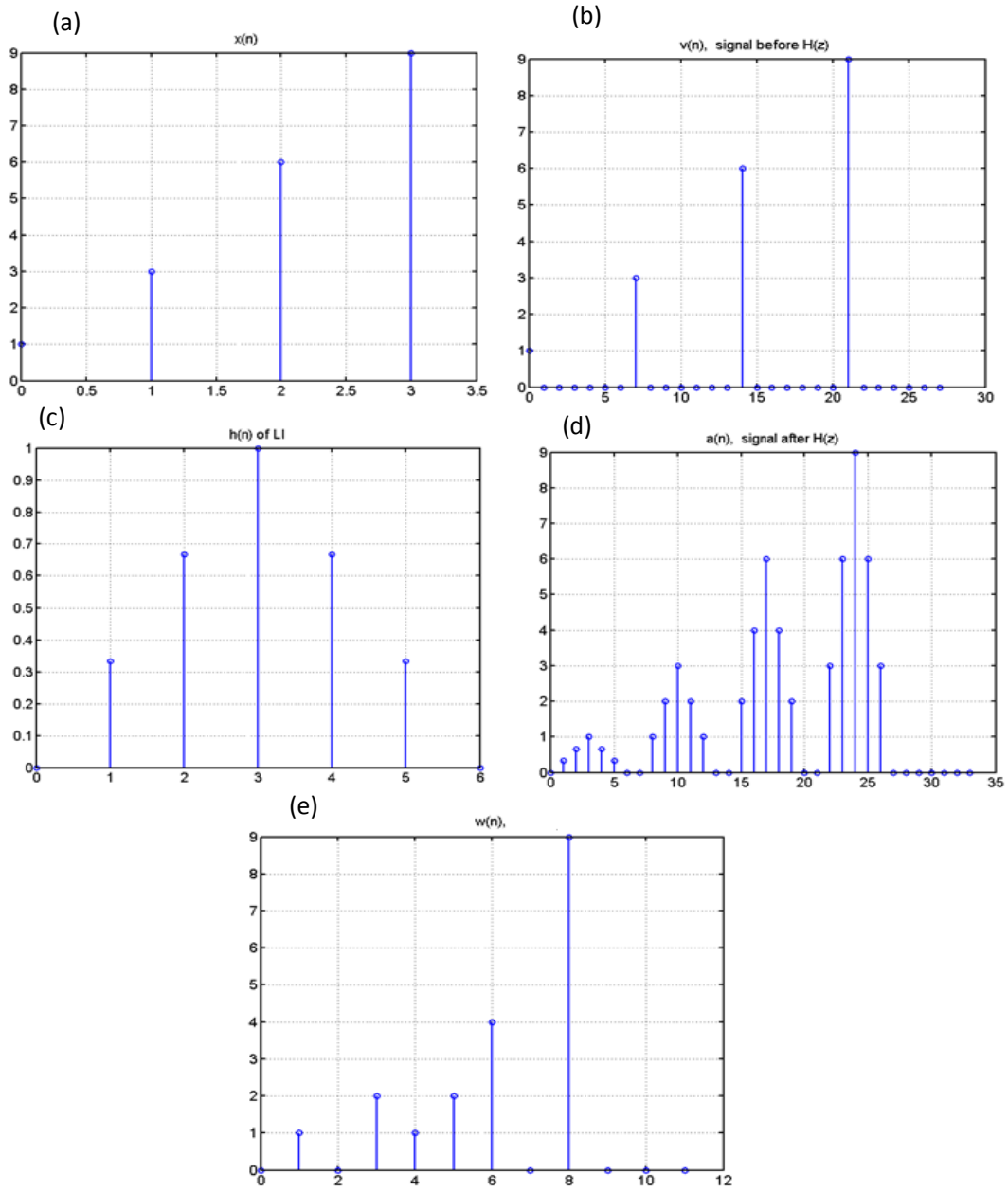


Fig 5.16 Time diagram for the input part of flexible decimator

The time diagram for L=7 and $K_1 = 3$ is shown Fig 5.16. x[n] is supposed to be the input samples. v[n] is the signal after the SRI. From Fig 5.16(b) it can be seen that 6

zeros have been added between each pair of the input samples. The impulse response of linear interpolator h[n]is shown in Fig 5.16(c). Now a new signal a[n] has been introduced in Fig 5.16(d). It is the signal after the linear interpolator H(z). It is equal to the convolution of v[n] and h[n]. Between every image of filter h[n], there are two zeros. Actually the number of the zeros can be calculated by the following equation:

$$number\_of\_zeros = L - 2K_1 + 1$$

The equation above proved that if $L \geq 2K_1$, non-zeros values of $w[kT_3]$ will be determined only by one input sample. However a larger L will lead to a better performance of the whole system.

The relation between the time index k of $w[kT_3]$ and the time index n of $x[nT_1]$ in equation (5.5) will be derived later in this section. A simple circuit shown in Fig 5.17(a) implements equation (5.5). "A" in the figure represents $x[nT_1]$, "B" and "C" represent $w[kT_3]$ and $w[(k+1)T_3]$. In a similar way used in the previous section, a new symbol shown in Fig 5.17(b) is introduced.

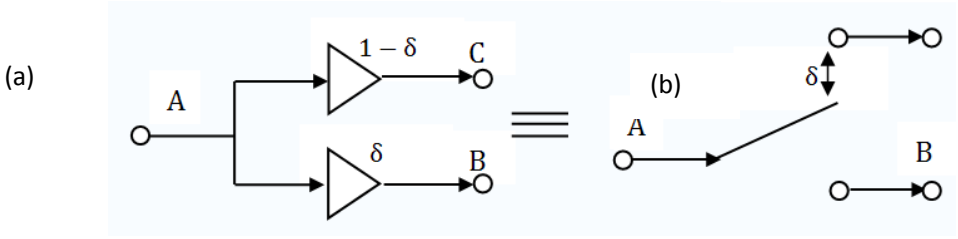


Fig 5.17 Symbolic representation of the input part of a flexible decimator

Until now the input part of the flexible decimator has been analyzed. The next step is to describe the efficient implementation of the output part of the system. The polyphase decomposition is used again to implement the combination of filter G(z) and the SRD with factor K_0. This is shown in Fig 5.18.

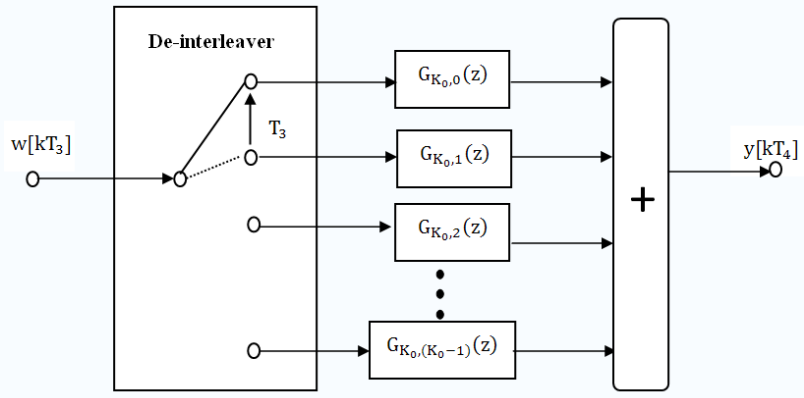


Fig 5.18 Output part of flexible decimator

An efficient implementation of the flexible decimator should combine the input part (shown in Fig 5.17) and output part (shown in Fig 5.18) together. This is can be done through the structure shown in Fig 5.19. The main issue still to be determined is the relation between n,k in equation (5.5) and the position of the switch in the 'linear-interpolation de-interleaver'. This position determine the exact values of p,q and δ.

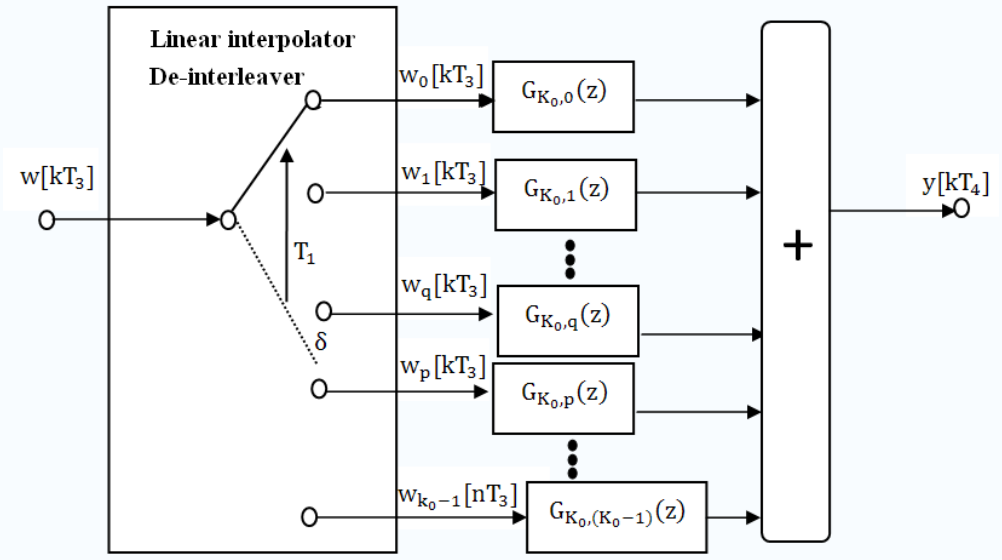


Fig 5.19 Final structure for flexible decimator using transposed structure

This can be solved in a similar way as what have been done in the flexible interpolator in the previous section. The switch rotates anti-clockwise and the step size X is a non-integer number:

$$X = \frac{K_0 T_1}{T_4} = \frac{K_0 F_{s4}}{F_{s1}} \qquad (5.6)$$

Every time that the switch has passed filter $G_{K_0,0}(z)$, the output index k, which indicates the time instance of the output sample, should be increased by one. At that time an output sample should be calculated by adding the outputs of all the polyphase filters, even they may have zero input. The two non-zero values can be obtained by a similar equation as given for the original interpolator structure. These equations describe the two interpolated signals $w_p[kT_3]$ and $w_q[kT_3]$ as a function of the input sample $x[nT_1]$.

$$w_p[kT_3] = \delta x[nT_1]$$

$$w_q[kT_3] = (1-\delta)x[nT_1]$$

It is clear that both samples are determined by δ. In normal situation, q=(p-1). The same time instance k for $w_p[kT_3]$ and $w_q[kT_3]$ will be used. Like the situation happens in flexible interpolator, there is also a special case. But this time the special case happens when p=0 .The corresponding value of q is equal to $K_0 - 1$. The equation changes to:

$$w_0[kT_3] = \delta x[nT_1]$$

$$w_{K_0-1}[kT_3] = (1-\delta)x[(n+1)T_1]$$

For $K_0 = 8$ and a total decimation factor is equal to 3.3, the step size X as given in equation (5.6) is equal to 2.424. That means during each output sampling interval, the switch steps over 2.424 positions in an anti-clockwise direction. The calculation results for the example above are shown in table 5.2.

Input index n	nX	p	q	δ	Output index k	Special case
0	0	0	-	0.000	0	no
1	2.424	6	5	0.576	1	no
2	4.848	4	3	0.152	1	no
3	7.273	1	0	0.727	1	no
4	9.697	7	6	0.303	2	no
5	12.121	4	3	0.879	2	no
6	14.545	2	1	0.455	2	no
7	16.970	0	7	0.030	2	yes

Table5.2: The control of the switch for the flexible decimator with a decimation factor equal to 3.3; (The value of $K_0 = 8$)

The control variables $p, q, k \text{ and } \delta$ can formally be calculated from nX, with n increase from 0 to infinite,

$$X = K_0 F_{s4}/F_{s1}$$

$$p = \{\text{-(integer part of nX)}\} \bmod K_0 \quad (5.7)$$

$$q = (p-1) \bmod K_0 \text{ (special case occurs if p=0)}$$

δ = 1-(non-integer part of nX)

$k = (nX + K_0 - 1)$ div K_0

One point must be mentioned here is that how to apply equation (5.7) in C environment. Suppose X=4.4 and $K_0 = 8$

			correct *p* value
For n=1,	-nX=-4.4	p=(-4)%8=-4	4
For n=2,	-nX=-8.8	p=(-8)%8=0	0
For n=3,	-nX=-13.2	p=(-13)%8=-5	3

By direct using equation (5.7), a wrong value of *p* will be got. The correct p values are shown on the right side. A few modifications should be added to correct this error.

```
p=(-floor(nX))% K0;
p=p+K0;
p=p%K0
```

Now the correct *p* values will be got.

5.4 Matlab simulation for flexible decimator

In this section the implementation of the flexible decimator specified in section 5.3 will be done in Matlab. The implementation has the following steps:

1. Set the input and output sampling frequency.
2. Set the value of K_0, the default value is 8.
3. Use Matlab command "remezord " to get coefficients of the filter G(z) according the input and output sampling frequency. The cut-off frequency is equal to half of the output sampling frequency. The sampling frequency of the filter is equal to K_0 times the output sampling frequency. In order to make the calculation simple in the future, the number of the filter coefficients should be a multiple of K_0.
4. Create a sine wave with a frequency smaller than the passband frequency of the filter.
5. Create an array called "delay" to store the delayed input samples for each polyphase filter. The length of this array is equal to the number of filter coefficients. Supposing the number of filter coefficients is equal to *N*, then the first N/K_0 elements in this array are the delayed input value for the first

polyphase filter $H_{K_0,0}$, the second N/K_0 elements are for the second polyphase filter $H_{K_0,1}$ and so on.

6. Create an array called "buf" with a length equal to $K_0 + 1$.
7. Calculate the control variable $p, q, k \; and \; \delta$.
8. If k increase by 1, one output value should be calculated. First right shift one value in array "delay". Then give the i^{th} value of array "buf" to j^{th} value array "delay", where j is calculated by the equation

$$j = (i - 1)\frac{N}{K_0} + 1$$

Calculate the output of each polyphase filter and add them together to be the final output sample. Set buf(1: K_0) to be zero. Pass the value of buf($K_0 + 1$) to buf(K_0) and set buf($K_0 + 1$) to be 0.

9. Pass $\delta x[n]$ to buf(p+1) and (1-δ)x[n] to buf(p). If p=0, then pass $\delta x[n]$ to buf(1) and (1-δ)x[n] to buf($K_0 + 1$). This is the special case described in section 5.3.

The Matlab file "flexible_decimator_li.m" is stored in CD.

Figure 5.20 shows the simulation result for $K_0 = 8$ and the decimator factor is 3.3. The input signal is a sine wave with frequency f=50Hz. The input sampling frequency F_{s1} is 1000 Hz.

Fig 5.20(a) is the amplitude response of the interpolation filter G(z). The cutoff frequency is equal to half of the output sampling frequency around 150Hz. The attenuation in the stopband is set to be 60dB. So all the aliasing effects created by the flexible decimator should have attenuations at least more than 60dB. Fig 5.20(b) is the spectrum of the input signal. A hamming window is applied here. From this figure it is clear that the signal frequency is 50Hz and the sampling frequency is 1000Hz. Fig 5.20(c) and (d) are the spectrum of the output signal. The maximum values occur at frequency 50Hz and 260Hz. This tells that the new sampling frequency is equal to 50Hz+260Hz which is nearly 3.3 times smaller than the original one. All the images in between have been attenuated by 60dB. This accords with what has been set in the filter before.

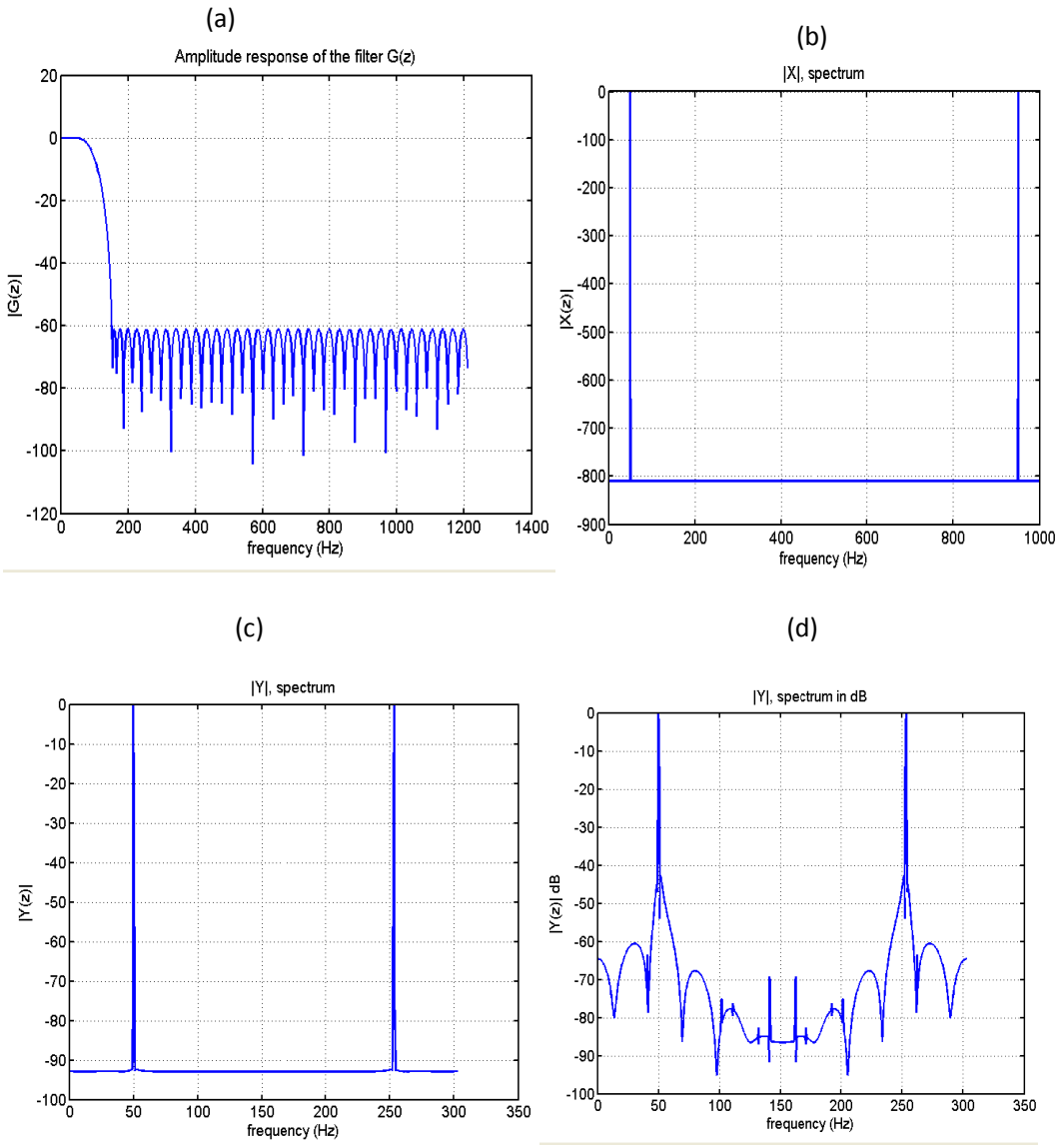


Fig 5.20 Matlab simulation result for flexible decimator with factor 3.3, ($K_0 = 8$)

Vary the input and output sampling frequency, the system still works. Hence the simulation result in Matlab proved that the structure described in Fig 5.19 can be realized. The real-time implementation will be discussed in the next chapter.

6 Real-time Implementation

A TI C6713 DSK board will be used in this chapter to make the real-time implementation of the flexible interpolator and flexible decimator specified in the last chapter. At first an introduction about this DSP board will be given. Through this interdiction, one can get a clear idea about the hardware circuit which will be used for the whole system. Then a few key points covered in the real-time implementation will be discussed. After that the final implementation will be explained in detail. At the final section of this chapter, some optimization policies will be found.

6.1 Introduction to TI C6713 DSK board

Digital signal processors such as the TMS320C6x (C6x) family of processors are like fast special-purpose microprocessors with a specialized type of the architecture and an instruction set appropriate for signal processing.

The C6713 is based on the very-long-instruction-word (WLIW) architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle. For example, with a clock rate of 225MHz, the C6713 is capable of fetching eight 32-bit instructions every 1/225MHz seconds. Features of the C6713 include 264 kB of internal memory , eight functional or execution units composed of six arithmetic-logic units (ALUs) and two multiplier units, a 32-bit address bus to address 4GB (gigabytes), and two set of 32-bit general-purpose registers.

The DSK package is powerful, yet relatively inexpensive ($395), with the necessary hardware and software support tools for real-time signal processing. It is a complete DSP system. The DSK board includes the C6713 floating-point digital signal processor and a 32-bit stereo codec TLV320AIC23 (AIC23) for input and output. It has 16MB of synchronous dynamic random access memory (SDRAM) and 258kB of flash memory. Four connectors on the board provide input and output. The voltage regulators on the DSK board provide 1.26V for the C6713 core and 3.3V for its memory and peripherals.

Fig 6.1 shows the DUETT board which is used to implement sample rate converters with arbitrary ratios in this thesis. Two PCM 3003 CODEC boards are connected to the TI DSK c6713 board. As is shown in the figure, both PCM3003 boards are mastered with respect to two unknown clocks (clock0 and clock1). The ADC part of PCM3003 #1 is used to sample the continuous input signal. Then the digital signal is passed to Multi-channel Buffer serial port 0(McBsp1) on the TI 6713 DSP. Then the interpolation or decimation policies mentioned in chapter 5 will be used to process

the input samples and calculate the output samples. At last McBsp1 pass the output samples to the DAC part of the PCM3003 #1 and get the final analog output signal.

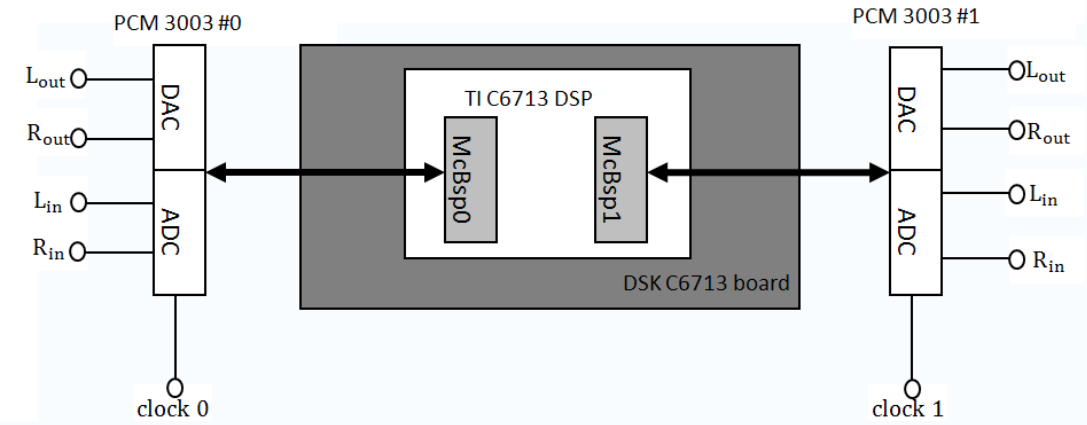


Fig 6.1 Hardware used for sample rate converter

6.2 Covering titles

6.2.1 Ratio Detection

The ratio of input and output sampling frequency is the base to derive an efficient implementation of interpolator or decimator. The step size of the switch shown in Fig 5.11 and Fig 5.19 depends on it. So the first task is to determine the ratio. As is already discussed in the section 1.2, the input and output sampling frequency is unknown. They can even be a function of time. The direct way to get the actual ratio is impossible. A fixed parameter should be found as a reference frequency. For example, suppose the reference frequency is equal to f_{ref}. Then two ratios $ratio1 = f_{in}/f_{ref}$ and $ratio2 = f_{out}/f_{ref}$ can be obtained. Finally the ration between f_{in} and f_{out} can be calculated by ratio1/ratio2.

In most DSP applications, interrupts have the control of starting a DAC or ADC. For a C6713 DSP, there are 16 interrupts. They can be issued internally or externally. In this implementation, three interrupts will be used. They are “Timer0 interrupt”, “McBsp0 interrupt” and “McBscp1 interrupt”. McBsp0 interrupt is used to start ADC of PCM3003#1 shown in Fig 6.1.It is issued externally by the clock0 shown in Fig 6.1. McBsp1 interrupt is used to start DAC of PCM3003#2 shown in Fig 6.1. It is also an external interrupt and initiated by the clock 1 shown in Fig 6.1. Timer0 interrupt is an internal interrupt. It is initiated by the onboard clock. For the implementation in this thesis, the onboard clock frequency is equal to 225MHz/4. Because a high frequency of Timer0 interrupt may affect the other two interrupts, this

frequency should be much smaller than either of other two. In this project, the frequency of Time0 interrupt is $f_{timer\_0} = 1098\text{Hz}$.

Three counters are set in each interrupt to count how many times each interrupt has been entered. When the counter of Time0 reaches $3 \times f_{timer\_0}$, the calculation for fs_{in} and fs_{out} will be started. That means the program uses 3 seconds to calculate a new ratio. In other words, this system has a delay for 3 seconds. Each time when the ratio changes, there will be a three-seconds waiting time during which no output will be given. The calculations can be obtained by the following equations:

$$fs_{in} = \frac{cnt\_McBSP\ 0}{cnt\_timer\ 0} \times f_{timer\_0}$$

$$fs_{out} = \frac{cnt\_McBSP\ 1}{cnt\_timer\ 0} \times f_{timer\_0}$$

$$ratio = \frac{fs_{in}}{fs_{out}}$$

6.2.2 FIR filter design

Lowpass filter is used to remove the image created by SRI or the aliasing effect caused by the SRD. In this thesis, the lowpass filter will be a FIR filter implemented by window method.

This window method is shown in Fig 6.2. The ideal frequency response of a lowpass filter with cutoff frequency equal to ω_0 is shown in Fig 6.2(a). It is a rectangular with length equal to $2\omega_0$. The corresponding impulse response is shown in Fig 6.2(b). The inverse fourier transform of a rectangular function is a si-function. It is symmetrical around t=0 and has an infinite duration. FIR states for finite impulse response, so the impulse response shown in Fig 6.2(b) should be truncated at some point. This can be done through sampling and multiplying by a rectangular window shown in Fig 6.2(c). The resulting finite unit sample of the FIR filter is shown in Fig 6.2(d). The frequency response of this FIR filter is shown in Fig 6.2(e). From this figure it is clear that it has overshooting and ripples. This is called Gibbs Phenomenon. In order to reduce this effect, a window that tapers smoothly to zero should be used to replace the rectangular window.

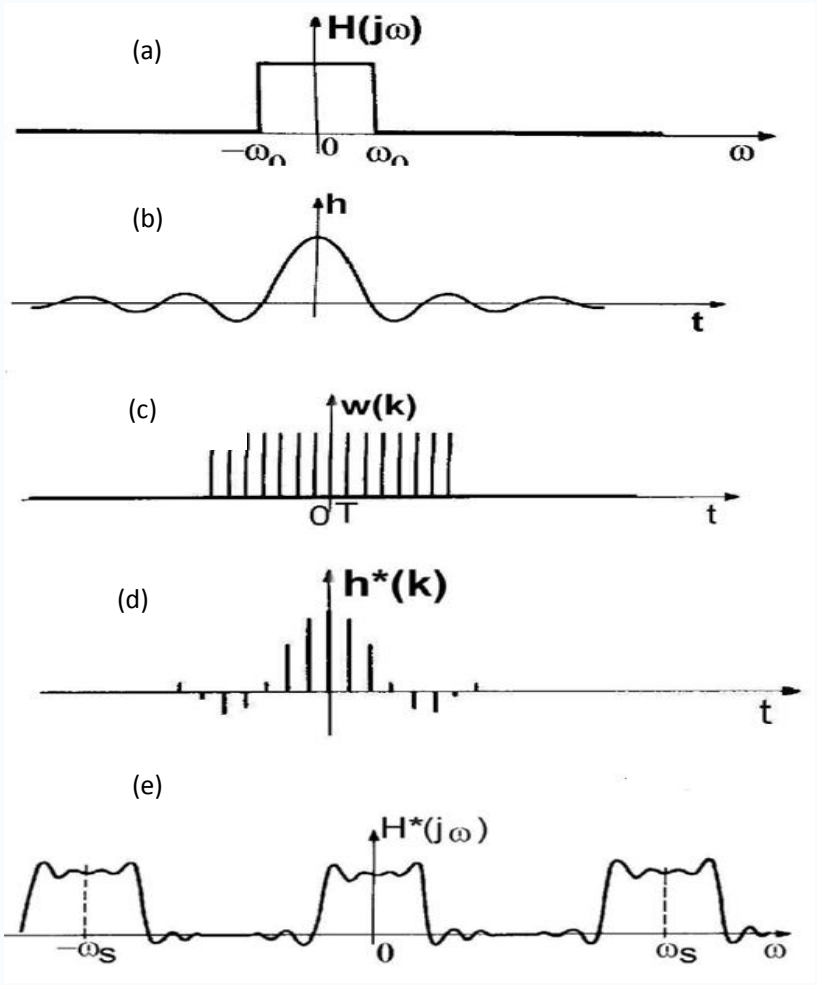


Fig 6.2 FIR filter design using a window method

Here a Hamming window will be used. The window function of a Hamming window is shown below:

$$w(k) = 0.54 - 0.46\cos\frac{2\pi k}{N}$$

N is the number of filter coefficients. The transition bandwidth is $6.6\pi(N+1)$. The main lobe to the first side lobe is 41dB. Comparison of the rectangular window and hamming window are shown in Fig 6.3. It is clear that the overshot is reduced. But the transition bandwidth is increased due to the increasing of the main lobe width of the window function.

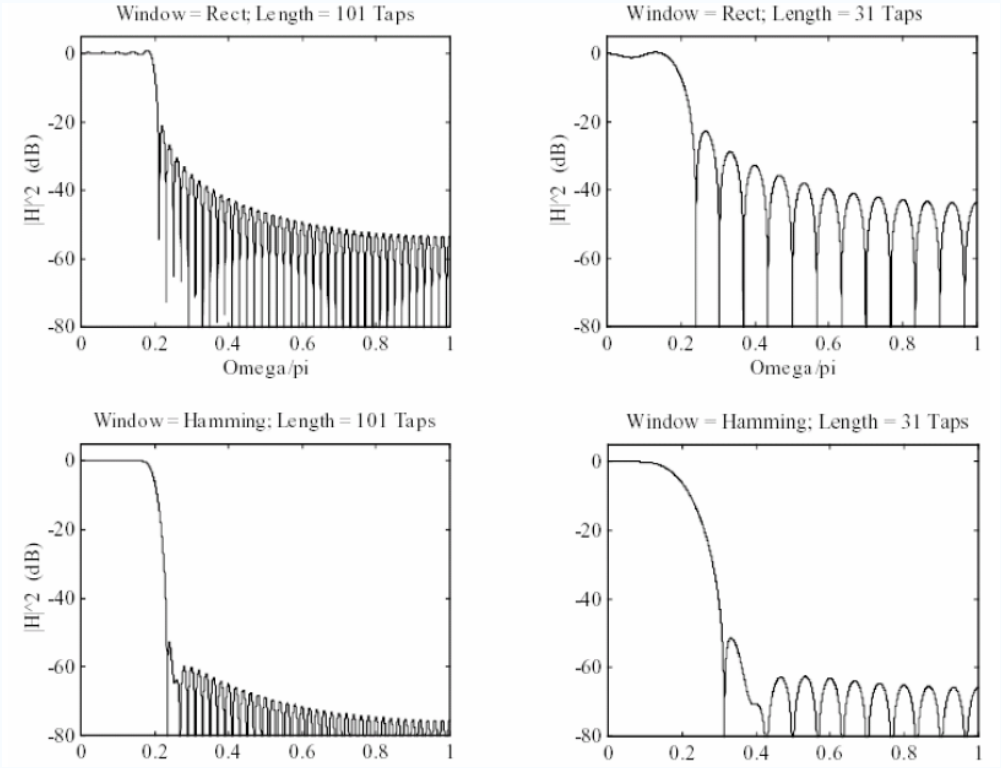


Fig 6.3 Comparison of the rectangular window and Hamming window

Suppose the sampling frequency is fs. The passband frequency is f_pass and the stopband frequency is f_stop. A lowpass FIR filter implemented by a Hhamming window can be obtained the following steps:

1. Calculate N by the following equation

N=3.3*fs /(f_stop-f_pass);

2. Calculate the normalized passband frequency ω_g

$$\omega_g = 2 * \pi * \text{f\_pass/f\_out;}$$

3. Calculate $h_d(n)$

$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_D(\omega)e^{j\omega n}d\omega = \frac{\omega_g}{\pi}\,\text{sinc}(\frac{\omega_g n}{\pi})$$

4. Calculate h(n)

$$h(n) = h_d(n)w(n)$$

where $w(n) = 0.54 - 0.46\cos\frac{2\pi n}{N}$

Until now, the function used to create FIR filter coefficients and delays still can't be written. The reason will be explained in the next section.

6.2.3 Memory Allocation

If one wants to define an array in C program, a certain length of this array should be given. For example,

```
short int h[10]
```

The length of this array is equal to 10. When you compile this command, the C complier will allocate 10*16bit memory space for it. These memory spaces are on stack. The following commands doesn't equal to the command above.

```
short int N=10
short int h[N]
```

This is because the C compiler doesn't accept a variable as the length of an array. It will report error when compile it.

But in our implementation, the second situation is actually needed. As is described in chapter 5, the character of the filter depends on the ration of the input and output sampling frequency. If $fs_{in} > fs_{out}$, the system should acts as a flexible decimator. In this situation, the sampling frequency of the filter should be equal to $K0 \times fs_{out}$ and the cutoff frequency should be $fs_{out}/2$. If $fs_{in} < fs_{out}$, the system should acts as a flexible interpolator. In this situation, the sampling frequency of the filter should be equal to $K0 \times fs_{in}$ and the cutoff frequency should be $fs_{in}/2$. Moreover the two sampling frequency fs_{in} and fs_{out} could be a function of time. It is no doubt that the length of the filter *N* is a function of time. The array h[N] used to store the filter coefficients can not be created through the normal way. This is why the function used to generate the filter coefficients and delays can't be achieved.

The command "void *malloc(size_t size)" in standard library can solve this problem. This command dynamically allocates *size* byte memory spaces on heap. Parameter "size_t" can be any standard data type in C. The return value of this command is a pointer points to the starting address of this memory spaces. To allocate the same array dynamically, the following codes can be used

```
short int N=10;
short *h;
h=malloc(sizeof(int)*N);
```

The main advantage of the this dynamic method is that when these memory spaces are not deeded, they can be simply reclaimed. The simple command below can achieve this function,

```
free(h);
```

After this command, all the memory space allocated by “malloc” before will be released.

This pair of commands is very important for our implementation. Each time when a new ratio is detected, the program will first release the memory space used to store the old filter coefficients and delays. Then it will allocate new memory space for the new filter coefficients and delays.

Now a function called “fir_design” can be programmed to create the filter coefficients and the delays. This function is part of the final code flexible_SRC.c stored in CD. It first calculates the number of filter coefficients N according to the specified frequency. In order to make the filter coefficients for all the polyphase filters same, N should be a multiple of K0 (in the program LL= K0). Then it will uses the method specified in section 6.2.2 to get the filter coefficients and stores them in array “b_float”. A new array “h[N]” will be created. This array is used to store the filter coefficients in an integer type. That means h[i]=b_float[i]*32768, i=0,1,2…N-1. The next step is to create an array “b_filter”to store the filter coefficients for each polyphase filters. The first N/LL elements in this array are the filter coefficients for the first polyphase, the second N/LL elements are for the second polyphase filter and so on. At last create an array called “delay” to store the delayed input value for each polyphase filter.

6.2.4 Fixed-point Optimization

From the acknowledgement in chapter 5, the calculations for the control variables p, q and δ are based on the step size of the switch X. It is clear that X is a non-integer number. Although the C6713 DSP supports the floating –point calculation, the time consumption of the floating-point is much larger than that of a fixed-point calculation. If a fixed-point calculation can be applied, the sampling frequency of the whole system can be increased obviously.

That means the date type of X in the code should be an “int” or “short int” instead of a “float”. This can be simply realized by the following example.

Suppose X=5.4823, then the representation of X with a type “short int” is

```
short int X_integer=floor(X);
short int X_non_integer=(X-X_integer)*32768;
```

In this two-lines codes, command "`floor(X)`" returns the integer value of *X*. So the integer part of X will be stored in variable "X_integer" and the non-integer part of X will be stored in variable "X_non_integer". Because the maximum value of a variable with data type "short int" is 32768, the non-integer part of X will be amplified by a factor 32768. This means the non-integer part of X will have accuracy equal to 15bit.

For the decimation situation, the values of p, q and δ will be obtained by the following codes:

```
dis_integer+=X_integer;
dis_non_integer+=X_non_integer;
dis_integer+=(dis_non_integer>>15);
dis_non_integer=dis_non_integer  & 32767;
p_pointer=-dis_integer;
p_pointer=p_pointer%LL;
// the following two lines correct the error caused by
//C development environment, the reason can be found
// in the last part of section 5.3
p_pointer+=LL;          //LL is equal to K0
p_pointer=p_pointer & SRD_mask; // SRD_mask=LL-1
delta=dis_non_integer;
```

"dis_integer" and "dis_non_integer" have a data type "int". Variable "dis_integer" is the integer part of the total distance the switch has rotated and variable "dis_non_integer" is the non-integer part of the total distance the switch has rotated. The step size of the switch is *X_integer*+*X_non_integer*. The variable "p-pointer" means the value of p. Variable "delta" is equal to the non-integer part of the total rotated distance. All variables can be obtained by a fixed-point calculation.

For interpolation, the method to calculate the total distance the switch has rotated has not been changed. Only the methods to get the values of p and δ are different. Form the code below it is clear that the calculation complexity is easier than the decimation.

```
p_pointer=dis_integer & (LL-1);   //LL is equal to K0
delta=dis_non_integer;
```

The value of p is equal to the dis_integer mod LL. Like the decimation case, all the calculations are calculated by fixed-point calculation.

One thing must be mentioned here is how to calculate the value of 1-δ. Because the largest value for the non-integer part is 32767, that means 32768 will be seen as 1. So "1-δ" will be equal to "32768-delta" in the code.

Fig 6.1 shows that the new sample form the PCM3003#0 will be transmitted to McBsp0. McBsp0 has 32 bits. So each channel has 16 bits. That means each new input sample will be represented by 16bits. 16 bits is actually a data type "short int". Fig 6.1 also shows that McBsp1 passes the output sample to PCM 3003#1. So the output sample will also have a data type "short int".

Until now it can be concluded that all the variable x[n],y[n],h[n] and δ are represented by a data type "short int". No floating-point calculations exists.

6.3 Code structure explanation

6.3.1 Interpolation Part

The program uses the structure shown in Fig 6.4 to implement a flexible interpolator. In this figure, the interpolating filter G(z) has been decomposed into four polyphase filters for simplicity. For general case, the number of polyphase filter will be K0. A buffer called "input_buf" with length 64 added before these polyphase filters. This buffer is important for synchronization which will be discussed later. Two indexes "buf_in" and "buf_out" are used to control the input and output of this buffer. "buf_in" indicates the position where a new sample should be stored. When a new sample x[n] is obtained, it will be stored in buffer at position "buf_in". "buf_out" indicates the sample which should be taken out, when the switch of the interleaver cross G(3). The initial values of them are 0 and 32. Theoretically the increasing speed of these two should be the same. That means the difference of the two indexes should be a constant value 32. But in real, this difference varied. An error will occur, when the difference of the two indexes is zero. It is obviously that a large buffer leads to a small possibility of error. But a larger buffer causes more time in address allocation and more space in memory. This leads to an inefficient structure. Hence the length of this buffer "input_buf" is chosen to be 64 as a compromise of the two factors above.

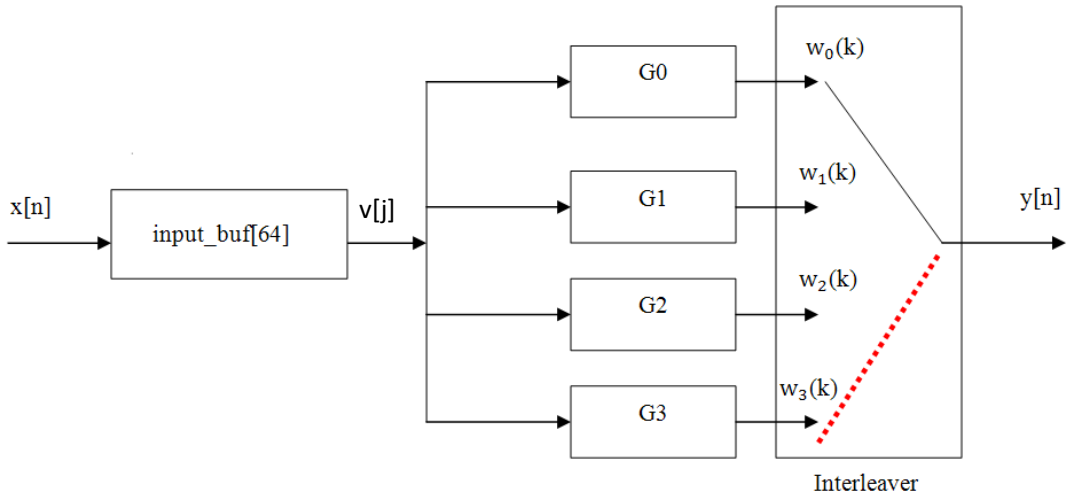


Fig 6.4 Structure for the real-time implementation of a flexible interpolator

As what has been explained in section 6.2.1, three interrupts have been used in the implementation of a flexible interpolator.

The function of Timer0 interrupt

1. Disable the input and output calculation in other two interrupts.

   There is a control variable "stable".

   stable=0, stop the calculation of interpolator

   stable=1, start the calculation of interpolator

2. Using three seconds to calculate the input and output sampling frequency and the ratio of them. The calculation time has been tried from 1up to 5 seconds. The results show that when larger than 2 seconds, the results are almost the same. That is the reason why three-second calculation time is used here.
3. Call the function FIR_Design to design an FIR filter. The filter coefficients are calculated by equation sin**x/x.**
4. Initialization. This is done through giving the initial value to all control variables in other two interrupts.
5. Disable Timer 0 interrupt. After the ratio has been calculated, Timer0 interrupt is no longer needed.

The function of McBsp0 interrupt

1. Get the new input sample and store it in buffer "input_buf[64]".
2. Shift the value in array "delay". This array is used to store the delayed input

values for each polyphase filter. The length of it is the order of polyphase filter *plus 3*. Why additional 3 delays are added will be explained by a real example later in this section.

The number of values which should be shifted in array "delay" is determined by a control variable "shift_times". The value of it is controlled by another interrupt "McBsp1"**.** When the value of variable "shift_times" is larger or equal than 1, we shift the value in array "delay". The new values are taken from buffer "input_buf".

shift_times=1, take one value from "input_buf" and shift for one times

shift_times=2, take two values from "input_buf" and shift for two times

shift_times=n, take n values from "input_buf" and shift for n times

After that the value of "shift_times" is set to be 0.

3. Calculate the new ration of input and output sample frequencies. If the difference of new ratio and old ratio larger than 0.01, enable Timer0 interrupt. This ensures the automatic ratio detection

The function of McBsp1 interrupt

1. Calculate the p and delta value used for linear interpolation calculation.
2. Calculate the output of polyphase filter specified by p and p+1.
3. Calculate the final output using linear interpolation algorithm.
4. Calculate the value of "shift_times". The value of this variable is very important for synchronization. To get this value, we should first know how many rounds the interleaver has rotated, this can be achieved by the following equation

rotate_round=dis_integer/LL

where *LL* is the number of polyphase filters
dis_integer is the integer part of the total distance which the switch has rotated

When "*rotate_round*" is increased by 1, "*shift_times*" will be set to 1. If "*rotate_round*" is increase by n, then "*shift_times*" will also be set to n.

The explanation of why the length of array "delay" should be added by 3 will be given here. Suppose v[j]={1,2,3,4,5,6,7,8,9,10} and the length of the filter is equal to 6. When the interleaver rotates to the red position shown in Fig 6.4, a special case will be got. The output is equal to (1-delta) $w_3(k)$ +delta× $w_0(k+1)$. Assuming j=6 in that time, then $w_3(k)$ and $w_0(k+1)$ will be obtained by the following equations:

$w_0(k+1) = \{2,3,4,5,6,7\} * g_0(n)$
$w_3(k) = \{1,2,3,4,5,6,\} * g_3(n)$
where "*" means convolution and $g_k(n)$ is the impulse response of the k^{th} polyphase filter

It is clear that if the length of *delay* is 6 (the order of the polyphase filter), $w_0(k+1)$ cannot be calculated. The length of array "*delay*" should be added by one to overcome such problem.

Now the second reason for increasing the length will be given. Suppose the time consumption of the CPU for interrupt "McBsp0" is 0.5μs and for interrupt "McBsp1" is 0.2μs. The following situation shown in Fig 6.5 may happen. The beginning of the solid rectangular indicates "McBsp0" occurs and the ending of solid rectangular indicates "McBsp0" stops. The beginning of the hollow rectangular indicates "McBsp1" starts and the ending of hollow rectangular indicates "McBsp1" stops.

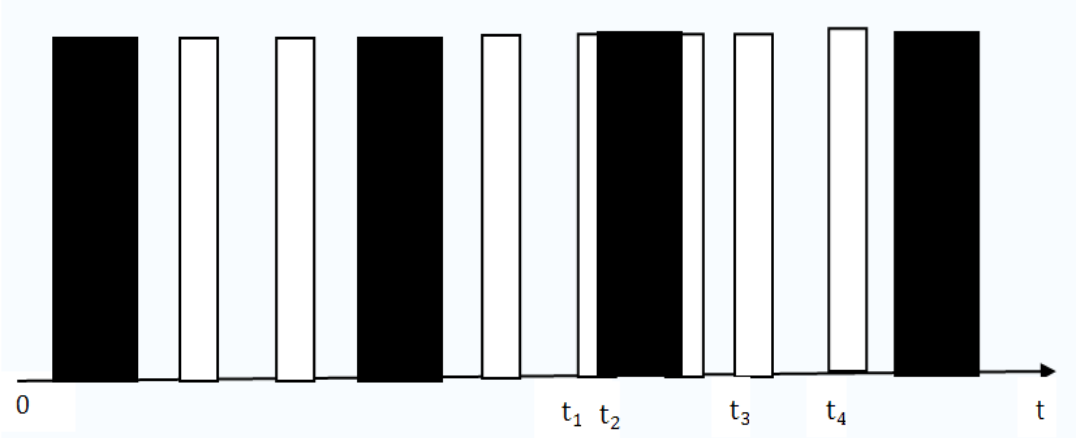


Fig 6.5 Time consumption of two interrupts

It is clear that everything is in order before time t_1. At time t_1 the program goes to "McBsp1"interrupt. At time t_2, "McBsp0" interrupt is initiated. Because "McBsp0" has a higher priority than "McBsp1", the program will jump to "McBsp0". But at this time the value of "*change_times*" will be zero. Therefore no shifting occurs for the array "*delay*". 0.5μs later, the program jumps to interrupt "McBsp1" again. Now the value of "*shift_times*" has increased one. That means the value in array "*delay*" should be shifted by one. At time t_3 and t_4, the program will jump to "McBsp1". The value stored in array "*delay*" will be used to calculate the filter output. As discussed before, the value in array "*delay*" has not been changed. It is no doubt that an error will be introduced in this case. Increase the length of array "*delay*" by one can solve the problem. Suppose the value in array "*delay*" is {1,2,3,4,5,6,7} and the length of polyphase filter is 6. At time t_1, the first six values in array "*delay*" are used to calculate the output of the filter. At time t_3 and t_4, {2,3,4,5,6,7} will be

used to calculate the filter output.

The worst situation may occur, when the two situations above happens together. In this case, the length should be increased by 2. For safety reason, the length of array "*delay*" is increased by 3.

6.3.2 Decimation Part

The program uses such a structure to implement a flexible decimator. Here assuming there are altogether 8 polyphase filters.

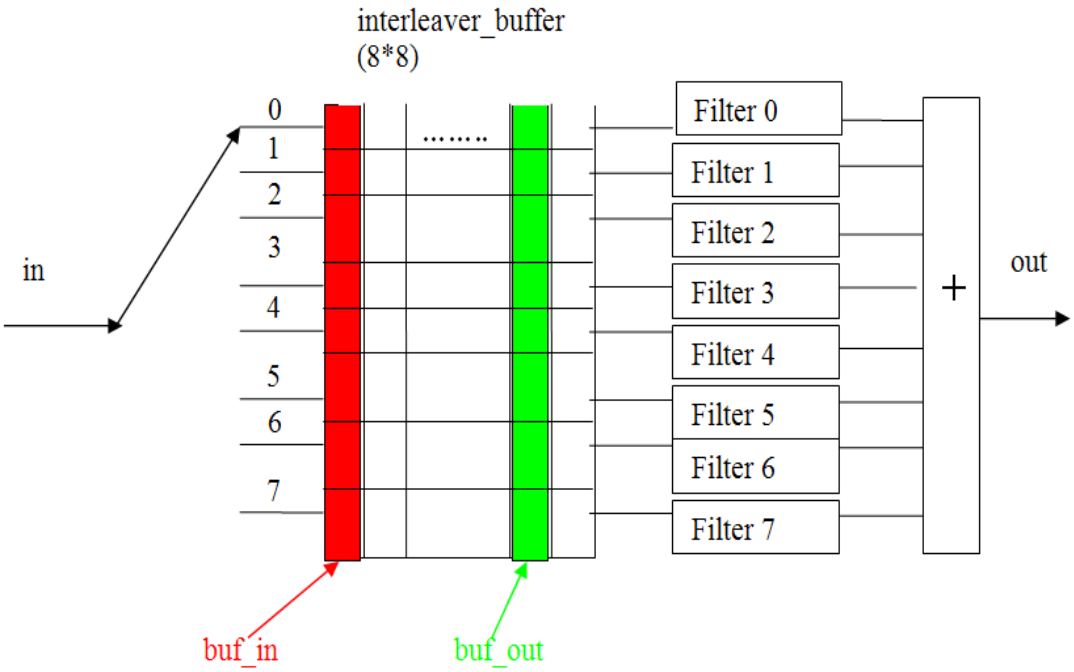


Fig 6.6 Structure for the real-time implementation of a flexible decimator

Firstly the input part of a flexible decimator will be explained. The de-interleaver rotates anti-clock wise with a fixed frequency equal to the input sample frequency. The rotate distance for each time is $X = LL * Fs_{out}/Fs_{in}$, where LL is the number of polyphase filters**.** When it rotates to the corresponding position, it will store two values in the red column. These two values are calculated by equation (5.5). This red column is specified by index "*buf_in*" .

If the de-interleaver exceeds 0, which means one round has rotated, "*buf_in*" will be increased by one. There is a special case, it occurs when the de-interleaver rotates between 0 and 7. In this case, one value will be stored in row 0 column ("*buf_in"*+1) and the another value will be stored in row 7 column "buf_in".

In this program, "McBSP0" interrupt corresponds to the input part. Now the code will be separated into several parts.

```
if (dis_integer>=1073741824){  //"40000000"hex
  buf_initial=buf_in-1 & mask;
  dis_integer=(dis_integer & mask );
}
```

The code above deals with the situation when *dis_integer* has an overflow. From section 6.2.4, it is clear that *dis_integer* has a data type "int". The largest value it can get is 2^{31}. In order to avoid overflow, the largest value will be set to 2^{30}. The value of *mask* here is equal to LL-1. *buf_initial* is the initial position of the de-interleaver when *dis_integer* is larger than 2^{30}.

After that, the value of control variable will be calculated. The code shown in section 6.2.4 will be used here.

The last part is to put the input value in the proper position in the buffer. This achieved by the following codes

```
if(!p_pointer){  //special case
     interleaver_buffer[0][buf_in]=((32768-delta)*inL)>>15;
     //buf_len is the length of buffer and equal to LL
     if(buf_in==buf_len-1)
          interleaver_buffer[LL-1][0]=(delta*inL)>>15;
    else
          interleaver_buffer[LL-1][buf_in+1]=(delta*inL)>>15;
    }
else{     //normal case
    interleaver_buffer[p_pointer]  [buf_in]= ((32768-delta)*inL)>>15;
    interleaver_buffer[p_pointer-1][buf_in]= (delta*inL)>>15;
    }
```

Three situations will be found in the code above instead of 2. The new added situation is that *p_pointer*=0 and *buf_in*=LL-1. In this case, one value will be stored in row LL-1 column LL-1 another will be stored in row 0 column 0.

The output part will be explained now. Every 1/fs_out seconds, a column from the

interleaver_buffer will be taken out and sent as input to eight poly-phase filters. These values are represented by the green column. Another index "*buf_out*" points to that column. After eight values in that column have been taken, clean the green column (making all values in that column to be 0) and increase "*buf_out*" by one.

Theoretically, like in interpolation situation, the distance between "buf_in" and "buf_out" is also fixed and equal to 4(assuming 8 polyphase filters are used as shown in Fig 6.6). But due to some unknown errors, the distance may be 0. In this case, an error will be created. Synchronization should be applied here. The final output of the flexible decimator is the sum of all the outputs of the polyphase filters.

Interrupt McBSP1 corresponds to the output part. In this interrupt, a new array "*delay*", which has a length subfilter_order*LL, is introduced. The first "subfilter_order" elements store the delayed values of the first poly-phase filter, the second "subfilter_order" elements store the delayed values of the second poly-phase filter and so on. Each time the program jumps to this interrupt, it will first shift the values in "*delay*" by one. Then copy the value of green column from "interleaver_buf" and give them to "*delay*" at position

i*subfilter_order-1**,** where "*i*" is integer between 1 and LL-1.

After that, all values of that green column will be set to zero. And then "*buf_out*" will be increased by one.

At last calculate the output of each poly-phase filter and add them together. The summation is the final output value.

6.3.3 Flexible SRC

After successful implementation of flexible interpolator and flexible decimator, a flexible SRC can be implemented simply. A state variable "*stable*" will be used to indicate the status. At first *stable* will be set to zero. The program will detect the ratio of the input and output sampling frequency. If ratio is larger than 1, the variable *stable* will be set to 2. If the ratio is smaller than 1, the variable *stable* will be set to 1. In the interrupts McBsp0 and McBsp1, there will be an "if" statement which depends on the value of variable "*stable*". Therefore the program will choose either flexible interpolator or flexible decimator according the value of variable *stable*.

7 Testing

7.1 Testing by Sine Waves

Until now the whole implementation has been finished. The current step is to test the result. The first testing method will be passing a sine wave to it and looking at the output signal.

In this test, the input sampling frequency is controlled by fixed quartz with frequency 6.144MHz. So the input sampling frequency will be 6.144MHz/256=24KHz. The output sampling frequency is controlled by a function generator. This function generator will generates a square wave with frequency up to 10MHz. So when this frequency is larger than 6.144MHz, the system will act as a flexible interpolator. When the frequency is smaller than 6.144MHz, the system will act as a flexible decimator. Note there is a limitation of the DSK board. If the output sampling frequency is smaller than 8 KHz, the output signal will have a distortion. Hence the minimum frequency of the square wave generated by the function generator is 8KHz*256=2.048MHz.

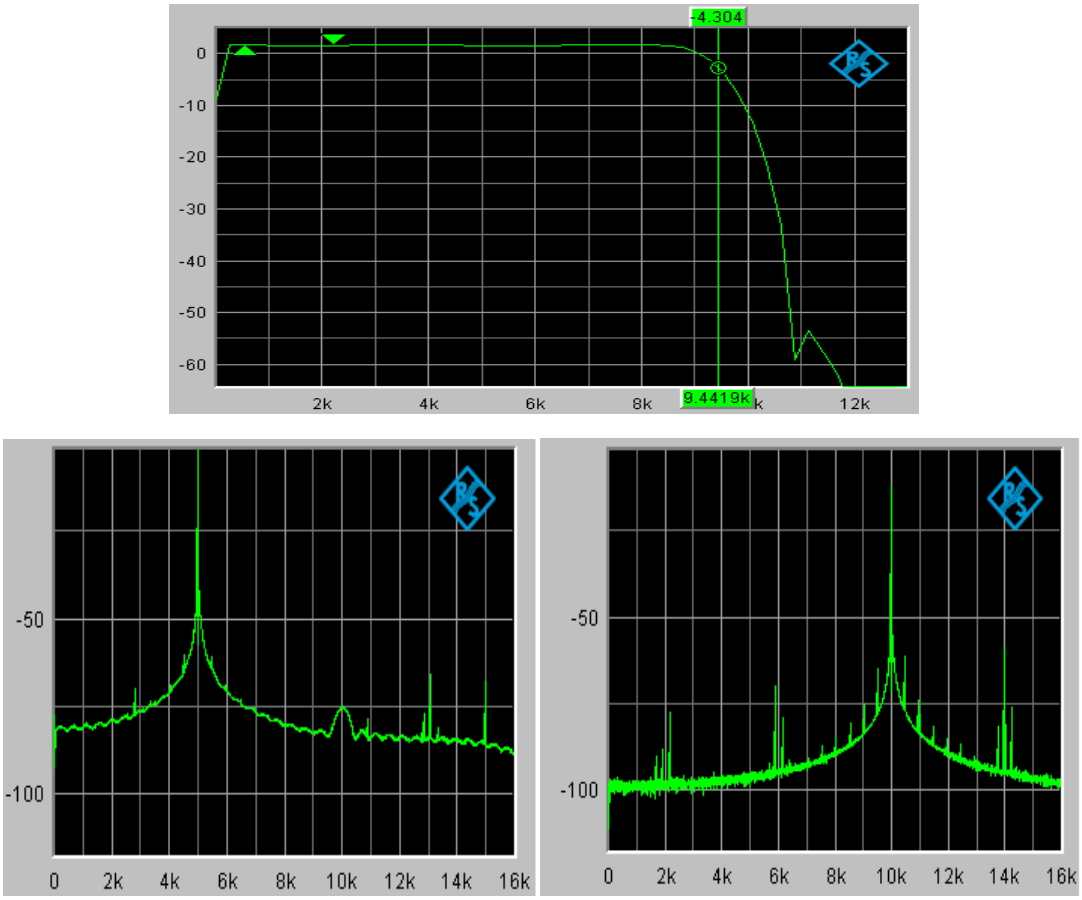


Fig 7.1 Testing results flexible interpolator:Fs_in=24KHz, Fs_out=32.07KHz

Fig 7.1 shows the testing results for a flexible interpolator. The input sampling frequency is 24 KHz and the output sampling frequency is 32.07 KHz. The upper figure is the frequency response of the whole system. As what has been expected, it is actually a lowpass filter with cutoff frequency equal to 12 KHz. From the C-code it is known that the cut-off frequency of the FIR filter is Fs_in/2=24KHz/2=12KHz. The measurement proved that the filter is set correctly. The left figure of the lower part in Fig 7.1 shows the spectrum of the output signal. A hamming widow is applied in the calculation of FFT. The input signal is a sine wave with frequency 5 KHz. From this spectrum it is clear that all the images introduced by SRI have attenuations more than 60dB. The right figure of the lower part in Fig 7.1 gives the situation that the input sine wave has a frequency equal to 10KHz. From the frequency response of this system it can be seen that 9.5 KHz is in the transition bands. Therefore the attenuation of the maximum value in this images is around 10dB down.

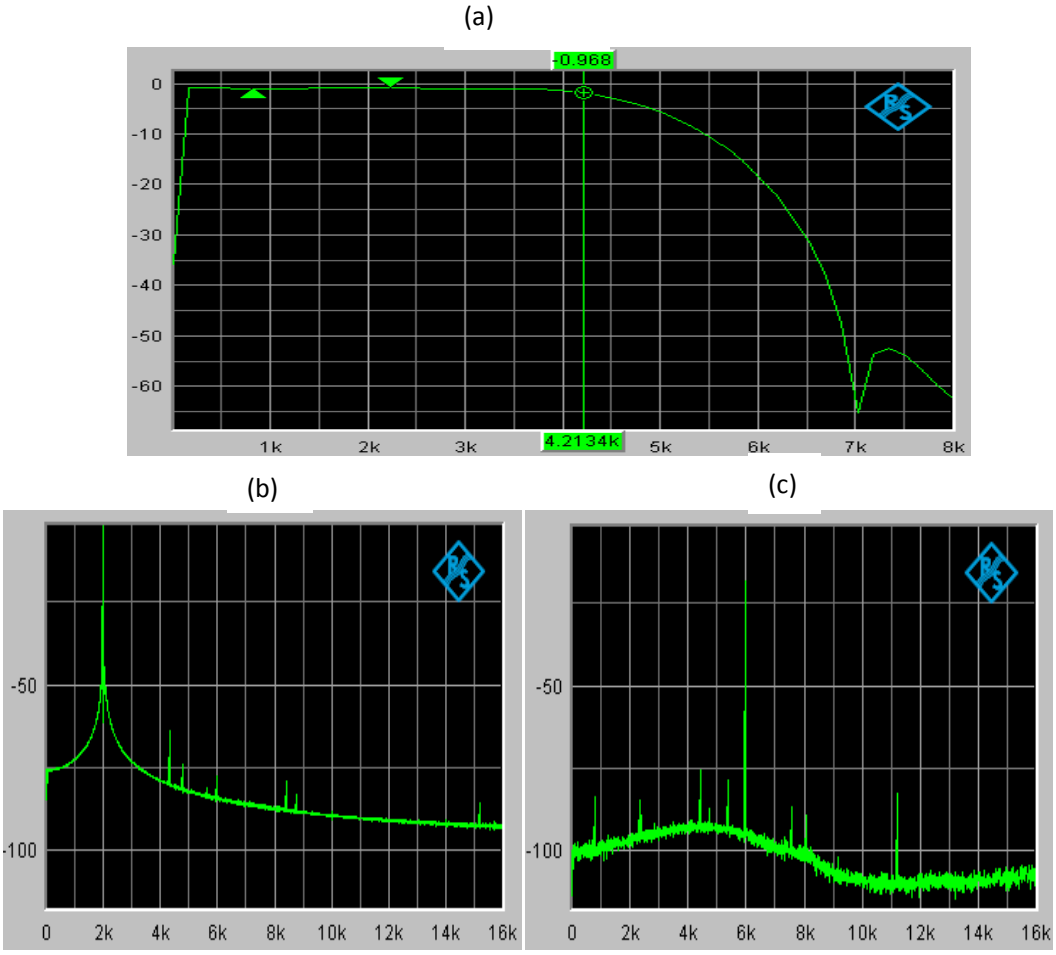


Fig 7.2 Testing results for flexible decimator: Fs_in=24KHz, Fs_out=13.59KHz

Fig 7.2 shows the testing results for a flexible decimator. The input sampling frequency is 24 KHz and the output sampling frequency is 13.59 KHz. Fig 7.2(a) is

the frequency response of the whole system. It is also a lowpass filter with cutoff frequency round 7 KHz which is exactly Fs_out/2. Fig 7.2 (b) shows the spectrum of the output signal. The input signal is a sine wave with frequency 2 KHz. From this spectrum it is clear that all the images introduced by SRD have attenuations more than 60dB. Fig 7.2 (c) gives the situation that the input sine wave has a frequency equal to 6KHz. From the frequency response of this system it can be seen that 6 KHz is in the transition bands. Therefore signal power is about 20dB compare to that in Fig 7.2(b).

Varying the input and output sampling frequencies, the system still works.

Another test is passing an audio signal to the system and connecting the output signal to a speaker. Different input and output sampling frequency are used here. The results are acceptable. There are no unwanted noise coming from the speaker and the content of the music can be heard clearly.

7.2 System Performance

This section gives the information that how much time the two interrupts (McBsp0 and McBsp1) need. It is clear that the order of FIR filter is the key factor to determine the time consumption. It is clear that the time consumption for a flexible interpolator or a flexible decimator is also different.

The software CCS offers the function to look at the time consumption of an interrupt. Table 7.1 shows the time consumption of the two interrupts with different filter order for a flexible decimator. File o-3 optimization are used here. The time here are represented by the CPU cycles. Each cycle is equal to $\frac{1}{225\,\mathrm{MHz}} \times 4 = 0.018\mu s$.

Order of FIR filter	McBsp0	McBsp1
89	435	2313
105	433	2587
133	435	3036
177	434	3772

Table 7.1 Time consumption for a flexible decimator

From Table 7.1 it can be seen that the time consumption of interrupt McBsp0 is unchanged when the order of FIR increases. The time consumption of interrupt McBsp1 increased linearly when the order of FIR filter increases. The relation can be described by the following equation:

$$\Delta T \approx 17\Delta N$$

where ΔT is the time increment and ΔN is the increment of the order of FIR filter. In this thesis, the Table 7.1 also shows that interrupt McBsp1 needs much more time than interrupt McBsp0. This result is no surprising. In interrupt McBsp1, all the outputs of polyphase filters should be calculated. Huge amount of time is needed to calculate these outputs. This is one drawback of this implementation. The usage of CPU may be around 30% for a long time and suddenly go to 100% in a short time. Such drawback limits the maximum output sampling frequency for a flexible decimator.

The same method is used to calculate the time consumption of the two interrupts for a flexible interpolator. Unfortunately this method doesn't work. The clock cycles used for these two interrupts are varied in time.

Thanks for the onboard LEDs, another testing method can be applied here by using these LEDs. The C6713 DSK board has 4 on board LEDs. When the program goes to interrupt McBsp0, LED_0 will be turned on. When the program leaves interrupt McBsp0, LED_0 will be turned off. Same scheme is applied for LED_1 and interrupt McBsp1. Now the time can be tested by connecting the LED to a scope. The results on the screen show that the time consumption is varied in time. This proved why the first method doesn't work. The only thing can be done here is to get an average value. For the order N=133, the average time consumption for interrupt McBsp0 is 47.8μs and for interrupt McBsp1 is 24.5μs. Vary the order of FIR filter, the results are almost the same. Note the result above is only valid when Fs_in=19.2KHz and Fs_out=38.8KHz. When the input and output sampling frequency are changed, the result for McBsp1 will also change accordingly.

7.3 THD

The total harmonic distortion (THD), of a signal is a measurement of the harmonic distortion and is defined as the ratio of the sum of the powers of all harmonic components to the power of the fundamental.

In most cases, the transfer function of a system is linear and time-invariant. When a signal passes through a non-linear device, additional content is added at the harmonics of the original frequencies. THD is a measurement of the extent of that distortion.

The measurement is most commonly the ratio of the sum of the powers of all harmonic frequencies above the fundamental frequency to the power of the fundamental:

$$\mathrm{THD} = \frac{\sum \text{All harmonic power}}{\text{fundamental power}} = \frac{P_2 + P_3 + P_4 + \cdots + P_n}{P_1}$$

Table 7.2 shows the theoretical simulation results of the THD values for a Linear Interpolator. The Matlab file "thd_LI.m", which is used to calculate the THD values, can be found in CD. The results tells that the THD value is a function of K_0. A larger K_0 leads to a higher THD. Looking at the data in Table 7.2 , the following relation can be found,

$$\mathrm{THD} \approx -12 \times (\log_2 \mathrm{K0} - 1) - 20.2$$

This means if K0 is increased by two times, the THD value will be 12dB better.

K0	THD
2	−20.2dB
4	-32.1dB
8	-44.7dB
16	-56.8dB
32	-68.9dB
64	-80.9dB
128	-93.0dB
256	-117.0dB

Table 7.2 THD linear interpolator

These THD values are calculated at the worst case. That means the bandwidth of the input signal is near the half of the input sampling frequency. In a practical flexible sampling rate-converter, the useful input bandwidth is always somewhat much smaller than half of the input sampling frequency. Therefore the corresponding THD should also somewhat better than listed in Table 7.2.

In this thesis, K_0 is set to be 8. From Table 7.2 it can be seen that the THD value should be smaller than -44dB. As said before, the THD value is also a function of input sampling frequency. Fig 7.3 shows THD measurement of a flexible interpolator.

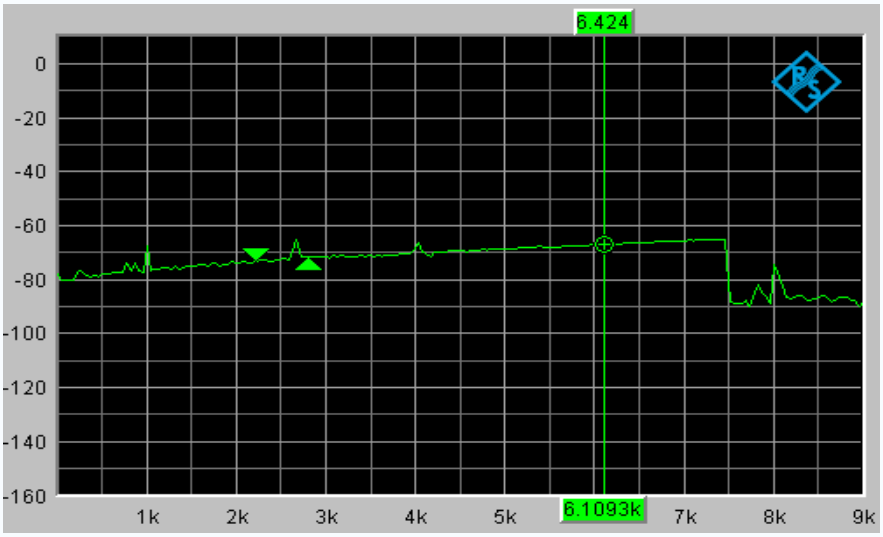


Fig 7.3 THD of flexible interpolator as function of input frequency

The input and output sampling frequency are 24KHz and 32.07KHz. From Fig 7.1 it can be seen that the transition band starts at 9KHz. So the measurement stops at 9KHz. Fig 7.3 shows that the THD value is a function of time. The best THD values are found in frequency range 4K and 8K. They are more than 60dB down. As explained before, it is better than the value shown in Table 7.2. For frequency range between 7.5KHz and 9KHz, the harmonics are in the stopband of the FIR filter. (Frequency response of the system can be found in Fig 7.1) So the THD values in that range are a little better.

The flexible decimator is implemented by using a transposed structure. Therefore it should have the same behavior as the flexible interpolator. Fig 7.4 shows the THD measurement results. The input and output sampling frequency are 24KHz and 13.59KHz. In most cases, the attenuation is under 70dB.

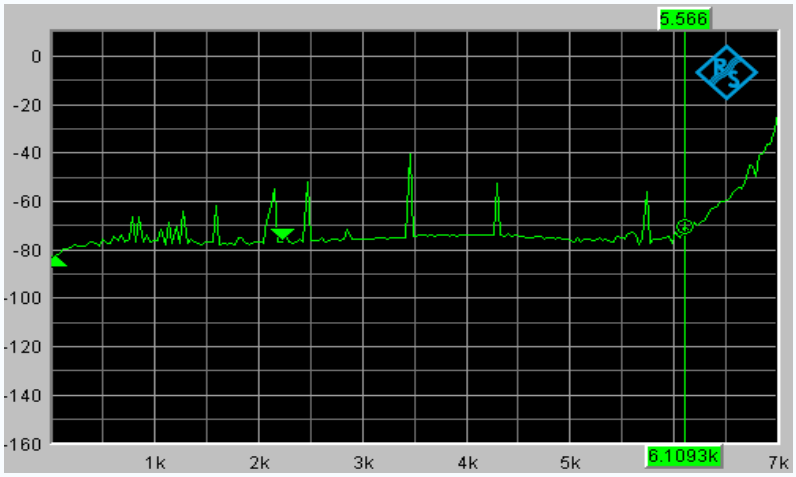


Fig 7.4 THD of flexible decimator as function of input frequency

8. Lagrange Interpolation

8.1 Lagrange Polynomial

In numerical analysis, a Lagrange polynomial, named after Joseph Louis Lagrange, is the interpolation polynomial for a given set of data points in the Lagrange form. It was first discovered by Edward Waring in 1779 and later rediscovered by Leonhard Euler in 1783. As there is only one interpolation polynomial for a given set of data points it is a bit misleading to call the polynomial the Lagrange interpolation polynomial. The more precise name is interpolation polynomial in the Lagrange form.

In this approach, polynomial approximation $\hat{x}(t)$ to x(t) is defined as

$$\hat{x}(t) = \sum_{k=-N_1}^{N_2} P_k(t)\mathrm{x[t+k]}, \qquad (8.1)$$

where $P_k(t)$ are the Lagrange polynomials given by

$$P_k(t) = \prod_{i=-N_1, i\neq k}^{N_2} \left(\frac{t-t_i}{t_k-t_i}\right), \quad -N_1 \le k \le N_2 \qquad (8.2)$$

Since

$$P_k(t_r) = \begin{cases} 1, & k = r \\ 0, & k \neq r \end{cases}, \quad -N_1 \le k \le N_2\ , \qquad (8.3)$$

it can be obtained from equations (8.1) to (8.3) that

$$\hat{x}(t_k) = x_a(t_k),\ -N_1 \le k \le N_2$$

From equation 8.1, the value of x(t) at an arbitrary value $t' = t_0 + \delta T$ (T is the input sample interval) is given by

$$\hat{x}(t') = x_a(t_0 + \delta T) = y[n] = \sum_{k=-N_1}^{N_2} P_k(\delta)\mathrm{x[n+k]}, \qquad (8.4)$$

where

$$P_k(\delta) = P_k(t_0 + \delta T) = \prod_{i=-N_1, i\neq k}^{N_2} \left(\frac{\delta - i}{k - i}\right),\ -N_1 \le k \le N_2 \quad (8.5)$$

8.2 SRC using Lagrange Interpolation

The mathematical background of Lagrange Interpolation is known from last section. Now the implementation of SRC using Lagrange interpolation algorithm will be given. At first an example will be given to make a clear idea how it works and then the performance will be discussed.

Consider the design of a SRC with an interpolation factor of 4/3. The order of the Lagrange polynomial is defined by $N_2 + N_1$. It has already been proved that the best approximation results can be obtained when the approximated point is in the middle of the given points. Therefore $N_2 = 1 + N_1$. When $N_2 = 1$, only two input samples will be used to determine the output samples and it is exactly a linear interpolator. For simplicity, a third-order polynomial approximation with $N_2 = 1$ and $N_1 = 2$ is used. In this case, the equation (8.4) is reduced to,

$$y[n] = P_{-2}(\delta)x[n-2] + P_{-1}(\delta)x[n-1] + P_0(\delta)x[n] + P_1(\delta)x[n+1] \quad (8.6)$$

Here the Lagrange polynomials $P_k(\delta)$ are given by

$$P_{-2}(\delta) = \frac{\delta+1}{-2+1} \times \frac{\delta+0}{-2+0} \times \frac{\delta-1}{-2-1} = \frac{1}{6}(-\delta^3 + \delta), \quad (8.7\text{ a})$$

$$P_{-1}(\delta) = \frac{\delta+2}{-1+2} \times \frac{\delta+0}{-1+0} \times \frac{\delta-1}{-1-1} = \frac{1}{2}(\delta^3 + \delta^2 - 2\delta), \quad (8.7\text{ b})$$

$$P_0(\delta) = \frac{\delta+2}{-0+2} \times \frac{\delta+1}{-0+1} \times \frac{\delta-1}{0-1} = -\frac{1}{2}(\delta^3 + 2\delta^2 - \delta - 2), \quad (8.7\text{ c})$$

$$P_1(\delta) = \frac{\delta+2}{1+2} \times \frac{\delta+1}{1+1} \times \frac{\delta-0}{1-0} = \frac{1}{6}(\delta^3 + 3\delta^2 + 2\delta), \quad (8.7\text{ d})$$

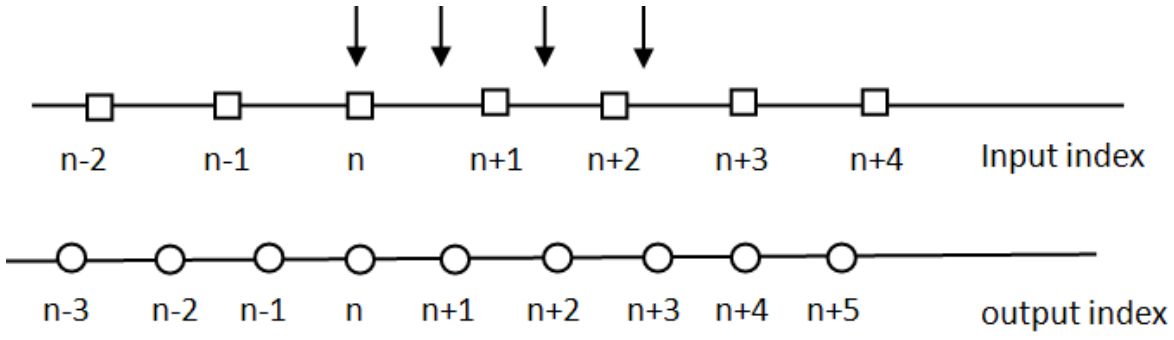


Fig 8.1 Input and Output sample location of an interpolator with factor 4/3

Fig 8.1 gives the information about the locations of input and output samples for an interpolator factor 4/3. The input sample locations are marked with a squire and the output sample locations are marked with a circle. The locations of output samples y[0], y[1] and y[2] in the input sample index is marked with an arrow.

From this figure it is clear that the value of δ for the calculation of y[n], to be labeled as δ_0, is 0. Substituting this value of δ in equations (8.7), the four $P_k(\delta)$ values can be obtained as

$$P_{-2}(\delta_0) = 0 \qquad P_{-1}(\delta_0) = 0$$
$$P_0(\delta_0) = 1 \qquad P_1(\delta_0) = 0$$

Next, the value of δ for the computation of y[n+1] is 3/4 and is labeled as δ_1. Using this value, the four $P_k(\delta)$ values can be obtained as

$$P_{-2}(\delta_1) = 0.0547 \qquad P_{-1}(\delta_1) = -0.2578$$
$$P_0(\delta_1) = 0.6016 \qquad P_1(\delta_1) = 0.6016$$

It is clear that y[n] and y[n+1] can be got by passing these values to equation (8.6).

The calculation of y[n+2] and y[n+3] can be obtained by the same way. But as has been explained before, in order to get an accurate approximation, the estimated sample should be in the center of the given samples. So input sample x[n-1],x[n],x[n+1] and x[n+2] are used for y[n+1]; input samples x[n],x[n+1],x[n+2] and x[n+2] are used for y[n+2].

From Fig 8.1it can be seen that the value of δ_2 for the computation of y[n+2] is 1/2. It is exactly the distance between the time index n+1 and the arrow on the right. The value of δ_3 for the computation of y[n+3] is 1/4. It is exactly the distance between the time index n+2 and the arrow on the right.

The calculation of the output samples can be changed into a matrix form

$$\begin{bmatrix} y[n] \\ y[n+1] \\ y[n+2] \\ y[n+3] \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x[n-2] \\ x[n-1] \\ x[n] \\ x[n+1] \\ x[n+2] \\ x[n+3] \end{bmatrix}$$

where $\boldsymbol{H}=\begin{bmatrix}\boldsymbol{H_0}\\ \boldsymbol{H_1}\\ \boldsymbol{H_2}\\ \boldsymbol{H_3}\end{bmatrix}$,

$$\boldsymbol{H_0} = \lceil P_{-2}(\delta_0) \quad P_{-1}(\delta_0) \quad P_0(\delta_0) \quad P_1(\delta_0) \quad 0 \quad 0 \quad \rceil$$

$$\boldsymbol{H_1} = \lceil P_{-2}(\delta_1) \quad P_{-1}(\delta_1) \quad P_0(\delta_1) \quad P_1(\delta_1) \quad 0 \quad 0 \quad \rceil$$

$$\boldsymbol{H_2} = \lceil 0 \quad P_{-2}(\delta_2) \quad P_{-1}(\delta_2) \quad P_0(\delta_1) \quad P_1(\delta_2) \quad 0 \quad \rceil$$

$$\boldsymbol{H_3} = \lceil 0 \quad 0 \quad P_{-2}(\delta_3) \quad P_{-1}(\delta_3) \quad P_0(\delta_3) \quad P_1(\delta_3) \rceil$$

The matrix *H* here is called the block matrix. $\boldsymbol{H_0}$, $\boldsymbol{H_1}$, $\boldsymbol{H_2}$ and $\boldsymbol{H_3}$ can be seen as 4 different FIR filters with order 5. $\boldsymbol{H}$ can be seen as a time-varying FIR filters with 6 coefficients. It will periodically choose the filter coefficients of one the four filters above. The period is equal to the output sampling interval.

Another realization of the example above can be achieved by substituting the Lagrange polynomials of equation (8.7) in equation (8.6) which yields,

$$y[n] = \delta^3\left(-\frac{1}{6}x[n-2] + \frac{1}{2}x[n-1] - \frac{1}{2}x[n] + \frac{1}{6}x[n+1]\right)$$

$$+\delta^2\left(-\tfrac{1}{2}x[n-1] - x[n] + \tfrac{1}{2}x[n+1]\right)$$

$$+\delta\left(\tfrac{1}{6}x[n-2] + x[n-1] + \tfrac{1}{2}x[n] + \tfrac{1}{3}x[n+1]\right)$$

$$+x[n]$$

Farrow Structure can be used in the realization of the above equations. The transfer functions of the three FIR digital filters are given by

$$H_0(z) = -\tfrac{1}{6}z^{-2} + \tfrac{1}{2}z^{-1} - \tfrac{1}{2} + \tfrac{1}{6}z$$

$$H_1(z) = \tfrac{1}{2}z^{-1} - 1 + \tfrac{1}{2}z$$

$$H_2(z) = \tfrac{1}{6}z^{-2} - z^{-1} + \tfrac{1}{2} + \tfrac{1}{3}z$$

The realization of such a structure is shown in Fig 8.2. In this realization, only the

value of the multiplier coefficient δ is periodically with the remaining digital filter structure kept unchanged. For general K, the number of filters will be K-1.

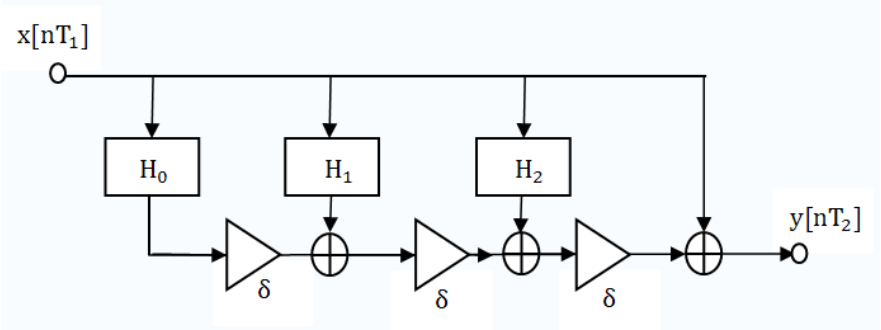


Fig 8.2 Farrow structure for Lagrange Interpolator

Fig 8.3 shows the amplitude response of such a structure for different K. The dashed line is the spectrum of the input signal with the bandwidth equal to $0.1Fs_{in.}$ With such bandwidth limitation, the attenuation for K>1 is around 60dB. But it can be seen from the figure, the attenuation of the image decrease strongly as the bandwidth of the input signal increases.

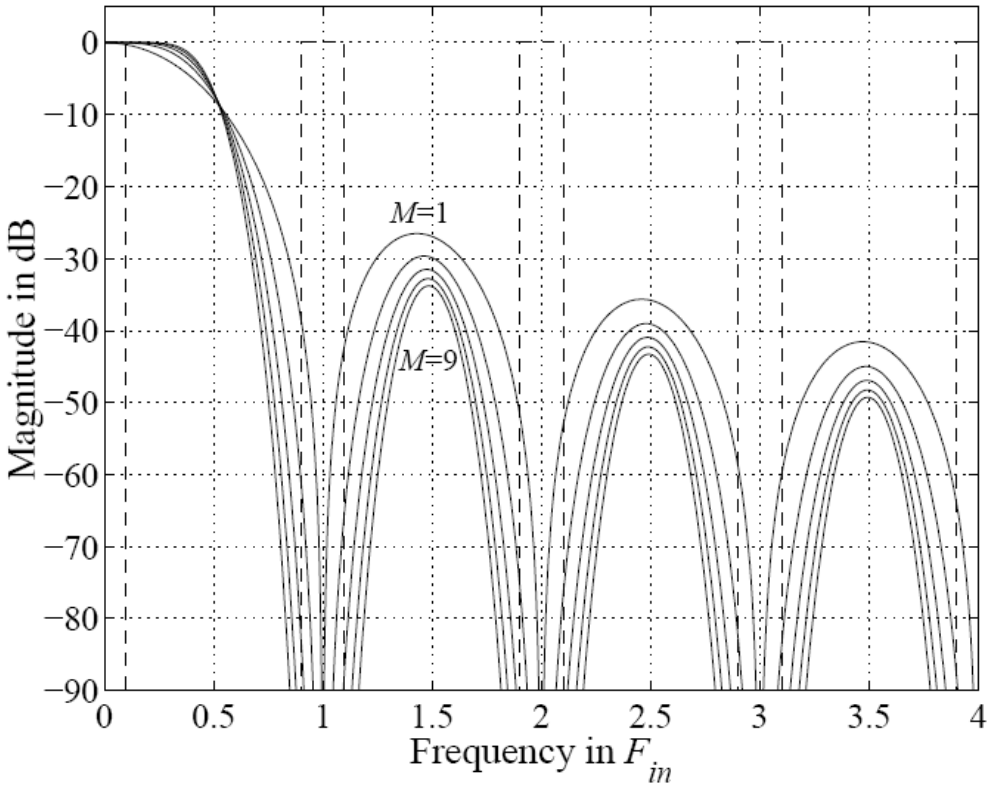


Fig 8.3 Amplitude responses of Lagrange Interpolator with order 1,3,5,7 and 9

8.3 Matlab simulation for Lagrange Interpolation

The purpose of this section is simulation the SRC using a Lagrange interpolation. Like what have been done for linear interpolator in section (5.1), a normal interpolator with interpolation factor K_0 is needed. This normal interpolator ensures a larger bandwidth usage. The value of K_0 is supposed to be 8 and the order of the Lagrange interpolator is supposed to be 4.

The first thing is to determine the Lagrange filter coefficients. The following code can generate the 3 filters shown in Fig 8.2. The filter coefficients are stored in array "lagrange_filter". Each row of this array represents one filter.

```
lag_orde=4;
lagrange_filter=zeros(lag_order,lag_order);
for j=-lag_order/2:lag_order/2-1
    b=[0,1];
    for i=-lag_order/2:lag_order/2-1
        if j~=i
          a=[1,-i];
          a=a/(j-i);
          b=conv(a,b);

        end
    end
    lagrange_filter(j+1+lag_order/2,:)=b(2:length(b));
end
lagrange_filter=lagrange_filter';
```

After constructing the Lagrange filter, the same scheme is applied as a Linear Interpolator. The switch rotates anti-clock with a step size X and a period $1/Fs_{out}$. The value of p and δ is needed to get the output samples. One thing has changed is that 4 values will be used instead 2. They are the output of the $(p-2)^{th}, (p-1)^{th}, p^{th}$ and $(p+1)^{th}$ filter. It is clear that if p<2, a new special case will be got. The previous output of polyphase filter $H_{8,6}(z)$ and $H_{8,6}(z)$ is needed. The final output can be obtained by passing these four values to the Lagrange filter. The value of δ here is still equal to non-integer part of the distance which the switch has rotated. The complete code is shown in CD with name lagrange.m.

Fig 8.4 shows the simulation result. The input signal is a sine wave with frequency equal to 0.1 Hz. The input sampling frequency is equal to 1 Hz. The ratio between output and input sampling frequency is 1.33. Fig 8.4 (a) is the input samples in time domain. Fig 8.4(b) shows the output signal in time domain. The time interval in Fig 8.4(a) is exactly 1.33 times larger than that in Fig 8.4(b).

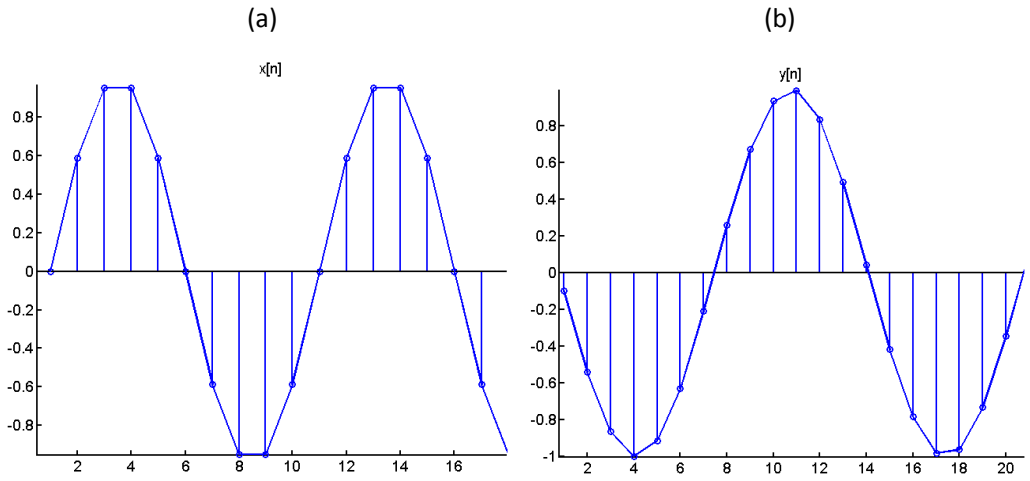


Fig 8.4 Lagrange interpolator in time domain

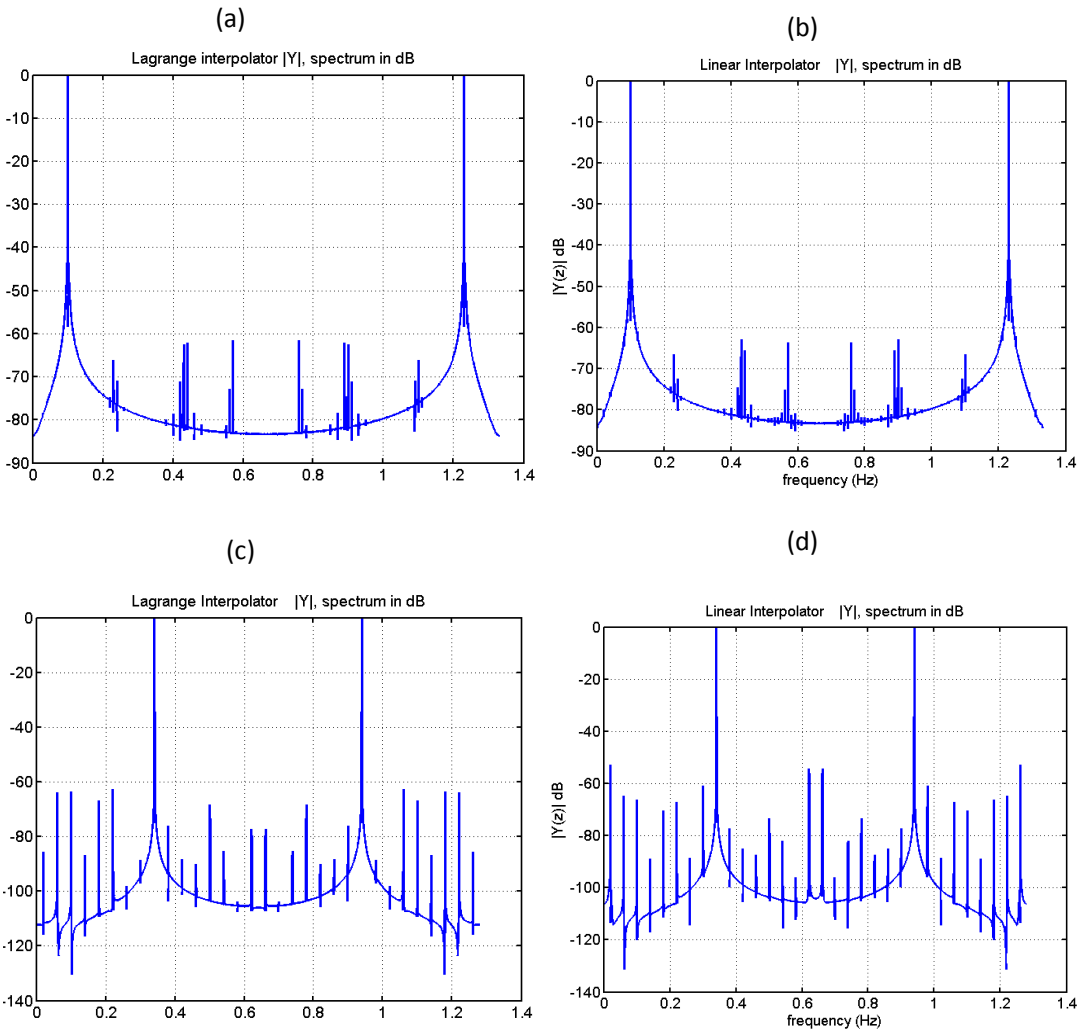


Fig 8.5 Comparison of Lagrange interpolator and Linear interpolator

Fig 8.5(a) shows the spectrum of the output signal. The attenuation of the images is more than 60 dB. Fig 8.5(b) shows output spectrum in the same condition with a linear interpolator. Compare the two spectrums, there is almost no difference. This is due to the fact that the bandwidth of the input signal is very small, only $0.1Fs_{in}$. Another simulation results shown in Fig 8.5(c) and 8.5(d) give the evidence that a third order Lagrange interpolator is better than the linear interpolator. The input sampling frequency is 1 Hz and the ratio is 1.28. The frequency of the input signal is changed to 0.34Hz. Fig 8.5(c) shows the spectrum of the output signal of a Lagrange interpolator. Fig 8.5(d) shows the spectrum of the output signal of a linear interpolator. The attenuation of the image for Lagrange interpolator is around 10dB better than that for a linear interpolator.

9. Conclusion

Now look back to the whole thesis and make a conclusion. This master thesis develops a system which can do arbitrary sampling rate conversion on a TI DSK C6713 board. The whole system can be seen as two separate parts. They are flexible interpolator and flexible decimator. As what has been stated in the title of this thesis, the linear interpolation theory is applied. An efficient realization of a flexible interpolator is first explained. The realization of a flexible decimator is obtained by using the transposed theory.

The goals of this thesis are stated in the first chapter. The first point is "flexible". From the testing results in chapter 7, the system can work in different ratio. This ratio can be an arbitrary. The second point is "self-tuning". This means the system should detect the ratio itself and can adapt itself when the ratio changes. This point is also proved in chapter 7. In the testing, the input sampling frequency is fixed and it is equal to 19.2 KHz. The output sampling frequency is controlled by a function generator. The output sampling frequency is varied between 8 KHz and 25 KHz. The system still works when the ratio changes. The quality of this system is measured by THD. The result meets the theoretical analyze. The last point is "Linear interpolator". The function of a Linear Interpolator (LI) is discussed in section 5.1. The realizations of the efficient circuits using such interpolation theory are also shown in Fig 5.11 and 5.19. It is obvious that the Linear Interpolator is an important component in both circuits. It is clear that all the goals of this thesis have been achieved.

At last, an alternative interpolation theory is given. It is so called Lagrange interpolation. From this chapter, it is known that the linear interpolation is a special case of Lagrange interpolation. It is the second-order Lagrange interpolation. It is no doubt that a high order Lagrange interpolation theory gives a better result. This is proved in the Matlab simulation results. Due to the time limitation, the real-time implementation of such a system has not been done.

Besides the implementation in this thesis, there are also some works that could be done in the future in order to improve the behaviour of the system. As is mentioned before, a high order interpolation method should be used instead of linear interpolation. The FIR filter used in this system is generated by a window function. This means the ripples in passband are not same. This causes an unstable system performance. An equiripple FIR filter such as "remez" filter should replace the existing filter.

Index

Reference

[1] Fliege: *Multirate Digital Signal Processing*
John Wiley & Sons (1994), ISBN: 0-471-93976-5

[2] P.P Vaidyanathan: *Multirate systems and filter banks*
Prentice-Hall, Englewood Cliffs, New Jersey(1993), ISBN: 0-13-605718-7

[3] A.W.M. van den Enden: *Digitale Signaalbewerking*
Delta Press (1987), ISBN: 90-6674-752-8

[4] Ad W.M. van den Enden:
Efficiency in multirate and complex digital signal processing
Delta Press (2001), ISBN: 90-6674-650-5

[5] Sanjit K.Mitra: *Digital Signal Processing—A computer-based Approach*
McGraw-Hill (2001), ISBN: 0-07-732105-9

[6] Rulph Chassaing:
Digital Signal processing and Applications with the C6713 and C6416 DSK
WILEY-INTERSCIENCE (2005), ISBN: 0-471-69007-4

[7] Rulph Chassaing: *DSP Applications Using C and the TMS320C6x DSK*
WILEY-INTERSCIENCE (2002), ISBN: 0-471-22112-0

[8] A.A.M.L. Bruekers, A.W.M. van den Enden:
Efficient structures for increasing and decreasing the sampling rate
Private Communication (1990)

[9] U.Zölzer: *Digital audio signal processing*
John Wiley (1997), ISBN: 0-471-97226-6

[10] T.A. Ramstad: *Sample-rate conversion by arbitrary ratios*
International Conference on Acoustics Speech and Signal processing
ICASSP (1982), page 101-104

[11] T.A. Ramstad:
Digital methods for conversion between arbitrary sampling frequencies
IEEE Transactions on Acoustics Speech and Signal Processing
June 1983, page 557-591

[12] J.G. Proakis, D.G. Manolakis,
Digital signal processing; principles, algorithms, and applications
Macmillan (1992) ISBN: 0-02-396815-X

[13] Lars Erup, Floyd M.Gardner
Interpolation in Digital Modems—Part 2: Implementation and Performance
IEEE Transactions on Communications, VOL.41, No.6, June 1993

[14] S.Cucchi, F.Desinan, G.Parladori and G.Sicuranza
DSP Implementation of Arbitary Sampling Frequency Conversion for High Quality Sound Application
IEEE CH2977-7/91/0000-3609 (1991)

[15] Hans Peter Kölzer
Courseware of course "Digital Signal Processing"
http://users.etech.haw-hamburg.de/users/Sauvagerd/DV_I6/dv_i6.html
[10.10.07]

[15] Ulrich Sauvagerd
Courseware of course "Digitale Signalverarbeitung auf DSPs"
http://users.etech.haw-hamburg.de/users/Sauvagerd/DSV_MES/dsv_mes.html
[10.07.07]

[16] http://en.wikipedia.org/wiki/Lagrange_polynomial [13.11.07]

[17] http://en.wikipedia.org/wiki/Total_harmonic_distortion [25.12.07]

[18] http://www.mathworks.com [01.09.07]

[19] http://ccrma.stanford.edu/~jos/pasp/Farrow_Structure_Variable_Delay.html

[20] Werner, Martin: *Digitale Signalverarbeitung mit Matlab*
Vieweg, F,(2006) ISBN: 978-3-8348-0043-5

[21] T. Saramäki, T. Ritoniemi
An efficient approach for conversion between arbitrary sampling frequencies
ISCAS 1996, page 285-288

[22] Nagahara, M.; Yamamoto, Y: *A new design for sample-rate converters*
Decision and Control, 2000. Proceedings of the 39th IEEE Conference on
Volume 5, Page(s):4296 - 4301 vol.5

[23] Chambers, J.P.: *Digital linear interpolator*
2007 ISSN: 0013-5194 Pages: 256 - 257

Appendix A

List of attachments In CD

`sim_interpolator.m`

The simulation of the structure shown in Fig 5.7

`sim_decimator.m`

The simulation of the structure shown in Fig 5.14

`flexible_interpolator_li.m`

The simulation of a flexible interpolator using linear interpolator

`flexible_decimator_li.m`

The simulation of a flexible decimator using linear interpolator

thd_LI.m

The measurement of THD for Linear Interpolator with different values of K0.

`lagrange.m`

The simulation of flexible interpolator using Lagrange interpolator

Project "flexible interpolator"

The real-time implementation of a flexible interpolator using linear interpolator

Project "flexible decimator"

The real-time implementation of a flexible decimator using linear interpolator

Project "flexible SRC"

The real-time implementation of a flexible SRC using linear interpolator