Examensarbete

FPGA Implementation of Flexible Interpolators and Decimators

Examensarbete utfört i Elektroniksystem vid Tekniska högskolan vid Linköpings universitet av

Venkatavikram Dabbugottu

LiTH-ISY-EX--13/4654--SE
Linköping 2013
FPGA Implementation of Flexible Interpolators and Decimators

Examensarbete utfört i Elektroniksystem vid Tekniska högskolan i Linköping av

Venkatavikram Dabbagottu

LiTH-ISY-EX--13/4654--SE

Handledare: Amir Eghbali
            ISY, Linköpings universitet

Examinator: Håkan Johansson
            ISY, Linköpings universitet

Linköping, 21 February, 2013
The aim of this thesis is to implement flexible interpolators and decimators on Field Programmable Gate Array (FPGA). Interpolators and decimators of different wordlengths (WL) are implemented in VHDL. The Farrow structure is used for the realization of the polyphase components of the interpolation/decimation filters. A fixed set of subfilters and adjustable fractional-delay multiplier values of the Farrow structure give different linear-phase finite-length impulse response (FIR) lowpass filters. An FIR filter is designed in such a way that it can be implemented for different wordlengths (8-bit, 12-bit, 16-bit). Fixed-point representation is used for representing the fractional-delay multiplier values in the Farrow structure. To perform the fixed-point operations in VHDL, a package called fixed point package [1] is used.

A 8-bit, 12-bit, and 16-bit interpolator are implemented and their performances are verified. The designs are compiled in Quartus-II CAD tool for timing analysis and for logical registers usage. The designs are synthesised by selecting Cyclone IV GX family and EP4X30CF23C6 device. The wordlength issues while implementing the interpolators and decimators are discussed. Truncation of bits is required in order to reduce the output wordlength of the interpolator and decimator.
Abstract

The aim of this thesis is to implement flexible interpolators and decimators on Field Programmable Gate Array (FPGA). Interpolators and decimators of different wordlengths (WL) are implemented in VHDL. The Farrow structure is used for the realization of the polyphase components of the interpolation/decimation filters. A fixed set of subfilters and adjustable fractional-delay multiplier values of the Farrow structure give different linear-phase finite-length impulse response (FIR) lowpass filters. An FIR filter is designed in such a way that it can be implemented for different wordlengths (8-bit, 12-bit, 16-bit). Fixed-point representation is used for representing the fractional-delay multiplier values in the Farrow structure. To perform the fixed-point operations in VHDL, a package called fixed point package [1] is used.

A 8-bit, 12-bit, and 16-bit interpolator are implemented and their performances are verified. The designs are compiled in Quartus-II CAD tool for timing analysis and for logical registers usage. The designs are synthesized by selecting Cyclone IV GX family and EP4X30CF23C6 device. The wordlength issues while implementing the interpolators and decimators are discussed. Truncation of bits is required in order to reduce the output wordlength of the interpolator and decimator.
Acknowledgments

I would like to thank my examiner Prof. Håkan Johansson and my supervisor Dr. Amir Eghbali for giving the opportunity to do my master thesis and for their help and guidance throughout the thesis period. I would also like to thank Dr. Kent Palmkvist for his valuable suggestions and help.
### 4.4 Wordlength Issues in Interpolator

4.5 VHDL Implementation of the Decimator

- 4.5.1 VHDL Implementation of the Delay Block $d_{in}$
- 4.5.2 VHDL Implementation of Subfilters

### 5 Testbench and Simulation Results

- 5.1 Introduction
- 5.2 Design Flow for Testing Interpolator
  - 5.2.1 Generating a Sine Wave
  - 5.2.2 VHDL Testbench
  - 5.2.3 Simulation results
- 5.3 Testbench for the Decimator
- 5.4 Synthesis and Timing Analysis
  - 5.4.1 Synthesis and Time Quest Analyzer Reports

### 6 Conclusion and Future Work

- 6.1 Conclusion
- 6.2 Future Work

### A Appendix

- A.1 VHDL Code
  - A.1.1 VHDL Code for Subfilter
  - A.1.2 VHDL Code for Delay Block of Decimator
  - A.1.3 VHDL Code for Farrow Structure of Decimator
  - A.1.4 User Package

### Bibliography
List of Figures

2.1 Interpolation by a factor of $L$ .................................................. 6
2.2 Original sequence and the upsampled sequence by a factor of 3 ... 7
2.3 Spectra of the original, intermediate, and output sequences ... 8
2.4 Decimation by a factor of $M$ .................................................. 8
2.5 Spectra of the filtered $x_1(m)$ and the decimated sequence ... 9
2.6 Sampling rate converter for conversion by a rational factor $L/M$ .. 9
2.7 Deriving a polyphase interpolator .......................................... 11
2.8 Deriving a Polyphase Decimator ............................................. 12
2.9 Noble identities ................................................................. 13
2.10 Polyphase interpolator ....................................................... 13
2.11 Polyphase decimator ......................................................... 13
2.12 Interpolator realization using time-varying coefficients, $m = nM + i$
and $k_i$ is the largest integer smaller than or equal to $(N - i)/M$ .. 14

3.1 Farrow structure ............................................................... 16
3.2 Modified Farrow structure .................................................. 17
3.3 Realization of the polyphase components $H_m(z)$ and $H_{M-m}(z)$ in
the interpolator structure ..................................................... 18
3.4 Flexible interpolator structure ............................................. 19
3.5 Flexible decimator structure ................................................ 19

4.1 Flowchart for implementing the interpolator ......................... 25
4.2 RTL representation of FIR filter ......................................... 26
4.3 The $G_k(z)$ block with six subfilters .................................... 27
4.4 Block diagram for VHDL implementation of the interpolator .... 29
4.5 Subfilters and delay multiplier chain block ......................... 31
4.6 Delay multiplier chain block with truncation ....................... 32
4.7 Block diagram for VHDL implementation of the decimator .... 34
4.8 A 16-bit delay multiplier chain block with truncation’s .......... 35
4.9 Block diagram of Decimator .............................................. 36

5.1 Design flow for testing interpolator .................................... 40
5.2 output of interpolator in Modelsim .................................... 41
5.3 Testbench for the Interpolator ........................................... 42
5.4 sinusoidal signal and the interpolated sinusoidal signal ($M=3$) .. 43
5.5 Frequency response for the interpolator output ..................... 44
5.6 Frequency response for the interpolator output ..................... 44
5.7 Frequency response for the interpolator output ..................... 45
5.8 Output of the decimator in Modelsim .................................. 46
5.9 Frequency response for the decimator output ....................... 47
5.10 Frequency response for the decimator output ...................... 47
5.11 Frequency response for the decimator output ...................... 48
List of Tables

3.1 Impulse response of The filter $H_0(z)$ ...................... 20
3.2 Impulse response of subfilters $G_0(z), G_1(z), G_2(z)$ .......... 21
3.3 Impulse response of subfilters $G_3(z), G_4(z), G_5(z)$ .......... 21

4.1 Table of specification .............................................. 24
4.2 Table for operation sizing rules .................................. 24
4.3 Fractional-delay multiplier values for $M = 8$ and $M = 5$ .... 30

5.1 Table for resource utilization and performance .................. 45

0.1 List of Abbreviations

SRC-Sampling ratio conversion
WL-Word length
FPGA- Field programming gate array
FIR- Finite impulse response
VHDL- VHSIC Hardware description language
ASIC - Application specific integrated circuits
CPLD - Complex programmable logic device
DSP - Digital signal processing
CD - Compact disc
DAT - Digital audio tape
Chapter 1

Introduction

1.1 Introduction

Nowadays, the modern digital systems are more complex. They consist of several DSP (Digital Signal Processing) processors that operate at different sampling frequencies. For example, in smart mobile phones we can find separate DSP processors for video, photo-camera, music, video-voice recording and communication [2]. Furthermore, there are three common sample rates utilized in the audio community. They are 32 KHz stream rate for broadcast industry, 44.1 KHz stream rate for compact disc and 48 KHz for digital audio tape (DAT) [3]. A common sample rate should be employed for these signals to combine/mix the three environments digitally. The stream at lower sample rate is interpolated to increase the sample rate in-order to match the higher sample rate signal.

Interpolators can be found in mixed-signal processing systems, digital receivers and in sigma-delta modulators [2]. In the sigma-delta modulators, the interpolation operation, i.e. oversampling, moves the signal frequency and quantization noise apart from each other. Decimators are used to reduce the higher sample rate to lower sample rate.

1.2 Background

For high interpolation factors most of the sample values are zero, so there is an unnecessary additional computation. The computational workload can be reduced by using the polyphase interpolators and decimators. The polyphase interpolation and decimation are efficient when compared with straightforward realization of interpolation and decimation. The disadvantages of the polyphase interpolator (decimator) structure are that they require new filters if the sample rate conversion (SRC) ratio changes. This limits the flexibility of interpolation (decimation) with different SRC ratios. More memory is required to store the coefficient if the interpolation factor is high. These problems can be solved elegantly by the linear-phase FIR interpolation (decimation) utilizing the Farrow structure. This structure is
flexible to the conversion factors, and also for an arbitrary set of integer factors, including prime numbers.

1.3 Purpose of the Thesis

In this thesis work, an interpolator and decimator FPGA prototype is designed. For many applications, FPGA implementation brings advantages that include: low cost, higher precision processing, design flexibility, and low power. The main objective of this thesis work is to implement the flexible interpolators and decimators on FPGA. The implementation is done in VHDL. In this thesis the interpolation/decimation factor can be varied from 2 to 20.

1.4 Thesis Outline

This thesis report has five chapters. Chapter 2 provides the basics of interpolation and decimation, SRC and polyphase interpolation and decimation. It also includes the basics of SRC by rational factors.

Chapter 3 gives a brief introduction to the Farrow structure, interpolation and decimation filters, SCR utilizing the Farrow structure and linear-phase FIR interpolation and decimation utilizing the Farrow structure.

Chapter 4 is about the VHDL implementation of the interpolators and decimators. This chapter discusses how the FIR filter and subfilters are implemented. The wordlength issues and the truncation in the design are mentioned in this chapter.

Chapter 5 gives the testbench for the interpolator and decimator. The frequency response of the interpolators and decimators with different interpolation and decimation factors are plotted.

Chapter 6 gives the conclusion and future work.
Chapter 2

Interpolation and Decimation

2.1 Introduction

There are many systems where the sampling rate of a signal needs to be converted into an equivalent signal with a different sampling rate [4]. Multiple sampling frequencies are used in many applications, in order to reduce and simplify the computational workload [5]. For example, in digital audio, three different sampling rates are used, 32 KHz in broadcasting, 44.1 KHz in digital compact disc, and 48 KHz in DAT.

This chapter begins with the basics of interpolation and decimation, and a brief overview of SRC. The concepts of polyphase interpolation and decimation structures are discussed.

2.2 Interpolation

Interpolation is a process of increasing the sampling rate and the system which performs it is called interpolator [5]. The aim of the interpolation is to get a new sequence with higher sampling rate without losing the information. Interpolation is a two-stage process, first the input signal is upsampled and then the upsampled signal is filtered. In the first stage, \( L-1 \) zero-valued samples are inserted in between consecutive samples of the original sequence, where \( L \) is the interpolation factor. Figure 2.1 shows the block diagram of the interpolator. Figure 2.2 illustrates the original sequence \( x(n) \) and the upsampled sequence \( x_1(m) \). The new signal, generated after the upsampler is [5]

\[
x_1(m) = \begin{cases} 
x(n \frac{m}{L}) & \text{for } m = 0, \pm L, \pm 2L \ldots \\
0 & \text{otherwise.}
\end{cases}
\]  

(2.1)
The Fourier transform of \( x_1(m) \) can be expressed as [5]

\[
    x_1(e^{j\omega T_1}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j n\omega T} = X(e^{jL\omega T_1}).
\]

(2.2)

In the z-domain, we have [5]

\[
    X_1(z) = X(z^L).
\]

(2.3)

![Figure 2.1. Interpolation by a factor of L.](image)

The sampling period of the new sequence \( x_1(m) \) is \( T_1 = \frac{T}{L} \). As the sampling rate of the new sequence is increased by \( L \), the spectrum of the sequence \( x_1(m) \) contains not only the original signal but also repeated images of the original signal. A lowpass filter is used to remove the images and the stopband edge must be at \( \omega_s T = \frac{\pi}{L} \) [5]. Figure 2.3 shows the spectra of the original sequence, the upsampled sequence \( x_1(m) \), and the signal after filtering \( y(m) \).

### 2.3 Decimation

Decimation is the process of reducing the sampling rate and the system which performs this task is called decimator [5]. Decimation of a signal with a factor \( M \) is a two-stage process. The first stage contains an anti-aliasing filter and in the next stage, a downsampler. By extracting every \( M \)th value of a signal, the sampling rate of the signal is reduced by a factor of \( M \). This process is done by a downsampler. Figure 2.4 shows the block diagram of the decimator. The output \( y(n) \) is [5]

\[
    y(n) = x_1(nM).
\]

(2.4)

The Fourier transform of \( y(n) \) can be expressed as [5]

\[
    Y(e^{j\omega T}) = \frac{1}{M} \sum_{k=0}^{M-1} X_1(e^{j(\omega T_1-2\pi k/M)}).
\]

(2.5)

The sampling period of \( y(n) \) is \( T = MT_1 \) with \( T_1 \) being the sampling period for \( x_1(m) \). Figure 2.5 shows the intermediate sequence and the output sequence of the signal. The output sequence consists of a sum of shifted, expanded replicas of the original signal. The stopband attenuation of the filter should be in such a way that the signal \( x_1(m) \) is not attenuated and high enough to prevent the aliased components. The signal \( x_1(m) \) must be band limited to \( \frac{\pi}{M} \) [5].
2.4 Sampling Rate Conversion by a Rational Factor

Sections 2.2 and 2.3 discussed the increasing or decreasing the sampling rate by an integer factor. This section gives a brief introduction on increasing the sampling rate by a rational factor.

The sampling rate can be increased by a factor of \( \frac{L}{M} \) with \( L > M \), where \( L \) and \( M \) are integers [5]. This sampling rate is achieved first by interpolating with a factor of \( L \) and then decimating by a factor of \( M \), as shown in Fig. 2.6. The stopband edge of the filter \( H(z) \) should be at \( \omega_s T = \min(\pi/L, \pi/M) \).

--- Example 2.1: SRC by a rational number ---

Consider a signal with sampling frequency of 500 Hz where the new sampling rate has to be increased to 1600 Hz. First interpolating it with a factor 16, gives \( 16 \times 500 = 8000 \) and then decimating it by a factor of 5 (8000/5=1600 Hz), results in an increase of the sampling frequency by a factor of 3.2 [6].

In Fig. 2.6, the upsampler and the filter perform the same task as discussed in Section 2.2. After the sampling rate is increased, we discard the samples we do not need and keep the ones required. In this process, many of the samples computed via interpolation are discarded, which is an inefficient process. To implement fractional
Figure 2.3. Spectra of the original, intermediate, and output sequences.

Figure 2.4. Decimation by a factor of $M$. 
2.4 Sampling Rate Conversion by a Rational Factor

Figure 2.5. Spectra of the filtered $x_1(m)$ and the decimated sequence.

Figure 2.6. Sampling rate converter for conversion by a rational factor $L/M$. 
interpolation and decimation effectively, there are many advanced techniques [5]. One efficient way is the polyphase interpolation and decimation.

2.5 Polyphase Interpolation and Decimation Structures

2.5.1 Polyphase Representation

This is a three-step process. First, $M$ signals are formed from $h(n)$, where $h(n)$ is a sum of $M$ partial signals [5]

$$h_i(n) = h(nM + i), \quad i = 0, 1, 2, ..., M - 1. \quad (2.6)$$

Second, the $h_i(n)$ are upsampled by $M$, i.e.,

$$h_i^{(M)}(n) = \begin{cases} h_i\left(\frac{n}{M}\right) & \text{for } n = 0, \pm L, \pm 2L, \ldots \\ 0 & \text{otherwise} \end{cases}. \quad (2.7)$$

Third, summing all the shifted versions of $h_i^{(M)}(n)$ as

$$h(n) = \sum_{i=0}^{M-1} h_i^{(M)}(n - i). \quad (2.8)$$

The polyphase representation of a signal $h(n)$ in the $z$-domain is [5]

$$H(z) = \sum_{i=0}^{M-1} z^{-i}H_i(z^M), \quad (2.9)$$

where

$$H_i(z) = \sum_{n=0}^{\infty} h_i(n)z^{-n} = \sum_{n=0}^{\infty} h_i(nM + i)z^{-n}. \quad (2.10)$$

2.5.2 Noble Identities

The noble identities allows to move the order of upsampling/downsampling and filtering [7]. Figure 2.9 shows the representation of the noble identities. Combination of the polyphase representation and noble identities gives the efficient realization of multirate structures. For example with $M=2$, the polyphase representation of $H(z)$ is derived from Eq. (2.9) as

$$H(z) = H_0(z^2) + z^{-1}H_1(z^2). \quad (2.11)$$

From the above equation, $H(z)$ is an addition of two subfilters and a delay shown in Fig. 2.7. Using the noble identities, the upsampler is moved forward as shown in Fig. 2.7. In reality the polyphase interpolator is realized as shown in Fig. 2.10, where a device called commutator is used instead of summation at the output.
Figure 2.7. (a) Interpolation by two, (b) obtained structure utilizing polyphase representation from Eq. 2.11, (c) restructured of (b), (d) polyphase interpolator using noble identities and (e) polyphase interpolator with commutator.
Interpolation and Decimation

At each time instant, the inputs of the summation has only one non-zero sample, so the output can be taken alternatively, beginning from the upper branch. A decimator is derived from the interpolator by reversing the signal-flow graph and replacing the upsampler by a downsampler [8] which is shown in Fig. 2.8. The polyphase interpolator and decimator are shown in Figs. 2.10 and 2.11. In these structures, all the subfilters operate at the lower sampling rate which leads to effective interpolation and decimation.

2.6 Converters with Time-Varying Coefficients

In the polyphase interpolation/decimation we have $M$ parallel subfilters. If all the subfilters are FIR filters with direct form structures, then it is possible to share the delay elements between the subfilters. The interpolator/decimator can be realised by changing the filter coefficients periodically, that is with time-varying coefficients. The interpolator can be realized by a filter with periodically time-varying coefficients [5], as illustrated in Fig 2.12.

The input signal $x(n)$ is fed into a chain of $N_i$ cascaded delay elements, where $N_i$ denotes the order of the filter. The content in the delay elements are multiplied...
2.6 Converters with Time-Varying Coefficients

Figure 2.9. Noble identities.

Figure 2.10. Polyphase interpolator.

Figure 2.11. Polyphase decimator.
by the impulse response values of $h_i(n)$, $h_i(n)=h(nM + i)$, $i = 0, 1, ..., M - 1$ [5].

The output signal $y(m)$ at the time instance $nM + i$ is [5]

$$y(nM + i) = \sum_{k=0}^{N_i} h_i(k)x(n - k)$$

(2.12)

**Figure 2.12.** Interpolator realization using time-varying coefficients, $m = nM + i$ and $k_i$ is the largest integer smaller than or equal to $(N - i)/M$. 
Chapter 3

Flexible Interpolators and Decimators

3.1 Introduction

This chapter gives a brief introduction to the Farrow structure, interpolation/decimation filters, SRC utilizing the Farrow structure, and linear-phase FIR interpolation and decimation utilizing the Farrow structure.

The Farrow structure based interpolators and decimators have been used for converting sampling rate from 44.1 KHz to 48 KHz in digital audio [9]. The Farrow structure based interpolators are used in applications like symbol timing recovery in QAM demodulation receiver [9], echo cancellation in digital modems and sampling rate equalization in WIMAX and GSM communication systems [9].

3.2 Farrow Structure

The efficient way of implementing the polyphase interpolation/decimation filters are by using the Farrow structure. The Farrow structure is composed of fixed FIR filters. The output of each filter is obtained after a delay of a single unit from the previous filter output [9]. Figure 3.1 shows the Farrow structure. The transfer function is [8]

\[ H(z) = \sum_{k=0}^{L} G_k(z) d_m^k, \quad m = 1, 2, \ldots, M - 1 \]  

(3.1)

where \( G_0(z), G_1(z), \ldots, G_K(z) \) are linear-phase FIR filters and \( d_m \) are the fractional-delay coefficients. The filters \( G_k(z) \) are designed to obtain a fractional-delay filter \( H(z) = z^{-d_m} \) [7]. If the fractional-delay coefficient value is the same for all inputs, the Farrow structure gives the delayed version of inputs, with a delay \( d_m \). The Farrow structure used in this thesis is shown in Fig. 3.1.
3.3 **Farrow Structure for SRC**

In the polyphase interpolation and decimation structures, if the SRC ratio changes, new filters are needed which limits the flexibility of interpolating and decimating with different SRC ratios. The polyphase interpolation and decimation structures lead to a large number of coefficients and more memory is required if the interpolation/decimation factor is high. This can be solved elegantly by utilizing the Farrow structure [10].

To perform any integer SRC, it is required to modify the fractional-delay values required by the Farrow structure and, it is possible to use one set of subfilters [11]. A delayed version of the input signal is generated by the Farrow structure, when \( d_m \) is constant for all input samples [11]. The Farrow structure performs SRC, if the \( d_m \) value changes for every input signal.

For example re-sampling a 8 KHz signal to 44.1 KHz, requires interpolation by 441 and decimation by 80. For these conversions, the polyphase interpolation and decimation structures are not preferable. Hence, SRC utilizing Farrow structure are preferred [10]. For more literature on SRC utilizing Farrow structure we refer to [8], [7], [12].

The interpolation/decimation filters in [8] uses the modified Farrow structure shown in Fig. 3.2. But in this thesis the Farrow structure shown in Fig. 3.1 is used. There is no different in these two structures, the structure shown in Fig. 3.1 is used because, it is easy to implement in VHDL.

3.4 **Linear-Phase FIR Interpolation and Decimation Utilizing the Farrow Structure**

This section gives the introduction of the interpolation and decimation filter transfer function proposed in [8] and a brief description on linear-phase FIR interpolator
3.4 Linear-Phase FIR Interpolation and Decimation Utilizing the Farrow Structure

Figure 3.2. Modified Farrow structure

and decimator. For the design of the interpolation and decimation filters we refer to [8].

The filters proposed in [8] are based on the properties of interpolation and decimation filters polyphase components. The transfer function of the polyphase components is written as [8]

\[ H(z) = \sum_{m=0}^{M-1} z^{-m} H_m(z^M) \] (3.2)

where \( H_m(z) \) are polyphase components and \( H(z) \) is a linear-phase interpolation or decimation lowpass filter. The filter \( H(z) \) is to approximate \( z^{-N/2} \) in the ideal passband region \( \omega T \in [-\pi/M, \pi/M] \) and zero in the ideal stopband \( \omega T \in [-\pi, -\pi/M] \cup [\pi/M, \pi] \) [8]. This approximation is achieved when the delay of each \( z^{-m} H_m(z^M) \) approximates \( z^{-N/2} \) in the passband region [8]. That is,

\[ z^{-m} H_m(z^M) \approx z^{-N/2} \]

\[ H_m(z^M) \approx z^{-(N/2)-m} \]

\[ H_m(z) \approx z^{-(N/2)-m}/M. \] (3.3)

From Eq. 3.3, it follows that \( H_m(z) \) is an allpass filter with a fractional delay of \( ((N/2) - m)/M \). Based on this approximation, the polyphase components are selected [8]. First \( H_0(z) \), is an \( N_0 \)th-order, Type-I linear-phase FIR filter of even order. The selection of \( H_0(z) \) ensures that \( H_0(z^M) \) can approximate \( z^{-N/2} \), provided \( N \) satisfies [8]

\[ N = N_0 M. \] (3.4)

The polyphase components \( H_m(z) \) are realized using the Farrow structure shown in Fig. 3.1 and the transfer functions \( H_m(z) \) are expressed as [8]

\[ H_m(z) = \sum_{k=0}^{L} d_m^k G_k(z), \quad m = 1, 2, \ldots, M - 1 \] (3.5)
where $G_k(z)$ are odd-order subfilters (linear-phase FIR filters) of order $N_1$ and $d_m$ represents fractional-delay coefficients which are anti-symmetric [8]. The impulse response of subfilters, i.e., $g_k(n) = g_k(N_1 - n)$ (k is even) for symmetric and $g_k(n) = -g_k(N_1 - n)$ (k is odd) for anti-symmetric. The subfilters $G_k(z)$ are here of odd order [8].

The filter $H_m(z)$ approximates an all-pass filter with a fractional delay of $N_1/2 + d_m$, provided $N$ satisfies [8]

$$N = (N_1 + 1)M. \quad (3.6)$$

The delay $((N/2) - m)/M$ must be equal to the delay $N_1/2 + d_m$ leading to

$$d_m = -\frac{m}{M} + \frac{1}{2} \quad (3.7)$$

where $d_m$ possess anti-symmetric according to $d_m = -d_{M-m}$.

### 3.4.1 Interpolator and Decimator structures

In the interpolation and decimation filters, the polyphase components $H_m(z)$ as in Eq.( 3.5) have $d_m$ exhibiting anti-symmetric form, so the polyphase components $H_m(z)$ and $H_{M-m}(z)$ can be written as [8]

$$H_m(z) = F_{m1}(z) + F_{m2}(z) \quad (3.8)$$

$$H_{M-m}(z) = F_{m1}(z) - F_{m2}(z) \quad (3.9)$$

where

$$F_{m1}(z) = \sum_{k=0}^{\lfloor L/2 \rfloor} d_m^{2k} G_{2k}(z) \quad (3.10)$$
3.4 Linear-Phase FIR Interpolation and Decimation Utilizing the Farrow Structure

\[ F_m(z) = \sum_{k=1}^{[L/2]} d_{2k-1} G_{2k-1}(z) \]  

where \([L/2]\) is the largest integer smaller than or equal to \(L/2\) and for \([(L+1)/2]\) it is largest integer smaller than or equal to \((L+1)/2\). The polyphase components \(H_m(z)\) and \(H_{M-m}(z)\) can be realized simultaneously using the same \(d_m\) [8]. The even polyphase components are realized from \(F_{m1}(z)\) and odd polyphase components by \(F_{m2}(z)\) as shown in Fig. 3.3. When \(M\) is even, \(d_{M/2} = 0\) resulting in \(H_{M/2}(z) = G_0(z)\). Also note that for \(M = 2\), the filters reduce to Type-I linear-phase FIR filters.

The interpolator can be realized as shown in Fig. 3.4. The wide arrow between \(G_k(z)\) and \(d^k_m\) indicates \(L + 1\) parallel signals. The interpolator structure has three main blocks, they are \(H_0(z)\), \(G_k(z)\) and \(d^k_m\), where \(H_0(z)\) is a linear-phase FIR filter of even order and \(G_k(z)\) are linear-phase FIR subfilters of odd order. The block \(d^k_m\) contains all the fractional-delay multipliers and adders to form the polyphase component outputs. The decimator structure is a transpose of interpolator structure and is shown in Fig. 3.5.
Table 3.1. Impulse response of the filter $H_0(z)$.

| $h(0)$ | $h(1)$ | $h(2)$ | $h(3)$ | $h(4)$ | $h(5)$ | $h(6)$ | $h(7)$ | $h(8)$ | $h(9)$ | $h(10)$ | $h(11)$ | $h(12)$ | $h(13)$ | $h(14)$ | $h(15)$ | $h(16)$ | $h(17)$ | $h(18)$ |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| $-0.00194202018921064$ | $0.00454034034339180$ | $-0.00852167281398188$ | $0.0264294829651506$ | $-0.0326345311590520$ | $0.0377645426334144$ | $-0.0411489015845974$ | $1.04233196249064$ | $-0.0411489015845974$ | $0.0377645426334144$ | $-0.0326345311590520$ | $0.0264294829651506$ | $-0.0411489015845974$ | $0.0377645426334144$ | $-0.0411489015845974$ | $0.0377645426334144$ | $-0.0411489015845974$ | $0.0377645426334144$ | $-0.0411489015845974$ | $0.0377645426334144$ |

By adding additional $d_k$ converter to the same filters ($H_0(z), G_k(z)$) several interpolators can be implemented simultaneously [8].

### 3.5 Filter Design

The specification of the overall lowpass filter is

$$1 - \delta_c \leq H_R(\omega T) \leq 1 + \delta_c, \quad \omega T \in [0, \omega_c T]$$

$$-\delta_s \leq H_R(\omega T) \leq \delta_s, \quad \omega T \in [\omega_s T, \pi]$$

The frequency response of the overall filter can be written as

$$H(e^{j\omega T}) = e^{-jN\omega T/2}H_R(\omega T)$$

(3.12)

where

$$H_R(\omega T) = h\left(\frac{N}{2}\right) + 2\sum_{n=1}^{N/2} h\left(\frac{N}{2} - n\right)\cos(\omega T n).$$

(3.13)

$H_R(\omega T)$ is the real zero-phase frequency response of $H(z)$. The passband edge is at $\omega_c T = \pi/M - \delta$, $\delta > 0$, and the stopband edge is assumed to be at $\pi/M$ or $\pi/M + \delta$. Refer to the paper [8] for optimization problem. The values for $L$ and $N_1$ are selected in a proper way. The value of $L$ is selected from the outline of [13]. The values of $L$ and $N_1$ are more or less independent of $M$. In this thesis the specifications of the filter are $\omega_c T = 0.5\pi/M$, $\omega_s T = \pi/M$, $\delta_c = 0.01$, and $\delta_s = 0.001$. We have $L = 5$ and $N_1 = 17$. The impulse response of filter $H_0(z)$ is shown in Table 3.1.

The impulse responses of the subfilters $G_k(z)$ are shown in Tables 3.2 and 3.3. The coefficients of the filter $H_0(z)$ and subfilters $G_k(z)$ are rounded to its respective wordlength. The $WL=6$ for 8-bit interpolator/decimator, $WL=14$ for 16-bit interpolator/decimator. The extra 2-bits are sign bit and gaurd bit.

$$h0\_rounded\_values = \text{round}(h0 \ast 2^W)$$

The wordlength issues and rounding of coefficients are discussed in Chapter 4.
### 3.5 Filter Design

Table 3.2. Impulse response of subfilters $G_0(z), G_1(z), G_2(z)$.

| $q_0(0) = q_1(17) = 0.000423907467916947$ | $g_1(0) = -g_1(17) = -0.0109728393164767$ | $g_2(0) = g_2(17) = 0.0062113898382746$ |
| $q_1(16) = 0.0627829541402154$ | $g_1(16) = 0.0627829541402154$ | $g_2(16) = 0.0627829541402154$ |
| $q_2(0) = q_3(17) = 0.0535689635516715$ | $g_2(0) = -g_2(17) = 0.0535689635516715$ | $g_3(0) = g_3(17) = 0.0535689635516715$ |
| $q_3(16) = 0.41654564052583$ | $g_3(16) = 0.41654564052583$ | $g_4(0) = g_4(17) = 0.41654564052583$ |
| $q_4(0) = q_5(17) = 0.05116943115647$ | $g_4(0) = -g_4(17) = 0.05116943115647$ | $g_5(0) = g_5(17) = 0.05116943115647$ |
| $q_5(16) = 0.6309115036598474$ | $g_5(16) = 0.6309115036598474$ | $g_6(0) = g_6(17) = 0.6309115036598474$ |
| $q_6(0) = q_7(17) = 0.05116943115647$ | $g_6(0) = -g_6(17) = 0.05116943115647$ | $g_7(0) = g_7(17) = 0.05116943115647$ |
| $q_7(16) = 0.10117306368847$ | $g_7(16) = 0.10117306368847$ | $g_0(0) = q_0(17) = 0.0260531771718914$ |
| $q_0(15) = 0.031204811929472$ | $g_0(15) = 0.031204811929472$ | $g_1(0) = q_1(17) = 0.0260531771718914$ |
| $q_1(14) = 0.0168087327062642$ | $g_1(14) = 0.0168087327062642$ | $g_2(0) = q_2(17) = 0.0260531771718914$ |
| $q_2(13) = 0.0077806722572729$ | $g_2(13) = 0.0077806722572729$ | $g_3(0) = q_3(17) = 0.0260531771718914$ |
| $q_3(12) = 0.0027829541027547$ | $g_3(12) = 0.0027829541027547$ | $g_4(0) = q_4(17) = 0.0260531771718914$ |
| $q_4(11) = 0.000423907467916947$ | $g_4(11) = 0.000423907467916947$ | $g_5(0) = q_5(17) = 0.0260531771718914$ |

Table 3.3. Impulse response of subfilters $G_3(z), G_4(z), G_5(z)$.
Chapter 4

VHDL Implementation

4.1 Introduction

In this thesis work, the linear-phase FIR interpolator and decimator are implemented in VHDL (VHSIC Hardware Description Language) where VHSIC stands for Very High Speed Integrated Circuits. A brief introduction to the linear-phase FIR interpolator and decimator was given in Chapter 2.

The VHDL is intended for the circuit simulations and synthesis but not all the VHDL designs are synthesizable. The main applications of VHDL are in CPLDs (Complex Programmable Logic Devices), FPGAs (Field programmable Gate Arrays) and in the field of ASICs (Application Specific Integrated Circuits). The VHDL code can be simulated, synthesized and implemented using several EDA tools [14]. Some EDA tools are Altera’s Quartus II, for Altera’s CPLD/FPGA and Xilinx’s ISE suite, for Xilinx’s CPLD/FPGA. In this thesis work, the VHDL code is simulated in Modelsim, version Modelsim.SE 6.4 and compiled in Quartus II for timing analysis and register usage.

VHDL is mostly used for simulation and synthesis of electronic designs. The process of compiling and mapping the VHDL code into an FPGA or an ASIC is called synthesis. All the VHDL constructs are not suitable for synthesis. For example, the construct ‘wait for 10 ns’ is not synthesizable but valid for simulations. The tools for synthesis of VHDL are inexpensive when compared with ASIC synthesis tools. Both the hardware design and testbenches are portable between design tools and vendors. The VHDL provides technology independent design [14]. To implement the design in new technology, we can go back to the behavioral VHDL description and then implement it the new technology knowing the correct functionality is preserved [14].

4.2 Fixed Point Package

A package called Ieee_proposed.fixed_pkg is used for implementing the fractional-delay block \(d_m^k\) in the interpolator and decimator. The fixed point package
4.3 VHDL Implementation of the Interpolator

The flowchart in Fig. 4.1 explains the step-by-step procedure flow of implementing the linear-phase FIR interpolator. The filter $H_0(z)$ has to be implemented in such a way that the design has to be re-used for a subfilter implementation. The filter $H_0(z)$ coefficients are fixed, so the coefficients are given as constants in the implementation instead of giving them as inputs. The subfilters $G_k(z)$ block has six subfilters with different coefficients but of same order. Instead of designing six subfilters, a subfilter can be designed by making coefficients as inputs. Therefore, six subfilters are implemented by instantiating the subfilter six times.

4.3.1 VHDL Implementation of the Filter $H_0(z)$

The transfer function of an FIR filter is [4]

$$H(z) = \sum_{k=0}^{N-1} h[k] z^{-k}.$$  \hfill (4.1)

In the time domain, we have

$$y[n] = \sum_{k=0}^{N-1} h[k] x[n - k].$$  \hfill (4.2)
The input signal is stored in the shift registers and the output of the shift registers is connected to the multipliers and then to the adders as shown in Fig. 4.2. The VHDL code of filter $H_0(z)$ is shown in 4.1. It is very important that the VHDL code is written in such a way that it is reusable or shared. It should be as generic as possible. By changing the order, $XIN\_WL$ and $COEF\_WL$ in line 13, an FIR filter of any order and wordlength can be implemented. The filter $H_0(z)$ has fixed coefficients, so they can be given as constants, shown in lines 36 and 37. When $RST$ is HIGH the coefficients are converted to signed bits. When $RST$ is LOW the input signal stored in the shift registers is multiplied with coefficients and then added, as shown in between lines 55-64.

**Figure 4.1.** Flowchart for implementing the interpolator.

**Listing 4.1.** VHDL code for filter $H_0(z)$
VHDL Implementation

Figure 4.2. RTL representation of FIR filter.
4.3 VHDL Implementation of the Interpolator

4.3.2 VHDL Implementation of the Subfilters $G_k(z)$

In the block $G_k(z)$ we have six subfilters. Instead of implementing six different subfilters, it is efficient to implement one subfilter and it can be instantiated $L$-times ($L=5$, where $k=0,1,2,3,4,5$) to implement the $G_k(z)$ block. The coefficients are different for every subfilter. A subfilter can be implemented from the filter code of $H_0(z)$, by changing the coefficients to inputs instead of constants. Figure 4.3 shows the instantiation of six subfilters forming the block $G_k(z)$. The VHDL code for the subfilter is given in the appendix A.1.1.

![Figure 4.3. The $G_k(z)$ block with six subfilters.](image-url)

The VHDL code for the block $G_k(z)$ is shown 4.2. The subfilter named $FIR\_FILTER\_16BIT$ is declared first, which is shown in between lines 31-38. From line 36 it is clear that the subfilter has an input port for coefficients and the input port is of type $SIG\_16$, which is a special array declared in the user package. The user package is shown in the appendix. The coefficients of six subfilters are declared in between lines 42-53. In the process (lines 65-89) the coefficients are converted into signed bits. The process executes when there is a change in the reset. All the subfilters instantiated between lines 93-104 are executed simultaneously.

Listing 4.2. VHDL code for block $G_k(z)$

```vhdl
LIBRARY IEEE;
```
ENTITY FARROW_STRUCTURE_16BIT IS
  GENERIC (F :INTEGER:=6;WL:INTEGER:=16;ORDER:INTEGER:=17) ;
  PORT(
    CLK,RST : IN STD_LOGIC;
    XIN : IN SIGNED(WL-1 DOWNTO 0);
    YOUT : OUT SIGNED(2*ORDER TO 0);
    COEF: IN SIG_16(0 TO ORDER) ;
    FOUT :OUT SIG_32(1 TO F)
  ) ;
END FARROW_STRUCTURE_16BIT;

ARCHITECTURE BEHAV OF FARROW_STRUCTURE_16BIT IS

COMPONENT FIR_FILTER_16BIT IS
  GENERIC (WL:INTEGER); PORT(
    CLK,RST : IN STD_LOGIC; XIN : IN SIGNED(WL-1 DOWNTO 0);
    COEF: IN SIG_16(0 TO ORDER) ;
    YOUT : OUT SIGNED(2*WL-1 DOWNTO 0)
  ) ;
END COMPONENT;

ARCHITECTURE COMPONENT DECLARATION

ARRAY DECLARATION

USER DEFINED PACKAGE

CHANGE W H E N WL IS CHANGED

CHANGE COEF_WL CHANGES

BEGIN

PROCESS(rst)

VARIABLE COEF_VAR_1: SIG_16(1 TO ORDER+1);--CHANGE COEF_WL CHANGES
VARIABLE COEF_VAR_2: SIG_16(1 TO ORDER+1);--CHANGE COEF_WL CHANGES
VARIABLE COEF_VAR_3: SIG_16(1 TO ORDER+1);--CHANGE COEF_WL CHANGES
VARIABLE COEF_VAR_4: SIG_16(1 TO ORDER+1);--CHANGE COEF_WL CHANGES
VARIABLE COEF_VAR_5: SIG_16(1 TO ORDER+1);--CHANGE COEF_WL CHANGES
VARIABLE COEF_VAR_6: SIG_16(1 TO ORDER+1);--CHANGE COEF_WL CHANGES
BEGIN
  FOR i IN 0 TO order-1 LOOP
    COEF_VAR_i(i+1):=TO_SIGNED(COEFF,1(i).COEF_WL);
  END LOOP;
END PROCESS;

COMPONENT INSTANTIATED

FIRST SUBFILTER

FIR1: FIR_FILTER_16BIT GENERIC MAP(WL) PORT MAP(CLK, RST, XIN, COEF_FARROW_6, FOUT(1)) ;
4.3.3 VHDL Implementation of the Delay Block $d_m^k$

Figure 4.4. Block diagram for VHDL implementation of the interpolator.

Figure 4.4 shows how an interpolator is implemented in VHDL. The filter $H_0(z)$ and subfilters $G_k(z)$ are instantiated first and then the delay block code is written. The components $H_0(z), G_k(z)$ instantiation is showed in between lines 31-44 in 4.3. These components are declared in lines 58 and 59. The output of $G_k(z)$ is connected to a signal $SOUT$ and the output of $H_0(z)$ is connected to a signal $H0\_SIGN$.

The delay block $d_m^k$ consists of $M - 1$ delay multiplier chain blocks, where $M$ is interpolation factor. Each delay multiplier chain block gives one polyphase component $H_m$, where $m = 1, 2, ..., M - 1$. The filter $H_0(z)$ gives the zeroth polyphase component $H_0$. The lower part of Fig. 4.5 shows a delay multiplier chain block. The Fig. 4.5 shows how the subfilters and delay multiplier chain block can form a Farrow structure. The sampled value from subfilter $G_5(z)$ is multiplied by the fractional-delay multiplier value and then added to the sampled value from the subfilter $G_4(z)$. This value is again multiplied with the fractional-delay multiplier value and then added to the sampled value of subfilter $G_3(z)$, this process ends after adding the sampled value from the subfilter $G_0(z)$ as shown between lines 110-117 in the code below. The values $d_m$ are generated using Eq. (4.3).
Even Factor $M = 8$ & Odd Factor $M = 5$

<table>
<thead>
<tr>
<th>Even Factor $M = 8$</th>
<th>Odd Factor $M = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_0=1/2$</td>
<td>$d_0=1/2$</td>
</tr>
<tr>
<td>$d_1=3/8$</td>
<td>$d_1=3/10$</td>
</tr>
<tr>
<td>$d_2=1/4$</td>
<td>$d_2=1/10$</td>
</tr>
<tr>
<td>$d_3=1/8$</td>
<td>$d_3=-1/10$</td>
</tr>
<tr>
<td>$d_4=0$</td>
<td>$d_4=-3/10$</td>
</tr>
<tr>
<td>$d_5=-1/8$</td>
<td></td>
</tr>
<tr>
<td>$d_6=-1/4$</td>
<td></td>
</tr>
<tr>
<td>$d_7=-3/8$</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3. Fractional-delay multiplier values for $M = 8$ and $M = 5$.

Equation. (4.3) is a modified form of Eq. (4.4).

$$d_{m+1} = \frac{-1}{M} + d_m$$

(4.3)

where $d_0 = 1/2$.

$$d_m = \frac{-m}{M} + \frac{1}{2}$$

(4.4)

The generation of fractional-delay multiplier value ($d_m$) is shown in between lines 93-103. The values $d_m$ for $M=8$ and $M=5$ are given in Table 4.3. The Fractional-delay multiplier values are converted into binary representation using 'to_sfixed'. Example 3.1 shows how a fractional number is converted into binary. As the addition and multiplication rules are different compared with the normal binary addition and multiplication, truncation of bits are required. RESIZE is used for truncating the bits and an example is shown below.

**Example 4.1: conversion and resizing**

```vhdl
variable ADD(16 downto -5);
ADD:=0000000000000000.00000 initially.
ADD := to_sfixed(15.5, ADD);
The binary value of ADD is: ADD:=0000000000001111.10000.

To change the wordlength from 16-bit to 8-bit
variable size_ADD (8 downto -5)
size_ADD := resize(ADD, size_ADD);
size_ADD=00001111.10000
```

Figure 4.6 shows how a delay multiplier chain block for 16-bit interpolator is implemented and also shows where truncation of bits is performed. The sampled values of the subfilters are 16-bit (15 downto 0) and it is multiplied with 8-bit (2 downto -5) delay multiplier $d_m$ value results in 24-bit (18 downto -5). The multiplied value is added to the 16-bit (15 downto 0) sample from the next subfilter resulting in 25-bits (19 downto -5). This 25-bit value is truncated to 16-bit (15 downto 0). The $d_m$ values are small and when the sampled output of the subfilters
are multiplied by the delay multiplier values they become very small. So five LSB and four MSB bits are truncated, line 114 from the code shows the truncation of bits after addition.

Listing 4.3. VHDL code for the interpolator

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
LIBRARY ieee_proposed;
USE ieee_proposed.fixed_pkg.all;
USE WORK_PKG_SIGNED.ALL;
USE WORK_PKG_UNSIGNED.ALL;
LIBRARY WORK;
USE WORK.ALL;
--------------------------------------------------ENTITY
```
**Figure 4.6.** Delay multiplier chain block with truncation.
4.3 VHDL Implementation of the Interpolator

```vhdl
--report 'Starting process PI' severity note;
if rising_edge(clk) then
  BEGIN
    --GENERATING Dm VALUES
    M:=to_sfixed(IP_factor,M);
    one:=to_sfixed(1,one);
    value:=(one/M);
    TEMP_IP_FACTOR:=RESIZE(-VALUE,TEMP_IP_FACTOR);
    FOR I IN 0 TO IP_FACTOR-1 LOOP
      Dm_SIGNED(I):=(TEMP_IP_FACTOR)+Dm_SIGNED(I);
    END LOOP;

    --CONVERTING THE INTEGER 'SOUT', TO FIXED POINT
    FOR I IN 0 TO F-1 LOOP
      SOUT_TEMP(I):=TO_SFIXED(SOUT(I),SOUT_TEMP(I));
    END LOOP;

    --MULT AND ADDING FOR EVERY INTERPOLATION FACTOR
    FOR I IN 1 TO IP_FACTOR-1 LOOP
      PROD:=SOUT_TEMP(I)*Dm_SIGNED(I);
      FOR J IN 1 TO F-1 LOOP
        ADD:=PROD+SOUT_TEMP(J);
        ADD_T:=RESIZE(ADD,ADD_T);
        PROD:=ADD_T+Dm_SIGNED(I);
      END LOOP;
      IOUT(1)<=ADD;
      WRITEOUT(1)<=ADD;
      END LOOP;
      FIN<=SOUT;
      H0:=TO_SFIXED(H0_SIGN,H0);
    end if;
end if:
```

```vhdl
--report 'Changing time' severity note;
if rising_edge(clk) then
  BEGIN
    --OPEN THE FILE write.txt FROM THE SPECIFIED LOCATION FOR WRITING
    file_open(file_pointer, '/edu/venda501/fir_filter/1.txt', APPEND_MODE);
    firsttime := false;
  end if;
  BIN_VALUE:=WRITEOUT;
  FOR I IN 0 TO IP_FACTOR-1 LOOP
    BIN_VALUE(I)<=BIN_VALUE(1);
    K:=0;
    V:=WL+COEF_WL+3;
    FOR J IN V DOWNTO -5 LOOP
      IF K=0 THEN
        BIN_VALUE(I)(J) := '1' THEN
          LINE_CONTENT(K) := '1';
        ELSE
          BIN_VALUE(I)(J) := '0' THEN
            LINE_CONTENT(K) := '0';
          END IF;
          K:=K+1;
        END IF;
      END LOOP;
      IF rising_edge(clk) then
        BEGIN
          --WRITE THE CONTENTS INTO THE FILE
          write(line_num,line_content);
          write(line(file_pointer,line_num));
          end if;
        end loop;
      END BEHAV;
```
4.4 Wordlength Issues in Interpolator

It is very important to consider the roundoff error and sign of the integer bit while converting into a binary value. A sign bit is used to represent the sign of the number and a guard bit is used to reduce the roundoff error. We have one sign bit and one guard bit in a 16-bit binary value. The wordlength of coefficients for the filter and subfilters are 14-bits. The coefficient values for the filter $H_0(z)$ and subfilters $G_k(z)$ are shown in the appendix.

The fractional-delay multiplier has three decimal bits and five fractional bits (2 downto -5). There are some fractional value which cannot be exactly converted into a binary value. For example, $d_1=0.3$ for $M=5$, after conversion the binary value is '000.01001' which is not equal to 0.3. The exact fractional value of '000.01001' is 0.28125.

4.5 VHDL Implementation of the Decimator

To implement a decimator, the first step is to implement the delay block $d_{mk}$, the second step is to implement the subfilters $G_k(z)$, and then we make use of the filter $H_0(z)$ which is designed for the interpolator. The delay block $d_{mk}$, the subfilters $G_k(z)$, and the filter $H_0(z)$ are instantiated in one design to form the decimator as shown in Fig. 4.7. The decimator is a $(M-1)$-input 1-output system, where $M$ is the decimation factor. The block diagram of the decimator is shown in Fig. 4.9.

![Figure 4.7. Block diagram for VHDL implementitation of the decimator.](image)

4.5.1 VHDL Implementation of the Delay Block $d_{mk}$

The delay block consists of $M$-1 delay multiplier chain blocks and six adders, where $M$ is the decimation factor. The blocks with dotted lines in Fig. 4.9 are the delay multiplier chain blocks. $H_1, H_2, \ldots, H_{M-1}$ are input samples and
4.5 VHDL Implementation of the Decimator

$G_{out}(0), G_{out}(1)....G_{out}(5)$ are output ports to the delay block. Every delay multiplier chain block has five outputs and they are $G_0, G_1, G_2, G_3, G_4, G_5$. The input $H_m$, where $m=1,2,...M-1$ is multiplied by the fractional-delay multiplier value resulting in $G_k$, where $k=1,2...5$. The $G_0$ has the same value of $H_m$ as shown in Fig 4.9.

A 16-bit delay multiplier chain block is shown in Fig. 4.8. A 16-bit input sample is multiplied by 8-bit fractional-delay multiplier value resulting in 24-bits (18 downto -5). The fractional-delay multipliers have very small values, so when the input samples are multiplied by the $d_m$ value they become comparatively very small. Thus we can truncate the unnecessary bits, 8-bits are truncated from 24-bits resulting in 16-bits (15 downto 0).

The first output of every delay multiplier chain block are summed to form $G_{out}(0)$ and every second output of the delay multiplier chain block are summed to form $G_{out}(1)$ and so forth. The wordlength of the delay block increases with increase in the decimation factor because of increase in delay multiplier chain blocks. In order to overcome this problem the output of the delay block is fixed for 32-bits (31 downto 0). Due to these truncations a 16-bit decimator works for 14-bits. The VHDL code for the delay block is given in the appendix.

4.5.2 VHDL Implementation of Subfilters

In the interpolator, the subfilters have single input fed to every subfilter. But in the decimator, the sum of the first output of every delay multiplier chain block is fed to the subfilter $G_0(z)$, the sum of second output of every delay multiplier chain block is fed to $G_1(z)$ and so forth. The subfilter block in decimator is transpose
Figure 4.9. Block diagram of Decimator.
of the subfilter block in interpolator. Hence the output of the subfilters block is
the sum of sampled outputs of every subfilter.

The delay block $d_{m}$, subfilters $G_{k}(z)$ and the filter $H_{0}(z)$ are instantiated in
between lines 69–75 as shown in 4.4.

**Listing 4.4.** VDL code for the decimator

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE WORK_PKG_SIGNED.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
USE STD.TEXTIO.ALL;
LIBRARY WORK;
USE WORK.ALL;

----ENTITY----
ENTITY DECIMATOR_BLOCK IS
  GENERIC(WL: INTEGER:=16; xin_wl : integer :=16; COEF_WL:INTEGER:=16;
          F :INTEGER:=6; DECIMATOR_FACTOR:INTEGER:=8) ;
  PORT ( 
    CLK,RST: IN STD_LOGIC;
    DECIMATOR_IN: IN SIG_16(0 TO DECIMATOR_FACTOR
                             -1);
    DECIMATOR_OUT: OUT SIGNED(WL+COEF_WL
                             -1 DOWNTO 0) 
  ) ;
END DECIMATOR_BLOCK;

ARCHITECTURE BEHAV OF DECIMATOR_BLOCK IS
  COMPONENT UBLOCK_DECIMATOR IS
    generic ( decimator_factor : integer ) ;
    PORT ( 
      CLK,RST : IN STD_LOGIC;
      DECIMATOR_IN: IN SIG_16(1 TO DECIMATOR_FACTOR
                               -1);
      GOUT : OUT SIG_16(0 TO F-1)----change wl is changed
    ) ;
  END COMPONENT;

  COMPONENT FARROW_STRUCTURE_16BIT IS
    PORT ( 
      CLK,RST : IN STD_LOGIC;
      GOUT_IN : IN SIG_16 (0 TO F-1) ;----- CHANGE WHEN WL IS CHANGED
      FOUT :OUT SIGNED(COEFF_WL-1 DOWNTO 0) 
    ) ;
  END COMPONENT;

  COMPONENT H0_FILTER IS
    PORT ( 
      CLK,RST : IN STD_LOGIC ;
      XIN : IN SIGNED(xin_WL-1 DOWNTO 0) ; ----INPUT PORT
      YOUT: OUT SIGNED(xin_WL+COEF_WL-1 DOWNTO 0) ; -----OUTPUT PORT
    ) ;
  END COMPONENT;

  ----SIGNALS----
  SIGNAL FOUT_SIGNAL: SIGNED(WL+COEF_WL-1 DOWNTO 0) ;
  SIGNAL GOUT_SIGNAL : SIG_16(0 TO F-1) ;
  SIGNAL FIRROUT_SIGNAL: SIGNED(xin_WL+COEF_WL-1 DOWNTO 0) ;
  SIGNAL INPUT_SIGNAL : SIG_16(1 TO DECIMATOR_FACTOR-1);
  signal write_out : signed ( wl+coef_wl
                             -1 downto 0) ;
  BEGIN
    PROCESS(clk)
      VARIABLE VAR : SIG_16(1 TO DECIMATOR_FACTOR-1);
      BEGIN
        FOR I IN 1 TO DECIMATOR_FACTOR-1 LOOP
          VAR(I) :=DECIMATOR_IN(I) ;
        END LOOP;
        DECIMATOR_OUT <= GOUT_SIGNAL+FOUT_SIGNAL;
        write_out<= FIRROUT_SIGNAL+FOUT_SIGNAL;
      END PROCESS;

    B1:UBLOCK_DECIMATOR g e n e r i c map( decimator_factor ) PORT MAP (CLK,RST,
    INPUT_SIGNAL,GOUT_SIGNAL) ;

    FA RROW:FARROW_STRUCTURE_16BIT PORT MAP(CLK,RST,GOUT_SIGNAL,FOUT_SIGNAL) ;

    FILTER :H0_FILTER PORT MAP(CLK,RST,DECIMATOR_IN(0) ,FIRROUT_SIGNAL ) ;

    FILE_WRITE: process(clk)
      file file_pointer : text;
```
variable line_content : string (1 to 32);
variable write_value : Signed (wl+coef_wl DOWNTO 1);
variable line_num : line;
variable i, j, K : integer := 0;
variable char : character := '0';
variable firsttime : boolean := true;

begin
−− − − − − − − − − − − − − − − − −
opening a file and writing to output.txt
−− − − − − − − − − − − − − − − − −
begin
if firsttime then
  −− Open the file write.txt from the specified location for writing
  file_open (file_pointer, "Decimator/outputfile10.txt", APPEND_MODE);
  firsttime := false;
  end if;
  −− Write VALUE := WRITE_OUT;
  k := to_integer (write_out);
  if rising_edge (clk) then
    −− Write line_num.K: -- write the line.
    writeline (file_pointer, line_num);
  end if;
end process;

END BEHAV;}
Chapter 5

Testbench and Simulation Results

5.1 Introduction

Once the design is completed, it has to be tested in order to check whether it performs as desired. The main advantage with VHDL is that a testbench can be designed to apply stimulus to the design that has to be tested and the testbench is portable between VHDL tools from different vendors. VHDL is not only a hardware description language but also a stimulus definition language. A VHDL testbench has an empty entity and an architecture with component that has to be tested, internal signals for input and output, system clock process and stimulus process.

5.2 Design Flow for Testing Interpolator

The design flow for testing the interpolator is shown in Fig. 5.1.

5.2.1 Generating a Sine Wave

A sine wave is generated in MATLAB with

\[ Xin = \sin(n \cdot wT); n = 0 \text{ to } 2000 - 1; wT = \pi/6. \]

The generated sine wave is rounded with the input wordlength, in this case the input signal wordlength is 16-bit.

\[ Xin_{\text{rounded}} = \text{round}(Xin \cdot 2^{(wl)}); \]

The rounded sine wave is written to a text file using

\[ \text{Fopen('Input.txt', 'w')} \text{ and } \text{Fprintf(fid, 'd/n', Xin_{rounded}).} \]
5.2.2 VHDL Testbench

A testbench is a empty design entity which serves as a host environment for another design entity. The entity under test is called "unit under test", which has to be instantiated in the architecture. The VHDL code for the interpolator testbench is shown below. The system clock and reset are generated in clock process and reset process. The input stimulus is generated using MATLAB, and the samples are written into a text file named "INPUT.TXT". A single sample is given for every clock period. Simulation results are shown in Fig. 5.2. From Fig. 5.2 it is very difficult to verify the output. Therefore, the output of the interpolator is written to a text file as shown in lines 126-155 in the interpolator VHDL code. The output of the interpolator (IOUT) is a 41-bit value, of which 36 are integer bits and 5 fractional bits.

5.2.3 Simulation results

A code is written in MATLAB to read the output text file generated by the interpolator and also to convert the 41-bit binary value to a rational value. The code is shown in 5.1.

Listing 5.1. Code to read the output file from the interpolator

```matlab
clc; close all; clear all; format long; format compact;
FracPart = 5; DecPart = 36; Bits = FracPart+DecPart+1;
M = 3;

fid = fopen('~/edu/venda501/thesis/inputs/input_M2.txt','r');
xin = fscanf(fid, '%d');
fclose(fid); freqz(xin)
```
Figure 5.2. output of interpolator in Modelsim.
Figure 5.3. Testbench for the Interpolator.
Figure 5.4 shows the input signal and the interpolated signal. Observe that for every input sample there are three output samples. The frequency response of the interpolator output is plotted in Fig. 5.5. The interpolator is simulated with different interpolation factors ($M=3,6,10$), and the frequency response is shown in Figs. [5.5-5.7]. The interpolator output is not quantized, so there is no noise in the plots.

![Figure 5.4. Sinusoidal signal and the interpolated sinusoidal signal ($M=3$).](image)

### 5.3 Testbench for the Decimator

The process for testing the decimator is the same as testing the interpolator. An upsampled sine wave is given as an input. The output of the decimator (decimator out) is a 32-bit binary value, it has 32 integer bits. The output text file generated by the decimator has integer values, implying that conversion of bits is not required. The frequency response of the decimator output is plotted in Figs. [5.9-5.11].
Figure 5.5. Frequency response for the interpolator output.

Figure 5.6. Frequency response for the interpolator output followed by perfect filter.
5.4 Synthesis and Timing Analysis

The report below shows the total use of the logical elements and logical registers by Interpolator. This synthesis is done by selecting the Cyclone IV GX family and EP4CGX30CF23C6 device. Table 5.1 shows the number of logical elements and logical array blocks used for the interpolator and decimator. The interpolation factor is 10 and the decimation factor is 4. Due to the difference in the factors there is a difference in the LEs, LABs, and $F_{max}$. The coefficients of the interpolation/decimation filters are stored in the registers. The output of interpolator/decimators are 32-bits, for the 16-bit interpolator/decimator. In this implementation the 16-bit interpolator output wordlength is 40 bits in which 8 bits are truncated. The performance of the interpolator can be increased by using advanced multiplication techniques.

5.4.1 Synthesis and Time Quest Analyzer Reports
Figure 5.8. Output of the decimator in Modelsim.
Figure 5.9. Frequency response for the decimator output.

Figure 5.10. Frequency response for the decimator output.
Figure 5.11. Frequency response for the decimator output.
From the time quest timing analysis report of the interpolator the $F_{max}$=12.44 MHz for the slow 1200 mv 85c model and $F_{max}$=13.95 MHz for the slow 1200 mv 0c model. For the decimator the $F_{max}$=53.39 MHz for the slow 1200 mv 85c model and $F_{max}$=59.99 MHz for the slow 1200 mv 0c model.
<table>
<thead>
<tr>
<th>Component</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 input functions</td>
<td>3901</td>
<td></td>
</tr>
<tr>
<td>3 input functions</td>
<td>9249</td>
<td></td>
</tr>
<tr>
<td>&lt;=2 input functions</td>
<td>1837</td>
<td></td>
</tr>
<tr>
<td>Register only</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Logic elements by mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal mode</td>
<td>5811</td>
<td></td>
</tr>
<tr>
<td>Arithmetic mode</td>
<td>9170</td>
<td></td>
</tr>
<tr>
<td>Total registers*</td>
<td>2,368 / 30,191 (8%)</td>
<td></td>
</tr>
<tr>
<td>Dedicated logic registers</td>
<td>2,368 / 29,440 (8%)</td>
<td></td>
</tr>
<tr>
<td>I/O registers</td>
<td>0 / 751 (0%)</td>
<td></td>
</tr>
<tr>
<td>Total LABs partially or completely used</td>
<td>1,099 / 1,840 (60%)</td>
<td></td>
</tr>
<tr>
<td>User inserted logic elements</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Virtual pins</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>I/O pins</td>
<td>98 / 167 (59%)</td>
<td></td>
</tr>
<tr>
<td>Clock pins</td>
<td>4 / 6 (67%)</td>
<td></td>
</tr>
<tr>
<td>Dedicated input pins</td>
<td>0 / 16 (0%)</td>
<td></td>
</tr>
<tr>
<td>Global signals</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MSRs</td>
<td>0 / 120 (0%)</td>
<td></td>
</tr>
<tr>
<td>Total block memory bits</td>
<td>0 / 1,105,920 (0%)</td>
<td></td>
</tr>
<tr>
<td>Total block memory implementation bits</td>
<td>0 / 1,105,920 (0%)</td>
<td></td>
</tr>
<tr>
<td>Embedded Multiplier 9-bit elements</td>
<td>0 / 160 (0%)</td>
<td></td>
</tr>
<tr>
<td>PLLs</td>
<td>0 / 4 (0%)</td>
<td></td>
</tr>
<tr>
<td>Global clocks</td>
<td>2 / 20 (10%)</td>
<td></td>
</tr>
<tr>
<td>JTAGs</td>
<td>0 / 1 (0%)</td>
<td></td>
</tr>
<tr>
<td>CRC blocks</td>
<td>0 / 1 (0%)</td>
<td></td>
</tr>
<tr>
<td>ASMI blocks</td>
<td>0 / 1 (0%)</td>
<td></td>
</tr>
<tr>
<td>GXB Receiver channel PCSs</td>
<td>0 / 4 (0%)</td>
<td></td>
</tr>
<tr>
<td>GXB Receiver channel PMAs</td>
<td>0 / 4 (0%)</td>
<td></td>
</tr>
<tr>
<td>GXB Transmitter channel PCSs</td>
<td>0 / 4 (0%)</td>
<td></td>
</tr>
<tr>
<td>GXB Transmitter channel PMAs</td>
<td>0 / 4 (0%)</td>
<td></td>
</tr>
<tr>
<td>Impedance control blocks</td>
<td>0 / 3 (0%)</td>
<td></td>
</tr>
<tr>
<td>Average interconnect usage (total/H/V)</td>
<td>18% / 16% / 21%</td>
<td></td>
</tr>
<tr>
<td>Peak interconnect usage (total/H/V)</td>
<td>31% / 26% / 39%</td>
<td></td>
</tr>
<tr>
<td>Maximum fan-out node</td>
<td>CLK-input clkctrl</td>
<td></td>
</tr>
<tr>
<td>Maximum fan-out</td>
<td>2368</td>
<td></td>
</tr>
<tr>
<td>Highest non-global fan-out signal</td>
<td>RST-input</td>
<td></td>
</tr>
<tr>
<td>Highest non-global fan-out</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>Total fan-out</td>
<td>54115</td>
<td></td>
</tr>
<tr>
<td>Average fan-out</td>
<td>3.07</td>
<td></td>
</tr>
</tbody>
</table>

* Register count does not include registers inside RAM blocks or DSP blocks.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis work, the interpolators and decimators work efficiently up to a conversion factor of 20. For larger factors new subfilters have to be designed. This design gives the flexibility to implement the different wordlength interpolators and decimators. This implementation proves that the design proposed in [8] works well for conversions by prime numbers and is flexible for the conversion factors. The fixed point package made the design more flexible and easier. By instantiating the delay block, several sampling rate converters can be implemented simultaneously, with the same set of filters. The only additional cost to obtain another converter is to add another delay block. Hence, several sampling rate converters can be implemented simultaneously at low cost. The interpolator and decimator cannot be implemented on the FPGA board, because the FPGA board cannot read or write the text files given in the code. The program Teraterm or Hyperterminal can be used to transfer a text file with large amount of binary data.

6.2 Future Work

- One can implement the design without using the fixed point package and verify the performance and registers usage.
- More advanced techniques can be used for truncation of bits.
- To implement the design on the FPGA board one can use the teraterm program. By using this program the entire text file can be transferred to the FPGA board via a PC serial port.
- Another way to implement the design on the FPGA board is by using the SDRAM. The input sequence is saved in the SDRAM and the output of the interpolator or decimator has to be pointed to the remaining memory address in the SDRAM. The SDRAM on FPGA board is small, so we cannot give many input samples.
Appendix A

Appendix

A.1 VHDL Code

A.1.1 VHDL Code for Subfilter

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
USE WORK_PKG_SIGNED.ALL;

LIBRARY WORK;
USE WORK.ALL;

ENTITY FIR_FILTER_16BIT IS
  GENERIC (ORDER :INTEGER:=17;WL:INTEGER:=16;COEF_WL:INTEGER:=16);
  WL IS WORDLENGTH OF XIN(INPUT SIGNAL);
  COEF_WL IS WORDLENGTH OF COEFFICIENTS OR TAPS;
  PORT (CLK,RST : IN STD_LOGIC ;
         XIN : IN SIGNED(WL DOWNTO 0);
         COEF: IN SIG_16(0 TO ORDER);
         YOUT: OUT SIGNED(COEF_WL+WL DOWNTO 0) );
END FIR_FILTER_16BIT ;

ARCHITECTURE BEHAVIOURAL OF FIR_FILTER_16BIT IS
  BEGIN
    INTAL :PROCESS(CLK,RST)
      VARIABLE M_OUT,ADD_OUT:SIGNED(COEF_WL+WL DOWNTO 0) :=(OTHERS=>0);
      BEGIN
        IF (RST='1') THEN
          FOR I IN ORDER DOWNTO 0 LOOP
            FOR j in WL DOWNTO 0 loop
              DFF(I)(j) <=0;
            end loop;
          END LOOP;
        ELSEIF (rising_edge(clk)) then
          ADD_OUT:= COEF(0)*XIN;
          FOR I IN 1 TO ORDER-1 LOOP
            M_OUT:=COEF(I)*DFF(ORDER-1-I);
            ADD_OUT:=ADD_OUT+M_OUT;
          END LOOP;
        END IF
      END PROCESS;
    END PROCESS;
  END BEHAVIOURAL;
```

53
A.1.2 VHDL Code for Delay Block of Decimator

--- Delay Block of Decimator ---

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
LIBRARY ieee_proposed;
USE ieee_proposed.fixed_pkg.ALL;
USE WORK_PKG_SIGNED.ALL;

LIBRARY ieee;
USE ieee.ALL;

ENTITY UBLOCK_DECIMATOR IS
  GENERIC(XIN_WL:INTEGER:=16;F:INTEGER:=6;
    DECIMATOR_FACTOR:INTEGER:=6;Dm_WL:INTEGER:=8);
  PORT (CLK,RST:IN STD_LOGIC;
    DECIMATOR_IN:IN SIG_16(1 TO DECIMATOR_FACTOR-1);
    GOUT:OUT SIG_16(0 TO F-1))
END UBLOCK_DECIMATOR;

ARCHITECTURE BEHAV OF UBLOCK_DECIMATOR IS
  TYPE Dm_VAR_ARRAY IS ARRAY(0 TO DECIMATOR_FACTOR-1) OF SFIXED(1 DOWNTO Dm_WL+2);
  TYPE Dm_TEMP_ARRAY IS ARRAY(1 TO DECIMATOR_FACTOR-1) OF SFIXED(2 DOWNTO Dm_WL+2);
  TYPE G_VAR_ARRAY IS ARRAY(0 TO F-1) OF SFIXED(XIN_WL DOWNTO 0);
  TYPE G_TEMP_ARRAY IS ARRAY(1 TO F-1) OF SFIXED(XIN_WL DOWNTO Dm_WL+1);

BEGIN
  PROCESS(CLK,RST) VARIABLE Dm_VAR:Dm_VAR_ARRAY;
  VARIABLE Dm_TEMP:Dm_TEMP_ARRAY;
  VARIABLE G_VAR:G_VAR_ARRAY;
  VARIABLE G_TEMP:G_TEMP_ARRAY;
  VARIABLE GOUT_VAR:SIG_16(0 TO F-1);
  VARIABLE TEMP1:SFIXED(1 DOWNTO Dm_WL+2);
  VARIABLE TEMP2:SIGNED(XIN_WL DOWNTO 0);
  VARIABLE once:SFIXED(XIN_WL DOWNTO 0);
  VARIABLE d:SFIXED(XIN_WL DOWNTO Dm_WL+1);
  VARIABLE value:SFIXED(XIN_WL DOWNTO XIN_WL+1);

  IF (RST='1') THEN
    FOR I IN 0 TO F-1 LOOP
      FOR J IN 0 TO 15 LOOP
        GOUT_VAR(I)(J):=0;
      END LOOP;
    END LOOP;
  END IF;

  ELSIF (RISING_EDGE(CLK)) THEN
    once:=TO_SIGNED(1,once);
    d:=TO_SIGNED(DM_VAR(1),Dm_VAR(1));
    value:=value*d;
    temp1:=RESIZE(-value,temp1);
  END IF;

  Dm_VAR(0):=TO_SFIXED(0,5,Dm_VAR(0));

  FOR I IN 1 TO DECIMATOR_FACTOR-1 LOOP
    Dm_TEMP(I):=Dm_TEMP(I-1).
    Dm_VAR(I):=RESIZE(Dm_TEMP(I),Dm_VAR(I));
    G_VAR(I):=TO_SFIXED(DECIMATOR_IN(I),G_VAR(I));
  END LOOP;

  FOR J IN 1 TO F-1 LOOP
    G_TEMP(J):=Dm_VAR(J-1)*G_VAR(J-1);
    G_VAR(J):=RESIZE(G_TEMP(J),G_VAR(0));
  END LOOP;

  FOR J IN 0 TO F-1 LOOP
    TEMP2:=TO_SIGNED(G_VAR(J).XIN_WL);
    GOUT_VAR(J):=TEMP2+GOUT_VAR(J);
A.1 VHDL Code

END LOOP;
END LOOP;
GOUT<=GOUT_VAR;
END IF;
END PROCESS;
END BEHAV;

A.1.3 VHDL Code for Farrow Structure of Decimator

--- Farrow structure for Decimator ---------------
-- MAIN CODE ---------------------------------
LIBRARY IEEE;
USE IEEE. STD_LOGIC_1164.ALL;
USE IEEE. STD_LOGIC_UNSIGNED.ALL;
USE IEEE. NUMERIC_STD.ALL;
USE WORK.PKG_SIGNED.ALL;

LIBRARY WORK;
USE WORK.ALL;

ENTITY ublk_FARROW_STRUCTURE IS
             DECIMATOR_FACTOR : INTEGER:=6; XIN_WL : INTEGER:=16);
    PORT( 
        CLK, RST : IN STD_LOGIC;
        DECIMATOR_IN : IN SIG_16(1 TO DECIMATOR_FACTOR-1);
        GOUT_IN : OUT SIG_16(0 TO F-1);
        FOUT : OUT SIGNED(COEF_WL+WL DOWN TO 0)
    );
END ublk_FARROW_STRUCTURE;

ARCHITECTURE BEHAV OF ublk_FARROW_STRUCTURE IS

COMPONENT H0_FILTER IS
    PORT( 
        CLK, RST : IN STD_LOGIC;
        XIN : IN SIGNED(XIN_WL-1 DOWN TO 0);
        YOUT: OUT SIGNED((XIN_WL+COEF_WL-1) DOWN TO 0)
    );
END COMPONENT;

COMPONENT UBLOCK_DECIMATOR IS
    PORT( 
        CLK, RST : IN STD_LOGIC;
        DECIMATOR_IN : IN SIG_16(1 TO DECIMATOR_FACTOR-1);
        GOUT : OUT SIG_16(0 TO F-1)--- change when WL is changed
    );
END COMPONENT;

COMPONENT FIR_FILTER IS
    GENERIC (WL : INTEGER);
    PORT( 
        CLK, RST : IN STD_LOGIC;
        XIN : IN SIGNED(WL-1 DOWN TO 0);
        COEF : IN SIG_16(0 TO ORDER);
        YOUT: OUT SIGNED(COEF_WL+WL-1 DOWN TO 0)
    );
END COMPONENT;

type coefficients is array (0 to order) of integer;
TYPE FIR_OUT_ARRAY IS ARRAY (0 TO F-1) OF SIGNED(COEF_WL+WL-1 DOWN TO 0);
constant coef_1 : coefficients :=(33, -69, 164, -323, 582, -989, 1699, -3230, 6933);
constant coef_2 : coefficients :=(-254, 462, -711, 1058, -1510, 1978, -2706, 4361,
                              -23066, 23066, -4361, 2706, -1978, 1510, -1058, 711, -462, 254);
constant coef_3 : coefficients :=(-101, 110, -636, 1410, -2635, 4605, -7911, 14322,
                              -9288, -9288, 14322, -7911, 4605, -2635, 1410, -636, 110, -101);
constant coef_4 : coefficients :=(62, -973, 1276, -1983, 3213, -4208, 6917, -13282,
                              23043, -23043, 13282, -6917, 4208, -3213, 1983, -1276, 973, -62);
constant coef_5 : coefficients :=(-388, -259, 801, -1357, 2033, -3390, 5255, -6306,
                              3064, 3064, -6306, 5255, -3390, 2033, -1357, 801, -259, -388);
constant coef_6 : coefficients := (44, -139, -43, 99, -157, 347, -99, 43, 139, -44);

-- SIGNAL DECLARATION

SIGNAL GOUT_SIGNAL: SIG_16(0 TO F-1);

SIGNAL FIR_OUT : SIG_32(1 to F-1);

SIGNAL H0_OUT : SIG_32(XIN WL : COEF WL - 1 DOWNTO 0);

SIGNAL INPUT_SIGNAL: SIG_16(1 TO DECIMATOR_FACTOR-1);

SIGNAL COEF_SIGNAL_1: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
SIGNAL COEF_SIGNAL_2: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
SIGNAL COEF_SIGNAL_3: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
SIGNAL COEF_SIGNAL_4: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
SIGNAL COEF_SIGNAL_5: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
SIGNAL COEF_SIGNAL_6: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES

SIGNAL TEST : ;

BEGIN

PROCESS(clk)

VARIABLE COEF_VAR_1: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
VARIABLE COEF_VAR_2: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
VARIABLE COEF_VAR_3: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
VARIABLE COEF_VAR_4: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
VARIABLE COEF_VAR_5: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES
VARIABLE COEF_VAR_6: SIG_16(0 TO ORDER) -- CHANGE COEF WL CHANGES

VARIABLE INPUT_VAR: SIG_16(1 TO DECIMATOR_FACTOR-1)

begin

if rst = '1' then

for i in 0 to order loop

COEF_VAR_1(I) := TO_SIGNED(COEFF_1(I), COEF_WL);
COEF_VAR_2(I) := TO_SIGNED(COEFF_2(I), COEF_WL);
COEF_VAR_3(I) := TO_SIGNED(COEFF_3(I), COEF_WL);
COEF_VAR_4(I) := TO_SIGNED(COEFF_4(I), COEF_WL);
COEF_VAR_5(I) := TO_SIGNED(COEFF_5(I), COEF_WL);
COEF_VAR_6(I) := TO_SIGNED(COEFF_6(I), COEF_WL);

END LOOP;

COEF_SIGNAL_1 <= COEF_VAR_1;
COEF_SIGNAL_2 <= COEF_VAR_2;
COEF_SIGNAL_3 <= COEF_VAR_3;
COEF_SIGNAL_4 <= COEF_VAR_4;
COEF_SIGNAL_5 <= COEF_VAR_5;
COEF_SIGNAL_6 <= COEF_VAR_6;
end if;

END PROCESS;

process2 : process(clk)

begin

if rising_edge(clk) then

for i in 1 to decimator_factor-1 loop

input_var(i) := decimator_in(i);

end loop;

input_signal<=input_var;

end if;

end process;

-- COMPONENT DECLARATION

H0_FILTER :

H0_FILTER PORT MAP(CLK, RST, DECIMATOR_IN(0), H0_OUT);

-- DELAY BLOCK

UBLOCK : UBLOCK_DECIMATOR PORT MAP(CLK, RST, input_signal, GOUT_SIGNAL);

GOUT_N : GOUT_SIGNAL;

FIR FILTER

FIR1 : FIR_FILTER GENERIC MAP(WL) PORT MAP(CLK, RST, GOUT_SIGNAL(5), COEF_SIGNAL_6, FIR_OUT(1));
FIR2 : FIR_FILTER GENERIC MAP(WL) PORT MAP(CLK, RST, GOUT_SIGNAL(4), COEF_SIGNAL_5, FIR_OUT(2));
FIR3 : FIR_FILTER GENERIC MAP(WL) PORT MAP(CLK, RST, GOUT_SIGNAL(3), COEF_SIGNAL_4, FIR_OUT(3));
FIR4 : FIR_FILTER GENERIC MAP(WL) PORT MAP(CLK, RST, GOUT_SIGNAL(2), COEF_SIGNAL_3, FIR_OUT(4));
FIR5 : FIR_FILTER GENERIC MAP(WL) PORT MAP(CLK, RST, GOUT_SIGNAL(1), COEF_SIGNAL_2, FIR_OUT(5));
FIR6 : FIR_FILTER GENERIC MAP(WL) PORT MAP(CLK, RST, GOUT_SIGNAL(0), COEF_SIGNAL_1, FIR_OUT(6));
FIR_OUT(1)+FIR_OUT(2)+FIR_OUT(3)+FIR_OUT(4)+FIR_OUT(5)+FIR_OUT(6);

END BEHAV;

A.1.4 User Package
LIBRARY IEEE;
USE IEEE.NUMERIC_STD.ALL;

PACKAGE PKG_SIGNED IS

    TYPE SIG_8 IS ARRAY (NATURAL RANGE <> ) OF Signed (7 DOWNTO 0);
    TYPE SIG_12 IS ARRAY (NATURAL RANGE <> ) OF Signed (11 DOWNTO 0);
    TYPE SIG_16 IS ARRAY (NATURAL RANGE <> ) OF Signed (15 DOWNTO 0);
    TYPE SIG_20 IS ARRAY (NATURAL RANGE <> ) OF Signed (19 DOWNTO 0);
    TYPE SIG_24 IS ARRAY (NATURAL RANGE <> ) OF Signed (23 DOWNTO 0);
    TYPE SIG_28 IS ARRAY (NATURAL RANGE <> ) OF Signed (27 DOWNTO 0);
    TYPE SIG_32 IS ARRAY (NATURAL RANGE <> ) OF Signed (31 DOWNTO 0);

    TYPE SIGS_32 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(31 DOWNTO 0);
    TYPE SIGS_28 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(27 DOWNTO 0);
    TYPE SIGS_24 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(23 DOWNTO 0);
    TYPE SIGS_20 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(19 DOWNTO 0);
    TYPE SIGS_16 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(15 DOWNTO 0);
    TYPE SIGS_12 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(11 DOWNTO 0);

    TYPE SPE_ARR IS ARRAY (NATURAL RANGE <> ) OF SFIXED(19 DOWNTO -5);
    TYPE SPE_ARR_12 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(23 DOWNTO -5);
    TYPE SPE_ARR_16 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(27 DOWNTO -5);
    TYPE SPE_ARR_20 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(31 DOWNTO -5);
    TYPE SPE_ARR_24 IS ARRAY (NATURAL RANGE <> ) OF SFIXED(35 DOWNTO -5);

    END PKG_SIGNED;
Bibliography

[1] D. Bishop, “Fixed point package user’s guide,”


