FPGA Implementation of an Interpolator for PWM applications

Master thesis in Electronic Systems at Linköping University by
Jasko Bajramovic
LITH-ISY-EX--07/4030--SE

Supervisor: Per Löwenborg
Examiner: Per Löwenborg

Linköping, 31 October, 2007
Title  
FPGA Implementation of an Interpolator for PWM applications

Abstract
In this thesis, a multirate realization of an interpolation operation is explored. As one of the requirements for proper functionality of the digital pulse-width modulator, a 16-bit digital input signal is to be upsampled 32 times. To obtain the required oversampling ratio, five separate interpolator stages were designed and implemented. Each interpolator stage performed upsampling by a factor of two followed by an image-rejection lowpass FIR filter. Since, each individual interpolator stage upsamples the input signal by a factor of two, interpolation filters were realized as a half-band FIR filters. This kind of linear-phase FIR filters have a nice property of having every other filter coefficient equal to zero except for the middle one which equals 0.5. By utilizing the half-band FIR filters for the actual realization of the interpolation filters, the overall computational complexity was substantially reduced. In addition, several multirate techniques have been utilized for deriving more efficient interpolator structures. Hence, the impulse response of individual interpolator filters was rewritten into its corresponding polyphase form. This further simplifies the interpolator realization. To eliminate multiplication by 0.5 in one of two polyphase subfilters, the filter gain was deliberately increased by a factor of two. Thus, one polyphase path only contained delay elements. In addition, for the realization of filter multipliers, a multiple constant multiplication, (MCM), algorithm was utilized. The idea behind the MCM algorithm, was to perform multiplication operations as a number of addition operations and appropriate input signal shifts. As a result, less hardware was needed for the actual interpolation chain implementation. For the correct functionality of the interpolator chain, scaling coefficients were introduced into the each interpolation stage. This is done in order to reduce the possibility of overflow. For the scaling process, a safe scaling method was used. The actual quantization noise generated by the interpolator chain was also estimated and appropriate system adjustments were performed.

Keywords  
interpolator, half band FIR filter, MCM, Altera
Abstract

In this thesis, a multirate realization of an interpolation operation is explored. As one of the requirements for proper functionality of the digital pulse-width modulator, a 16-bit digital input signal is to be upsampled 32 times. To obtain the required oversampling ratio, five separate interpolator stages were designed and implemented. Each interpolator stage performed upsampling by a factor of two followed by an image-rejection lowpass FIR filter. Since, each individual interpolator stage upsamples the input signal by a factor of two, interpolation filters were realized as a half-band FIR filters. This kind of linear-phase FIR filters have a nice property of having every other filter coefficient equal to zero except for the middle one which equals 0.5. By utilizing the half-band FIR filters for the actual realization of the interpolation filters, the overall computational complexity was substantially reduced. In addition, several multirate techniques have been utilized for deriving more efficient interpolator structures. Hence, the impulse response of individual interpolator filters was rewritten into its corresponding polyphase form. This further simplifies the interpolator realization. To eliminate multiplication by 0.5 in one of two polyphase subfilters, the filter gain was deliberately increased by a factor of two. Thus, one polyphase path only contained delay elements. In addition, for the realization of filter multipliers, a multiple constant multiplication, \(\text{MCM}\), algorithm was utilized. The idea behind the \(\text{MCM}\) algorithm, was to perform multiplication operations as a number of addition operations and appropriate input signal shifts. As a result, less hardware was needed for the actual interpolation chain implementation. For the correct functionality of the interpolator chain, scaling coefficients were introduced into the each interpolation stage. This is done in order to reduce the possibility of overflow. For the scaling process, a safe scaling method was used. The actual quantization noise generated by the interpolator chain was also estimated and appropriate system adjustments were performed.
I would like to thank Per Löwenborg and Kent Palmquist at ES for their help during this thesis work.
Contents

1 Introduction 1
   1.1 Background ........................................... 2
   1.2 Requirement Specification ............................. 2
   1.3 Overview ............................................... 3

2 Methodology and Simulation Tools 5
   2.1 Introduction ........................................... 5
   2.2 MatLab and its Features ............................... 5
   2.3 Mentor Graphics ....................................... 6
       2.3.1 HDL Designer .................................... 7
       2.3.2 ModelSim .......................................... 7
   2.4 Leonardo Spectrum .................................... 8
   2.5 Design flow ............................................ 8
       2.5.1 Implementation Steps ......................... 8

3 Theory 11
   3.1 Sample Rate Conversion ............................... 11
       3.1.1 Interpolation .................................... 12
       3.1.2 Polyphase Representation ....................... 15
       3.1.3 Half-Band FIR filters ......................... 18
   3.2 Noise in Digital Systems ............................. 21
       3.2.1 Scaling ............................................ 22
       3.2.2 Scaling Methods ................................ 27
       3.2.3 Safe Scaling ..................................... 27
       3.2.4 $L_2$-norm ....................................... 28
       3.2.5 Signal Scaling in Cascode of Digital Filters .. 29
   3.3 Scaling of Multistage Interpolators .................. 31
   3.4 Roundoff Noise in Multistage Interpolator Realization 35
4 Implementation

4.1 Introduction ................................................. 39
4.2 Interpolator design using basic design method ......... 41
4.3 Linear Programming ........................................ 48
4.4 Design Method 3 ............................................. 51
4.5 Chosen Design Method .................................... 56
4.6 Coefficient rounding ....................................... 56
4.7 Scaling ..................................................... 61
4.8 Roundoff Noise Measurement .............................. 63
4.9 MCM algorithm ............................................. 64
4.10 FPGA Implementation ..................................... 66
  4.10.1 Introduction ......................................... 66
  4.10.2 Board Applications ................................. 73
  4.10.3 Audio CODEC Interface ............................. 75
  4.10.4 SRAM memory ...................................... 85

5 Simulation Results ............................................. 89

6 Conclusion

6.1 Final Thoughts ............................................. 93
6.2 Further Work ............................................. 94

Bibliography .................................................. 95
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Interpolator</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Interpolation by factor two</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Polyphase interpolation</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Identity for filter and samples</td>
<td>17</td>
</tr>
<tr>
<td>3.5</td>
<td>Polyphase interpolator</td>
<td>18</td>
</tr>
<tr>
<td>3.6</td>
<td>Half-band FIR filter impulse response</td>
<td>19</td>
</tr>
<tr>
<td>3.7</td>
<td>Half-band FIR filter realization</td>
<td>20</td>
</tr>
<tr>
<td>3.8</td>
<td>Utilizing the symmetry for the half-band FIR filter realization</td>
<td>21</td>
</tr>
<tr>
<td>3.9</td>
<td>Scaling of overflow node</td>
<td>23</td>
</tr>
<tr>
<td>3.10</td>
<td>Illustration of two’s complement arithmetic</td>
<td>24</td>
</tr>
<tr>
<td>3.11</td>
<td>Addition of two numbers when numerical range is unlimited</td>
<td>25</td>
</tr>
<tr>
<td>3.12</td>
<td>Addition of two numbers when two’s complement representation is used</td>
<td>25</td>
</tr>
<tr>
<td>3.13</td>
<td>Multiplication and corresponding shift-and-add realization</td>
<td>26</td>
</tr>
<tr>
<td>3.14</td>
<td>Multiplication with decimal numbers</td>
<td>26</td>
</tr>
<tr>
<td>3.15</td>
<td>Scaling of cascaded FIR filters</td>
<td>30</td>
</tr>
<tr>
<td>3.16</td>
<td>Scaling of cascaded FIR filters</td>
<td>31</td>
</tr>
<tr>
<td>3.17</td>
<td>Interpolation by a value of four</td>
<td>32</td>
</tr>
<tr>
<td>3.18</td>
<td>Polyphase representation</td>
<td>33</td>
</tr>
<tr>
<td>3.19</td>
<td>Polyphase representation</td>
<td>34</td>
</tr>
<tr>
<td>3.20</td>
<td>Roundoff noise model</td>
<td>36</td>
</tr>
<tr>
<td>3.21</td>
<td>Round off noise measurement</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Interpolation chain consisting of five separate interpolator stages, OSR = 32</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Single interpolator stage</td>
<td>42</td>
</tr>
</tbody>
</table>
4.3 Polyphase interpolator. .................................. 42
4.4 Single stage, half-band FIR filter magnitude response, 
    $OSR = 2$. ............................................. 46
4.5 Magnitude response of the interpolation chain, $OSR = 
    32$. .................................................. 47
4.6 Magnitude response of interpolation chain, $OSR = 32$. 55
4.7 Stopband attenuation with respect to changing coefficient 
    word length, $M$. ..................................... 58
4.8 Magnitude response of the interpolation chain, $OSR = 
    32$. .................................................. 60
4.9 Structure for round-off noise simulation, $M$. ............... 63
4.10 Altera DE2 Board. ........................................ 66
4.11 Altera DE2 Board block diagram. ......................... 69
4.12 Interpolator test structure. ................................ 74
4.13 Audio CODEC block diagram. ............................ 76
4.14 Left justified mode. ....................................... 78
4.15 Short .................................................... 79
4.16 Register map. ............................................. 82
4.17 SRAM write cycle. ....................................... 85

5.1 The output from the audio interface. ....................... 90
5.2 The output from the interpolator block. .................... 90
5.3 The amplitude spectrum of the sinus signal from the 
    audio interface. ...................................... 91
5.4 The amplitude spectrum of the sinus signal from the 
    interpolator block. .................................... 91
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Impulse response, (h(n)), of first half-band FIR filter, (H_1(z))</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>Impulse response of (H_1(z)) calculated by using optimization technique.</td>
<td>50</td>
</tr>
<tr>
<td>4.3</td>
<td>Impulse response of first half-band FIR filter, (H_1(z))</td>
<td>52</td>
</tr>
<tr>
<td>4.4</td>
<td>Impulse response of second half-band FIR filter, (H_2(z))</td>
<td>53</td>
</tr>
<tr>
<td>4.5</td>
<td>Impulse response of third half-band FIR filter, (H_3(z))</td>
<td>53</td>
</tr>
<tr>
<td>4.6</td>
<td>Impulse response of fourth half-band FIR filter, (H_4(z))</td>
<td>53</td>
</tr>
<tr>
<td>4.7</td>
<td>Impulse response of fifth half-band FIR filter, (H_5(z))</td>
<td>54</td>
</tr>
<tr>
<td>4.8</td>
<td>Calculated interpolator filter coefficient word lengths</td>
<td>61</td>
</tr>
<tr>
<td>4.9</td>
<td>Calculated values of critical nodes</td>
<td>62</td>
</tr>
<tr>
<td>4.10</td>
<td>The required number of adders for the actual interpolator implementation</td>
<td>65</td>
</tr>
<tr>
<td>4.11</td>
<td>Allocated Audio Codec pins</td>
<td>84</td>
</tr>
<tr>
<td>4.12</td>
<td>SRAM pin description</td>
<td>86</td>
</tr>
<tr>
<td>5.1</td>
<td>Calculated SNR values</td>
<td>92</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Nowadays, there is a requirement in many digital systems for an increase of the sample rate of the signal stream. This is true, as many of modern digital systems are more and more complex, consisting of several DSP processors that operate at different sampling frequencies. For example, each new generation of mobile phones are becoming more complex as they have to incorporate more functions and new features. Thus, in current mobile phones, one can find separate DSP processor for communication, video, photo-camera, music and voice recording. Furthermore, in the audio community three common sample rates are utilized. For broadcast industry a 32 kHz stream rate is needed, for a Compact Disc (CD) media a 44.1 kHz of stream rate is needed and finally for digital audio tapes (DAT) a 48 kHz is needed [9]. If we want to combine/mix signals from these three environments digitally, a common sample rate for all of the signals must be employed. To preserve audio integrity, the stream at the lower sample rate must have its sample rate increased, that is interpolated, in order to match the sample rate of the higher sample rate signals. Furthermore, interpolators can also be found in digital receivers as a part of the timing recovery loop and in oversampled sigma-delta modulators. In sigma-delta modulators, the interpolator is very important part [2]. Here, interpolators are used to obtain a high-resolution signal which is consequently fed to the input of modulator. By the interpolation operation, i.e. oversampling, signal frequency and quantization noise are moved further apart from each other. Oversampling is an essential requirement for proper functionality of sigma-delta modulators.
1.1 Background

This thesis was carried out at the Division of Electronics Systems at Linköping Institute of Technology, as a part of one large research project. The overall project goal is to implement a “Digital CMOS Pulse-Width Modulator for Class-D Power Amplifications”. In this thesis, the multistage interpolator will be designed from the requirement specification down to the working FPGA prototype.

1.2 Requirement Specification

- Oversampling ratio of 32, i.e. $OSR^1 = 32$.
- Multi-stage interpolator realization.
- Half-band filters must be used for realization of image-rejection filters.
- The overall stopband attenuation of the multistage interpolator chain must be equal to $86\,dB$.
- Filter multiplications must be realized as a shift and add operations.
- Utilize MCM$^2$ technique.
- FPGA prototype of designed system.
- System test and testing strategies.
- VLSI implementation definitions.
- High operation speed.
- Small implementation area.

$^1$OSR stands for oversampling ratio and is defined as $OSR = \frac{f_{\text{sample}}}{2f_0}$, where $f_0$ is the highest frequency component if the signal is of frequency $f$.

$^2$MCM stands for Multiple Constant Multiplication.
1.3 Overview

- Low power consumption.

1.3 Overview

In this section, an overview of the thesis is given.

In Chapter 2, a brief description about technical tools that have been used throughout the project, have been given. Furthermore, the approach to the project and chosen design methodology is also discussed.

In Chapter 3, a theoretical background relevant to the successful system implementation is given. The chapter begins with an interpolator description, followed by the theory related to the half-band FIR filters. In the last sections of this chapter, several different noise sources that are present in digital systems are described and discussed. In addition, theory related to safe scaling, $L_2$-scaling and roundoff noise measurement is also given.

In Chapter 4, an implementation of a multistage interpolator stage is given. Initially, system parameters are calculated in Section 4. Here, system requirements presented in Section 1.2 must be fulfilled. Section 4.10 gives the system implementation on FPGA board.

Chapter 5, concludes the thesis. Here, implementation results from Chapter 4 are compared with the given requirement specification in Section 1.2. Suggestions, on what could be done to improve implementation and discussion of further work are also given in this chapter. Furthermore, VLSI implementation definitions are discussed in last parts of chapter.
Chapter 2

Methodology and Simulation Tools

2.1 Introduction

To make it easier to understand discussions in chapters to come, a small introduction to the simulation tools and chosen design methodology must be made. The goal of this chapter is to make the reader familiar with design steps and those tools that are used for realization of given tasks.

This chapter starts with a brief introduction to the simulation tools, where features that are relevant to the project work are described. The last part of the chapter describes the design steps that are used to accomplish a working system prototype on FPGA board.

2.2 MatLab and its Features

MatLab stands for Matrix Laboratory and is a technical computing environment used for high-performance numeric computation and graphical visualization [4]. This is a very powerful tool that is used in many different fields of industry and academic community. The reason behind MatLab popularity is the tools high computational capacity but also its user friendly environment where problems and solutions are expressed just as one writes them down mathematically on a piece of paper. As such MatLab is well suited as a simulation tool in first
steps of the design flow, since the tool is a very powerful digital signal processing tool. *MatLab* functions that are used during multistage interpolator design and simulation can be found in Appendix. The interested reader can read through the *MatLab* help files since the amount of time needed to explain each and every function that were used during design phase would be considerable.

During the first parts of the design flow, the most recent version of *Simulink* was also used. As a part of *MatLab*, *Simulink* is a user-friendly and easy to use, graphical environment with predefined digital signal processing blocks. As an early attempt of integration between the multistage interpolator and the sigma-delta modulator, a bug in *Simulink* was detected. The simulation performed by using predefined *Simulink* blocks, that could be found in the *Simulink* library, gave different results than those obtained in *MatLab*. Thus, we conclude that the current version of *Simulink* was not suitable for digital systems where multiple stages alter the overall sampling frequency.

During the design phase, we also had to use an older version of *MatLab* since the current version did not support the function `foptions`. This function was used for filter coefficient optimization as a part of linear programming.

### 2.3 Mentor Graphics

*Mentor Graphics* delivers a set of electronic design automation, (EDA), tools which are used for synthesis and fast prototyping of digital systems [5].

Since the goal of this project is to design an executable VHDL model of the multistage interpolator with an ultimate goal of achieving a synthesizable model on FPGA board, only design tools for FPGA design are used. Such tools include *HDL Designer* which is the graphical design approach for design creation, the *ModelSim* used for VHDL simulation and debugging and the *LeonardoSpectrum* that is used for final system synthesis.
2.3 Mentor Graphics

The rest of this section will give a brief introduction to the tools mentioned in text above. For an in detail information regarding FPGA design tools, the interested reader can visit the home page of Mentor-Graphic [5].

2.3.1 HDL Designer

This tool is used for creation and management of VHDL designs. It is an 'easy to use' tool in the sense that the user can design in a way he/she is most comfortable with, as the tool offers a graphical interface [6]. Furthermore, HDL Designer offers various textual or graphical editors the users can choose from in order to generate VHDL code. This tool also have a graphical environment that could be used for design review, archival and reuse.

By using HDL Designer, a top-down design flow can be implemented where the model to be designed is realized in a several steps of successive model refinements. One can have a top-model under which a number of sub-blocks exists. Naturally, each block is defined by using the VHDL programming language. Once, subblocks are defined, they are compiled, in order to validate lexically correct VHDL code.

2.3.2 ModelSim

The next step towards a fully functional FPGA prototype, a functional validation of the digital system is performed with simulation. For this the tool ModelSim is used [7]. This tool provides a comprehensive simulation and debug environment for FPGA design. Like the HDL Designer, this tool offers a user friendly graphical interface. Here, all design signals defined in the previous stage, are listed. Thus, giving the possibility to examine if the behavior of designed system, displayed in the wave window of ModelSim, fulfils intended system functionality. Furthermore, the wave window allows users to combine several signals in one buss field, allowing easier design validation. Also, appearance of waveforms can be manipulated which is helpful for both troubleshooting and system grading. ModelSim also gives the
possibility to examine the hierarchy of the designed system. In short, 
ModelSim shortens and facilitates validation of a design model.

2.4 Leonardo Spectrum

Once the system functionality is simulated and verified in ModelSim, 
the final step in system implementation is to synthesize it on to the 
FPGA chip. For this purpose the logic synthesis tool, LeonardoSpectrum is used [7]. The logic synthesis is the process of translating a 
VHDL model into technology specific gate-level description. For this 
project the Altera FPGA board was used. 
The abbreviation, FPGA stands for Field Programmable Gate Arrays 
and constitute a special class of chips which can be programmed at a 
gate by gate level. This results in speed and flexibility without having 
to design a custom chip for a given system specification.

2.5 Design flow

To facilitate the project work from actual system idea down to the final 
verified and fully operational system prototype on an FPGA board, a 
good and effective design flow must be used. Chosen design flow will 
influence the entire project work and determine if the final system will 
be successfully implemented or not. Thus, appropriate design flow had 
to be selected.

For this project a top-down design flow has been chosen [8]. Such 
design flow starts with the high-level system description with specified requirements and ends up with a functional gate level system implementation. In-between design steps some detours have been taken as means to increase the understanding of the relationship between different design parameters.

2.5.1 Implementation Steps

The implementation process towards a fully functional system can be seen as a series of steps. These steps are described in text below.
2.5 Design flow

Step 1 - Literature Study

This initial step is very important at the beginning of the design since it gives the foundation and basis for understanding different design aspects that influence the system realization. Here, a large amount of information relevant to the system implementation is collected. Since broad knowledge is required for realization of multistage interpolators, considerable amount of technical papers have been collected. Papers that are relevant for this project can be found in Bibliography chapter.

Step 2 - High Level Implementation

The next implementation step that follows is the high level modeling of the multistage interpolator. Here, different system parameters are determined and calculated. Such parameters include estimation of the required number of interpolator stages, simulation of several different system implementations, calculation of the impulse response of each image rejection filter in the multistage interpolator realization, scaling coefficient estimation, estimation of required data word length for filter coefficients and measurement and estimation of generated roundoff noise. All simulations in this design step were performed in MatLab. The reason behind the use of MatLab in this initial step is that it is a powerful tool for implementing digital signal processing systems.

This implementation step is very important for the success of overall system design since crucial system parameters are calculated, such as filter coefficients, number of interpolator stages, etc. Stated in another way, erroneously estimated parameters would negatively influence the final results. Thus, large amount of working hours was put during this step.

Step 3 - Gate Level Implementation

As the system parameters have been determined, the next step towards a fully functional multistage interpolator prototype, was the system implementation on the FPGA board. For this purpose an Altera FPGA Board was used.
Methodology and Simulation Tools

Here, several circuit optimizations were performed with an objective to decrease the overall system power consumption but also to have satisfactory system throughput and speed. Thus, a high level MatLab model was implemented with lowest possible hardware utilization. For this, an MCM algorithm was used to realize all multiplication operations as 'shift and add operations'. To increase system throughput, the optimization technique pipelining was also used. The main objective, was to decrease propagation delay through the circuit as to meet the timing requirements.
Chapter 3

Theory

In the section that follows, theory needed for realization of multistage interpolator will be discussed. A brief introduction, with examples, to multistage interpolators, scaling and round-off noise is given.

3.1 Sample Rate Conversion

In present days, multirate techniques are used in many digital signal processing systems. Such systems are called multirate systems. The area of multirate digital signal processing is basically considered with problems in which more than one sampling rate is required in the digital system. By using multirate techniques, the effective sampling rate of the discrete signal is changed after the signal has been digitized. Thus, sampling rate conversion has many applications. For example, sample rate conversion is mandatory in real-time precessing, when two separate hardware processors operating at different sample rates must exchange digital information. Multirate techniques are also used in the modern telecommunication field, where digital transmission systems are required to handle data at different samplings rates, i.e. video, low-bit rate speech. Furthermore, sample rate conversions are also used to reduce the computational complexity of certain narrow band digital filters. As such, their hardware implementation will be cheaper [9] [1].

There are two fundamental processes in multirate systems. The process of increasing the sampling rate of a signal is called interpolat-
tion and similarly, the process of reducing the sample rate of a signal is called decimation. In this project only the interpolation operation is used for an increase of the sample rate. Therefore, in the following text only theory related to interpolation will be discussed.

3.1.1 Interpolation

The process of increasing the sample rate of a discrete signal $x(n)$ is called interpolation. The goal of the interpolation operation is to obtain a new digital sequence with higher sampling rate than the original sequence. Naturally, the resulting sequence must contain the same information as the original one. Main operations performed by an interpolator block are upsampling, followed by an image-rejection lowpass filter. One such combination is shown in Fig. 3.1.

![Figure 3.1: Interpolator.](image)

The upsampler is used to increase the sampling rate of a discrete signal $x_{\text{old}}(n)$ by some factor $L$. This is done by placing the $L - 1$ equally spaced zeros between each pair of original samples. The resulting signal $x_{\text{new}}(m)$ is given by

$$x_{\text{new}}(m) = \begin{cases} x_{\text{old}}\left(\frac{m}{L}\right), & m = 0, \pm L, \pm 2L, \ldots \\ 0, & \text{otherwise} \end{cases} \quad (3.1)$$

The resulting sampling period for the new digital sequence is

$$T_{\text{new}} = \frac{T_{\text{old}}}{L} \quad (3.2)$$

and the new sampling frequency is

$$f_{\text{new}} = Lf_{\text{old}} \quad (3.3)$$

Thus, the Fourier transform of $x_{\text{new}}(m)$ and its corresponding $z$-transform are
3.1 Sample Rate Conversion

\[ X_{\text{new}}(e^{j\omega T_{\text{new}}}) = X_{\text{old}}(e^{jL\omega T_{\text{new}}}); \quad X_{\text{new}}(z) = X_{\text{old}}(z^L), \]  
(3.4)
respectively.

By upsampling operation, the frequency spectrum of \( X_{\text{new}} \) contains not only the information baseband, i.e. \(-\pi/L\) to \(\pi/L\), but also images of the baseband centered at harmonics of the original sampling frequency, i.e. \(\pm 2\pi/L\), \(\pm 4\pi/L\), \(\pm 8\pi/L\) \ldots\) As a result, these repeated images must be filtered out. Thus, the upsampled signal \( x_{\text{new}}(m) \) must be filtered with a digital lowpass filter. This lowpass filter is called image-rejection filter.

The ideal frequency response is calculated as

\[ H(e^{j\omega_{\text{new}}}) = \begin{cases} G, & |\omega_{\text{new}}| \leq \omega_{\text{new}}T = \frac{\pi}{L} \\ 0, & \text{otherwise} \end{cases} \]  
(3.5)
where the gain, \( G \), in passband should be equal to \( L \) \cite{9} \cite{10}.

The interpolation process will be illustrated through a simple example. Assume that we have digital sequence \( x_{\text{old}}(n) \) as shown in Fig. 3.2 (a), and we want to increase the sample rate by a factor of \( L = 2 \).
The frequency spectrum of $x_{\text{old}}(n)$ sequence is provided on the right side of Fig. 3.2. Here, only the signal spectrum between 0 and $3f_{\text{old}}$ is shown. In order to upsample $x_{\text{old}}(n)$, a single zero, $(L - 1)$, is inserted between each original sample values. Consequently, the new sequence $x_{\text{int}}(m)$ is created. This is shown in Fig. 3.2 (b). Here, $x_{\text{int}}(m) = x_{\text{old}}(n)$ when $m = 2n_{\text{old}}$. That is, the old sequence is now embedded in new sequence and can be located at every second sample time instance, i.e. $x_{\text{old}}(n_{\text{old}}) = x_{\text{int}}(nLt_{\text{sample}})$, where $L = 2$ and $n = 0, 1, \ldots$.

The frequency spectrum of $x_{\text{int}}(m)$, i.e. $X_{\text{int}}(m)$, is shown on the right side of Fig. 3.2 (b), where $f_{\text{new}} = 2f_{\text{old}}$. The solid curves without dashed box around them in $X_{\text{int}}(m)$ are called for images. The final step in interpolation process will be to filter out $x_{\text{int}}(m)$ sequence with a lowpass digital filter and by that attenuate the unwanted spectral images. The frequency response of lowpass filter is shown as the
3.1 Sample Rate Conversion

dashed box at 0Hz and \( f_{\text{new}} \) in Fig. 3.2 (c). This lowpass filter is called for an interpolation filter, and its output sequence is the desired \( x_{\text{new}}(n) \) having the corresponding frequency spectrum \( X_{\text{new}}(m) \), as shown in Fig. 3.2 (c).

3.1.2 Polyphase Representation

As shown earlier in Fig. 3.2 (b), the input signal contains a number of zero values when interpolated. This feature of the interpolation process can be exploited to reduce the computational workload of the interpolator, since it is unnecessary to perform arithmetic operations involving zero values. As a result of this, in practice, an interpolator is realized in its polyphase structure. The corresponding interpolator structure is called for Polyphase interpolator.

Usually, for the realization of an interpolation filter, FIR filters are used. Since, FIR filters have finite impulse response length, they can easily be decomposed into their corresponding polyphase structures. Thus, by using the polyphase representation the transfer function \( H(z) \) of any FIR filter can be written as

\[
H(z) = \sum_{k=0}^{L-1} z^{-k}H_kz^L = [Nz^{-1} \ldots z^{-(M-1)}] \begin{bmatrix} H_0(z^L) \\ H_1(z^L) \\ H_2(z^L) \\ \vdots \\ H_{M-1}(z^L) \end{bmatrix} \tag{3.6}
\]

where the right hand side of Eq. (3.6) is called polyphase representation [11].

Thus, depending on the upsampling factor, \( L \), the resulting polyphase filter realization will have \( L \) sub-filters. This is best illustrated through an example.

Assume that we have a 12-tap FIR-filter as illustrated in Eq. (3.7) on the next page and in addition assume that the interpolator is interpolating by a factor four.
\[ H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} \]
\[ + h(4)z^{-4} + h(5)z^{-5} + h(6)z^{-6} + h(7)z^{-7} \]
\[ + h(8)z^{-8} + h(9)z^{-9} + h(10)z^{-10} + h(11)z^{-11} \]  
\[ (3.7) \]

since \( L = 4 \) and using the relation, \( h_k(n) = h(k + Ln), 0 \leq k \leq L - 1 \), we obtain

\[ H(z) = \underbrace{h_0 + h(4)z^{-4} + h(8)z^{-8}}_{H_{00}(z)} \]
\[ + z^{-1} \underbrace{[h(1) + h(5)z^{-4} + h(9)z^{-8}]}_{H_{01}(z)} \]
\[ + z^{-2} \underbrace{[h(2) + h(6)z^{-4} + h(10)z^{-8}]}_{H_{02}(z)} \]
\[ + z^{-3} \underbrace{[h(3) + h(7)z^{-4} + h(11)z^{-8}]}_{H_{03}(z)} \]  
\[ (3.8) \]

This is also illustrated in Fig. 3.3.
3.1 Sample Rate Conversion

![Diagram of sample rate conversion](image)

**Figure 3.3:** Polyphase interpolation.

Furthermore, by using Novel Identity in Fig. 3.4, further simplification of the interpolator structure is possible [1] [9] [10].

![Diagram of identity for filter and samples](image)

**Figure 3.4:** Identity for filter and samples.

The advantage with this new structure is that the sampling frequency in each branch is lower than in the original structure. This is the case since the sample rate is increased after the lowpass filtering.
In the figure above we can see that the output of the polyphase interpolator is realized as a rotating switch. This switch is called *commutator* and is used to rotate through four positions illustrated in Fig. 3.5. Thus, the commutator applies four $x_{new}(m)$ output samples to the following interpolator stage. The reason for using a *commutator* is the observation that after the upsampler in Fig. 3.3, three succeeding sample values are equal to zero and the lower polyphase branches have a delay of one, two and three sample periods. Consequently, at each time instant at the output of polyphase interpolator, only one of four polyphase branches produces a non-zero sample value. Thus, for each input value, four output values are generated as the sampling rate at the output of the interpolator is four times higher compared to the input sample rate.

#### 3.1.3 Half-Band FIR filters

To achieve further hardware simplifications, half-band filters can be used for realization of *image-rejection* filters. They are a special kind of FIR filters that have the advantageous property that the impulse response has every other filter coefficient equal to zero, expect for the middle one. This good property enables us to avoid approximately half the number of multiplication needed for implementation of the filter. Thus, less hardware is needed for the actual filter implementation.
3.1 Sample Rate Conversion

For the half-band FIR filter, the frequency response is symmetric around $f_{\text{sample}}/4$. Further, the sum of the passband edge, $f_{\text{pass}}$, and the stopband edge, $f_{\text{stop}}$, is equal to $f_{\text{sample}}/2$. In addition, the stopband, $\delta_{\text{stop}}$, and passband, $\delta_{\text{pass}}$, ripples must be equal or otherwise the filter symmetry will be lost.

Just to illustrate a half-band FIR filter realization, an example is presented in the following text. Figure 3.6 shows the filter coefficients for an 11-tap half-band FIR filter.

![Figure 3.6: Half-band FIR filter impulse response.](image)

The half-band filter impulse response is calculated by using the MatLab function `remezord`. The values of the passband and stopband edges are chosen such that their sum satisfies the equality $f_{\text{stop}} + f_{\text{pass}} = f_{\text{sample}}/2$ and values for passband and stopband ripples are chosen to satisfy the equality, $\delta_{\text{pass}} = \delta_{\text{stop}}$.

In Fig. 3.6 we can see that every other filter coefficient of $h(n)$ is equal to zero. Thus, only 7 multiplications per output sample is performed. Hence, for an $N$-tap half-band FIR filter, only

$$ Number_{\text{Mult}} = \frac{N + 1}{2} + 1 \quad (3.9) $$

multiplications per output sample are performed. The transversal structure of our 11 tap half-band FIR filter is shown in Fig. 3.7, where the $h(1)$, $h(3)$, $h(7)$ and $h(9)$ multipliers are absent.
Furthermore, multipliers in Fig. 3.7, can be implemented by using shifts, adders and subtracters which further simplifies the realization complexity of half-band FIR filters. In addition, the number of adders and subtracters can significantly be reduced by using their partial results. This is explained with the help of small example, in Section 3.2.1 on page 26.

Also, it is possible to simplify the half-band FIR filter realization by utilizing the symmetry of filter coefficients. Thus, further filter simplification is possible where the number of needed multiplication operations is even more reduced. This statement can be observed in Fig. 3.6. Here, we can see that filter coefficients on both sides of the middle tap, have the same numerical values. Thus, Fig. 3.8 illustrates one of several possible filter implementations when filter symmetry is utilized.
3.2 Noise in Digital Systems

As a last step to reduce the computational complexity, for the implementation of polyphase decomposed half-band FIR filters a *Multiple Constant Multiplication* technique, MCM, was used. Here, filter multipliers have been realized as a shift, add and subtract units.

![Figure 3.8: Utilizing the symmetry for the half-band FIR filter realization.](image)

As a last step to reduce the computational complexity, for the implementation of polyphase decomposed half-band FIR filters a *Multiple Constant Multiplication* technique, MCM, was used. Here, filter multipliers have been realized as a shift, add and subtract units.

3.2 Noise in Digital Systems

In practice, both parameters of \(LTI\) \(^1\) discrete system and their signals can only take discrete values. The limitation imposed on signal value representations is mostly due to the limited number of registers that could be found inside digital systems. In ideal case, the digital designer would have an infinite number of registers available for storing results and system parameters for fast calculations, but this is not feasible both from implementation point of view but also from a technological stand point \([1]\).

As an example, assume that we have two unsigned data words, each represented with 16 bits, that will be accumulated in a \(MAC\) \(^2\). For each new iteration, a register file used for storing the results must be increased by \(N_1+N_2-1\), where \(N_1\) and \(N_2\) represents the data word lengths \([3]\). After several iterations, we will be faced with a very large signal value that has to be stored in a register file. Thus, this is

---

\(^1\) LTI stands for Linear Time Invariant  
\(^2\) MAC stands for Multiple And Accumulate unit.
not possible. Therefore, most digital systems are operating using a fixed-point\(^3\) data representation.

In fixed-point digital system implementations, the input-output behavior is not ideal. The quantization of signals and system parameters will introduce unwanted errors and oscillations into the system. To have a satisfactory dynamic signal range at the output, overall system noise must be suppressed or kept below previously determined signal levels.

The noise generated is due to the quantization of arithmetic operations, as both arithmetic operations of multiplication and addition will result in signal values that must be rounded or truncated to an appropriate data word length, i.e. to the number of available registers. Another type of error in digital filters occurs due to nonlinearity caused by quantization of arithmetic operations. Such error could lead to unwanted oscillations at the output of the filter.

Several techniques can be used to suppress or limit generated digital noise to some satisfactory level.

### 3.2.1 Scaling

Scaling is a circuit technique used to prevent overflows in fixed-point arithmetic [1]. Overflows occur when a signal value exceeds the given signal range. As a result, large errors occur at the system output.

To reduce probability of overflow, scaling multipliers have to be introduced into the system. Fig. 3.9 illustrates how scaling coefficients are introduced into the digital network with the purpose of scaling a critical node.
3.2 Noise in Digital Systems

The overflow node \( v(n) \) is scaled by multiplying all input signals of the network \( N_2 \), by an introduced scaling coefficient \( c \). All signals at the network outputs are then multiplied by \( 1/c \). By doing so only the gain from the input of the filter to the critical node is changed without affecting the transfer function of the network. To be sure that the transfer function of the network is unchanged, the scaling coefficient \( c \) is chosen in such a way that the following inequality is true, \( c (1/c) = 1 \). Common values of \( c \) are \( c = 2^{\pm n} \), where \( n \) is some integer value, i.e. scaling coefficients are usually multiple of two as they should be easy to implement in hardware.

Since the filter coefficients and signal values in digital systems can take both positive and negative values, they are usually represented in hardware in two’s complement format. The two’s complement is often the chosen format for integer value representation and is well suitable for hardware implementation of digital systems [17].

The overflow characteristics of the two’s-complement representation is shown in Fig. 3.10.
As seen from Fig. 3.10, the largest number in two’s-complement representation is $1 - Q$, where $Q$ is the quantization step, and the smallest number is $-1$. The value of the quantization step depends on how many bits the signal is represented with. Figure 3.10 shows that if some number $x$ is larger than $1 - Q$, the number will be interpreted as $x - 2$ or if some other number is smaller than $-1$, than the number will be interpreted as $x + 2$.

One of the benefits with two’s-complement representation is that all critical nodes found in a digital system do not have to be scaled during the scaling procedure. The nodes which have to be scaled are inputs to all multiplications with a non-integer coefficient and filter outputs. The reason why additions and multiplications with integer factors do not have to be scaled is that temporary overflows in additions can be accepted if the final result is within the proper signal range, i.e. between $1 - Q$ and $-1$, and multiplications with integer coefficients can be interpreted as a chain of repeated additions.

To clarify the statements above, Fig. 3.11 and Fig. 3.12 depict the situation in which the output value of an addition network is valid despite the fact that an intermediate value has exceeded the valid numerical range, which for this example is set to the $[-1 1]$. 
3.2 Noise in Digital Systems

As seen in Fig. 3.11, when the numerical range used for signal representation is not limited to some fixed range, there will be no overflows in the addition network, since both the intermediate result and the output value is within the valid numerical range.

This also holds for the case when the chosen signal representation is two's complement. Here, addition with same numbers as that in Fig. 3.11 above is performed. The corresponding results are shown in Fig. 3.12. For this example, the two’s complement arithmetic previously depicted in Fig. 3.10 is used. Thus, signal values larger than $1 - Q$ are represented as $x - 2$, so the intermediate value of the adder tree, results in $-0.5$, which is within the valid numerical range which is set to $[-1 1]$ for two’s complement representation. Thus, the output is within the signal range and there is no need for scaling coefficient introduction.

In previous section we have shown that the arithmetic operation addition does not have to be scaled if the final result of the addition is within the valid numerical range. This nice feature of addition operations can be put to use for realization of multiplier where multiplication operation is performed by some integer value. The idea

---

Figure 3.11: Addition of two numbers when numerical range is unlimited.

Figure 3.12: Addition of two numbers when two's complement representation is used.
is to perform a multiplication as a number of addition operations and appropriate input signal shifts. Figure 3.13 illustrates this process.

![Diagram](image)

**Figure 3.13:** Multiplication and corresponding shift-and-add realization.

Here, both the multiplication operation and its corresponding shift-and-add structure are shown. For this example unsigned binary representation is used for signal value representation. Also, sufficiently long data word length is chosen such that overflows never occur.

On the other hand when an input signal value is to be multiplied by some decimal number, the result of the multiplication operation might be erroneous. This is illustrated in Fig. 3.14.

![Diagram](image)

**Figure 3.14:** Multiplication with decimal numbers.

In first case, the result of the multiplication is correct one since we do not use two’s complement for signal representation of signal values. On the contrary, the result of the second multiplier is erroneous despite the fact that the resulting value is within appropriate numerical range,
3.2 Noise in Digital Systems

i.e. between $[-1, 1]$. The value of 1.25 which is larger than 1, in two's complement arithmetic is represented as $1.25 - 2$, that is as $-0.75$. This is the case since we are using two's complement arithmetic. This overflow at the input of the multiplier will result in incorrect output result which is an unwanted behavior in digital systems. Therefore, critical nodes must be scaled and kept within a valid numerical range which for two’s complement is between $1 - Q$ and $-1$.

3.2.2 Scaling Methods

Methods that can be used for scaling of critical nodes in digital systems are several. They include safe scaling and scaling with some probability for overflow. The idea behind each of these methods is to lower the signal gain after an addition which has a high probability of overflow. In following section a brief presentation to different scaling methods that are used for realization of the multistage interpolator, is given.

3.2.3 Safe Scaling

For the safe scaling method, overflows will never occur under normal operation. Here, normal operation conditions include conditions where no external disturbance, supply line disturbance or disturbance caused by hardware malfunction, is present in the digital system [1]. Such unwanted disturbances will result in abnormal signal values which must be avoided and be suppressed in digital systems. For safe scaling, overflow can never happen since all critical nodes in the digital system are scaled in such a way that their scaled signal values are equal to or less than input signal values. Thus, overflows will occur only if the input to the filter overflows.

Unfortunately, this is a rather pessimistic scaling method since signal precision, i.e. signal dynamic range is lost. This influences the overall SNR$^4$ negatively, as the signal dynamic range is decreased, i.e. allowable signal swings, as seen from analog point of view are lower at the critical nodes. Thus, large noise sources introduced in to the

$^4$SNR stands for Signal to Noise Ratio.
system will most likely detoriate wanted information signal power. To calculate scaling constants in critical overflow nodes, initially the impulse response from the input of system to the critical node must be calculated. This is given by following relationship:

\[ |v(n)| = |x(n) * x(n - x)| = \sum_{k=0}^{\infty} |h(k)x(n - x)| \leq \sum_{k=0}^{\infty} |h(k)||x(n - k)| \]

\[ \leq M \sum_{k=0}^{\infty} |h(k)| \tag{3.10} \]

where

\[ M \geq |x(n)| \tag{3.11} \]

The \( x(n) \) is input sequence, while \( h(n) \) is the impulse response from the input of the digital system to the critical overflow node. Thus, for safe scaling, values of the scaling multipliers, \( c_i \), are chosen so that the following inequality is valid:

\[ M \sum_{k=0}^{\infty} c_i |h(k)| \leq M \tag{3.12} \]

In practice, actual system scaling by the safe scaling method is done as follows:
First, calculate the impulse response, from the system input to the critical node by using relation 3.12. Then, the system input is multiplied by scaling multiplier \( c \) given by

\[ c = \frac{1}{\sum_{k=0}^{\infty} |h(k)|} \tag{3.13} \]

as to reduce the risk of overflow at the critical node. The output is then multiplied with the inverse of \( c \), i.e. \( 1/c \).

3.2.4 \( L_2 \)-norm
Another scaling method is the \( L_2 \)-norm. This scaling method is one of three different scaling cases of the \( L_p \)-norms that are based on the
3.2 Noise in Digital Systems

frequency properties of the signal. The $L_p$-norms exploit the knowledge of how the input signal spectrum varies with frequency as one introduce scaling multipliers inside the digital system.

The $L_2$-norm does not guarantee that overflow never occurs inside the system. However, this scaling method is not so pessimistic as the safe scaling since the $L_2$-norm value of some deterministic signal $x(n)$ is the root-mean-squared (rms) value of the signal. Thus, this scaling method is also well suited for scaling of white-noise input signals as the method ensures that the variance at the critical node equals that of the input. The $L_2$-norm value is calculated as follows:

$$\|H(e^{j\omega T})\|_2 = \sqrt{\frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\omega T})|^2 d\omega T}$$

where $H(e^{j\omega T})$ is the frequency response from the input of the filter to the critical node. Here, we assume impulse sequence as the input signal. The scaling coefficients for $L_2$-norm are always chosen so that the resulting values are smaller than 1. By using Parseval’s relation the $L_2$-norm can be written as:

$$\|H(e^{j\omega T})\| = \sqrt{\sum_{k=0}^{\infty} h(n)^2}$$

where $h(n)$ is the impulse response from the input of filter to the critical scaling node. The scaling by $L_2$-norm is done as follows: First, calculate the $L_2$-norm, from the system input to the critical node by using Parseval’s relation, Eq. 3.15. Then, the system input is multiplied by the scaling multiplier in order to reduce the risk of overflow at the critical node.

3.2.5 Signal Scaling in Cascode of Digital Filters

In practice, interpolation filtering by digital filters with high order, say $N = 120$, are often done as a cascode of digital filters that have smaller orders than the original one. Such filter realization is possible since the transfer function of the original filter can usually be factorized. Arguments in favor of such filter realization is a high degree of design
freedom such that several digital optimization techniques can be put into the use. For example, one might use half-band filters for filter realization where every second tap is equal to zero except for middle one\(^5\). Pipelining technique can also be used where registers are introduced between filter stages leading to increased throughput of overall system. One can also take advantage of half-band filters symmetry and realize only half of filter multiplications and the interleaving technique can also be used. As a result, the hardware cost for the filter implementation compared to the original filter is considerably lower.

One such filter realization is illustrated in Fig. 3.15. Here, the original filter is factorized by a factor of three such that the original filter is implemented as a cascode of three partial filters that have lower orders than the original one. When combined, they result in same frequency response as the original filter.

As stated in the previous section, the overflows inside and at the output of the filter must be eliminated or kept below some predefined value. Thus, scaling constants must be inserted into the filter inputs. Each individual filter has a separate scaling multiplier that is determined by some of the previously explained scaling methods. Consequently, to maintain correct behavior of the multistage filter realization each filter output must be multiplied by the inverse of the calculated scaling multiplier \(c\) as seen in Fig. 3.15.

\[ x(n) \rightarrow H_1(z) \rightarrow H_2(z) \rightarrow H_3(z) \rightarrow y(n) \]

**Figure 3.15:** Scaling of cascaded FIR filters.

The complexity of the cascade of digital filters can further be reduced if they are FIR and realized in direct form structure [1]. In such a filter structure, the filter output is not fed back to the inputs. Thus, scaling coefficients are only introduced at the input and output of the filter. This of course, limits the need for scaling constants at

\(^5\) This of course depends on factorizing factor.
3.3 Scaling of Multistage Interpolators

the output of the filter since they could be propagated and integrated in the scaling constant calculation of adjacent filter stages. This idea is illustrated in figure 3.16 below:

\[ x(n) \rightarrow c_1 H_1(z) \rightarrow c_2 H_2(z) \rightarrow c_3 H_3(z) \rightarrow y(n) \]

**Figure 3.16:** Scaling of cascaded FIR filters.

Here, the constant \( c_1 \) is used to scale the output of filter \( H_1(z) \). Consequently, during the calculation of the constant \( c_2 \) scaling constant \( c_1 \) is propagated and combined in the final value of \( c_2 \). That is,

\[
c_1 = \frac{1}{\|F_1\|_2} \quad c_2 = \frac{1}{c_1} \frac{1}{\|F_2\|_2} \quad c_3 = \frac{(1/c_1)(1/c_2)}{\|F_3\|_2}
\]

where

\[
F_1 = H_1 \quad F_2 = H_1 H_2 \quad F_3 = H_1 H_2 H_3
\]

3.3 Scaling of Multistage Interpolators

In this project, interpolation operation is performed in multiple steps. Thus, the sampling frequency is increased stepwise. As a result, each interpolator stage in the multistage realization, performs upsampling and filtering separately. To maintain correct behavior of the multistage interpolator, unwanted overflows inside the multistage system must be eliminated. Therefore, the input and outputs of individual interpolator stages must be scaled.

By multistage implementation, scaling process becomes cumbersome as the output of upsampler is no longer Wide-Sense Stationary (WSS) but instead Cyclo-WSS. For the Cyclo-WSS process, the signal has statistical properties that vary cyclically with time, meaning that different samples at the output of one interpolator stage will have
different statistical properties, i.e. different mean and variance values. This of course limits the use of regular scaling methods presented earlier in Section 3.2.2, as they cannot be used for scaling of multistage interpolator. The reason for the limited use of regular scaling methods is the assumption that the upsampler output is Cyclic-WSS and not WSS [13].

The chosen method that is used for scaling of multistage interpolator is based on previously presented scaling method of Section 3.2.5 [13]. Here, each interpolator stage is implemented as a polyphase decomposition structure. As a result, the variance $\sigma^2$ at the output of interpolator is no longer time-varying and periodic with period $L$, where $L$ denotes the upsampling factor. Consequently, the interpolator output becomes Wide-Sence Stationary, WSS, and regular scaling methods as those presented in Section 3.2.2 can be used [13].

The scaling procedure used in this project, is illustrated through an example. As shown in Fig. 3.17, we have two interpolator stages that combined, increase the sampling frequency four times. By using polyphase representation and Novel identities, the transfer function $H_1(z)$ of each individual filter is decomposed to its corresponding polyphase structure [1] [9]. As a result, each branch consists of a single rate filter. Hence, their outputs can be scaled using one of several scaling methods explained earlier in Section 3.2.2.

![Figure 3.17: Interpolation by a value of four.](image)

Assume further that the filters have the following transfer functions:

\[
H_1(z) = 1 + z^{-1} + z^{-2} + z^{-3} \quad H_2(z) = 1 + z^{-1} + z^{-2} + z^{-3}
\]

(3.18)

As a first step, the upsampler and filter, $H_1(z)$, are rewritten by
3.3 Scaling of Multistage Interpolators

using polyphase representation, to its corresponding polyphase structures, $F_{10}(z)$ and $F_{11}(z)$. This is illustrated in Fig. 3.18.

Actual mathematical calculation is:

$$H_1(z) = 1 + z^{-1} + z^{-2} + z^{-3}$$
$$= (1 + z^{-2}) + z^{-1}(1 + z^{-2})$$
$$= \left(1 + z^{-2}\right) + z^{-1} \left(1 + z^{-2}\right)$$

$$= \frac{F_{10}(z)}{S_{10}(z)} + \frac{z^{-1}}{S_{11}(z)} \left(1 + z^{-2}\right)$$

Now, using the safe scaling method, corresponding values at the output of the polyphase branches are:

$$S^6(F_{10}(z)) = 2 \quad S(F_{11}(z)) = 2$$

Consequently, the scaling constant $c_1$ is calculated as

$$c_1 = \frac{1}{\max\{S(F_{10}), S(F_{11})\}} = \frac{1}{2}$$

In the final step, using the Novel Identities, filter $H_1(z)$ and upsampler by two will switch places, as illustrated in Fig. 3.19.

6S is annotation for Safe Scaling
In the same manner as previously, we will divide the upsampler by four and \( F_2(z) \) into four polyphase branches as in Fig. 3.19. Here, \( F_2(z) = H_1(z^2)H_2(z) \). The transfer function for each interpolator becomes

\[
H_1(z^2) = 1 + z^{-2} + z^{-4} + z^{-6} \quad H_2(z) = 1 + z^{-1} + z^{-2} + z^{-3}
\]

Thus,

\[
F_2(z) = (1 + z^{-2} + z^{-4} + z^{-6})(1 + z^{-1} + z^{-2} + z^{-3}) = (1 + 2z^{-4} + z^{-8}) + z^{-1} \left( 1 + \frac{z^{-1}}{F_{21}(z)} \right)
\]

\[
F_{20}(z) + z^{-2} \left( 2 + 2z^{-4} \right) + z^{-3} \left( 2 + 2z^{-4} \right)
\]

and safe scaling values at the output of individual filter are

\[
S(F_{20}(z)) = 4 \quad S(F_{21}(z)) = 4 \quad S(F_{22}(z)) = 4 \quad S(F_{23}(z)) = 4
\]
Finally, scaling constant $c_2$ is calculated to be

$$c_2 = \frac{1}{\max\{S(F_{20}), S(F_{21}), S(F_{22}), S(F_{23})\}} = \frac{1}{2}$$  \hspace{1cm} (3.24)

### 3.4 Roundoff Noise in Multistage Interpolator Realization

As stated previously in Section 3.2, fixed-point arithmetic is usually used for implementation of arithmetic operations of digital systems. We observed that if the result of an arithmetic operation is too large compared with available number of registers that are assigned to save that result, an overflow happens. We also saw that multiplications and additions that are used in fixed-point arithmetic with appropriately adjusted signal levels generally do not cause overflow errors. Instead unwanted errors often occur when the result of arithmetic operation is rounded or truncated to an $n$-bit binary number. These errors manifest themselves as unwanted noise at the output of the filter that correlates and destroys wanted signal power.

Both, rounding and truncation, are known under the name of quantization. The rounding operation causes less noise, but instead requires more complex circuit [1]. When truncation operation is used, the least significant bits are removed. As a result more noise is added to the system.

The quantization noise is oftenly modeled as a linear additive error and can be written as

$$y_Q = ax_Q + e(n)$$  \hspace{1cm} (3.25)

where $e(n)$ annotates additive error and $x_Q$ is quantized signal value and $a$ is the gain. Figure 3.20 shows corresponding noise model.
Here, a white noise source $e(n)$ is added to the output of the multiplication element. The white noise $e(n)$ represents the quantization error of the product rounding.

The noise power, i.e. the power of $e(n)$ at the output of the multiplication element is equal to the variance, $\sigma_e^2$ of the $e(n)$ which is given by

$$\sigma_e^2 = kQ^2 \quad (3.26)$$

where $Q$ is the quantization step at the output of the multiplication element and $k = \frac{1}{12}$. The quantization step is defined as

$$Q = 2^{-(B-1)} \quad (3.27)$$

Now, after some manipulations, the noise variance can be written as

$$\sigma_e^2 = k2^{-2(B-1)} = \frac{k2^{-2B}}{4} \quad (3.28)$$

As seen from Eq. (3.28), the value of the noise variance depends on how many bits the data word is represented with. More bits result in less noise, less bits result in more noise. This can be observed from Eq. (3.27). The use of more bits for data representation has also its disadvantages. One of those is that more hardware must be used for system implementation.

The total variance, $\sigma_e^2$, of the roundoff noise at the output of the digital filter with more than one noise source is equal to the sum of the variance contributions from all noise sources [1].
3.4 Roundoff Noise in Multistage Interpolator Realization

\[ \sigma_{tot}^2 = \sum_i \sigma_i^2 \left( \sum_{n=0}^{\infty} g_i^2(n) \right) = \sum_i \sigma_i^2 \| G_i \|_2^2 \]  

(3.29)

Quantization noise can be measured according to the scheme shown in Fig. 3.21.

![Figure 3.21: Round off noise measurement.](image)

Both systems are driven by the same input signal. Usually, a white noise source with zero mean, \( \mu \), and variance equal to 1, \( \sigma^2 = 1 \), is used. Furthermore, both systems have the same filter coefficients that are quantizes to same word length. One system, \( H_{\text{ideal}}(z) \) is chosen to have infinite data word length while other, \( H_{\text{Quant}} \) has a finite word length. The difference between these two systems gives a measure of the generated roundoff noise.
Chapter 4

Implementation

In this chapter the realization of a multistage interpolator will be presented. Initially, a high level MatLab implementation will be discussed followed by the system gate level implementation.

4.1 Introduction

As a requirement for a proper functionality of pulse width modulator, multistage interpolator with an oversampling ratio of 32, was modelled in a sequence of refinement steps. The main idea behind multistage interpolator realization is to perform the interpolation operation in a sequence of smaller interpolation steps. Such interpolation approach will result in a minimization of the overall computational workload with reduced power consumption as final outcome. For these reasons it is advantageous to perform the interpolation operation in a sequence of steps since considerable power savings can be achieved.

This definitely is true when a discrete input signal is upsampled 32 times. In this case, the interpolation operation can be performed in a chain of five successive interpolator stages where each stage upsamples the input signal by factor of 2. Thus, obtaining an OSR$^1$ of 32 times. That is $2^5 = 32$, where 2 is the upsampling factor, $L$, and 5 is the number of interpolator stages, $n$. The corresponding interpolation chain is illustrated in Fig. 4.1.

---

$^1$OSR stand for oversampling ratio
By performing interpolation by two, image-rejection filters that are found in the interpolation chain can be realized as a half-band FIR filters. From the previous chapter, we saw that a half-band filter is a type of FIR filter, where transition region is centered at one quarter of the sampling rate, that is at \( f_s/4 \), where \( f_s \) stands for sampling frequency. Both the end of the passband and the beginning of the stopband are equally spaced on either side of \( f_s/4 \). Furthermore, we saw that the main advantage with using half-band FIR filters is that every other filter coefficient value is equal to zero except for the middle one which is equal to 0.5 [10].

This nice feature of half-band FIR filters is utilized for realization of image-rejection lowpass filters. Naturally, one should not forget that the values of the stopband and passband ripples should be identical, which limits the degree of freedom in the design. As a result of this limitation the required filter order is usually increased. Still, the half-band FIR filter gives the best possible solution for the implementation of image-rejection filters when low hardware utilization and power savings are the main goals to be achieved.

In addition, from the previous chapter we also observed that the largest computational workload performed in any FIR filter is done by the multiplier units. In digital systems, multiplication operation is realized as the combination of adders and logic AND operations [2]. Thus, by minimizing the number of multipliers used for realiza-

\[ \text{Figure 4.1: Interpolation chain consisting of five separate interpolator stages, OSR = 32.} \]
4.2 Interpolator design using basic design method

In this project, the digital signal had to be upsampled by factor of 32. To obtain the best possible implementation, the required upsampling...
Implementation

factor was achieved by performing interpolation operation in five separate stages, where each individual interpolator stage is performing upsampling by a factor of 2 followed by an image-rejection lowpass filter. One such interpolator stage is shown in Fig. 4.2.

\[
x(n) \xrightarrow{\uparrow 2} H_1(z) \xrightarrow{} y(m)
\]

**Figure 4.2:** Single interpolator stage.

To obtain a more simplified interpolator structure, two multirate techniques, Novel Identities and polyphase representations are also used [1] [9] [10]. By utilizing polyphase representation, the image-rejection FIR filter is divided into two subfilters, \(H_{10}\) and \(H_{11}\).

By utilizing the second multirate technique, i.e. Novel identities, image-rejection subfilters and upsampler blocks will switch places. This will result in an additional reduction of computational workload, since the actual subfilters \(H_{10}(z)\) and \(H_{11}(z)\), will operate at a lower sampling rate than the subsequent stages. As a result, the number of operations that must be performed by an interpolator is the same as in the straightforward realization shown in Fig. 4.2, but the number of operations per second is reduced by a factor of \(L\) [1] [9]. This is illustrated in Fig. 4.3.

\[
x(n) \xrightarrow{f_{\text{sample}}} H_{10}(z) \xrightarrow{} y(2n) \\
H_{11}(z) \xrightarrow{} y(2n+1)
\]

\[
y(m) \xleftarrow{2f_{\text{sample}}} H_{10}(z) \xleftarrow{} y(2n) \\
\]

**Figure 4.3:** Polyphase interpolator.

What we can see in the Fig 4.3 is that initially, the input signal is filtered by two polyphase subfilters and consequently the signal
4.2 Interpolator design using basic design method

is upsampled by a factor of two. For the first lowpass filter in the interpolator chain in Fig. 4.1, the values of the passband edge $\omega_T$ and stopband edge $\omega_S$, are set to $\pi/2 - \alpha$ and $\pi/2 + \alpha$, respectively. Here, the constant $\alpha$ has to be smaller than $0.5\pi$. Here, the value of $0.10\pi$ is chosen [1] [10].

By choosing values of passband- and stopband\(^3\) edges as those presented previously, the first constraint for half-band FIR filter is fulfilled. It is easily shown that $\omega_T + \omega_S = \pi$, i.e. $\pi/2 - \alpha + \pi/2 + \alpha = \pi$.

Furthermore, since the discrete signal is upsampled by factor of 2, the actual sampling frequency is also increased by two and the signal spectrum is moved to the lower parts of the frequency axis. Hence, after the first interpolator stage, information spectrum is located at $[f_{signal}/f_{sample\text{new}}]2\pi$. Here, $f_{sample\text{new}} = 2f_{sample\text{Nyquist}}$ where $f_{sample\text{Nyquist}} = 2f_{signal}$. Thus, the oversampling ratio, OSR, after the first interpolator stage becomes equal to 2, i.e.

$$OSR = \frac{f_{sample\text{new}}}{f_{sample\text{Nyquist}}} = 2$$

Consequently, the information spectrum will be compressed further as the sampling frequency increases by a factor of two after each new introduced interpolator stage. More precisely, signal spectrum will be located at $[f_{signal}/(4f_{sample\text{Nyquist}})]2\pi$, $[f_{signal}/(8f_{sample\text{Nyquist}})]2\pi$, $[f_{signal}/(16f_{sample\text{Nyquist}})]2\pi$, ... after each new interpolator stage.

Hence, after the first interpolator stage, signal spectrum will be located at $[f_{signal}/(2f_{sample\text{Nyquist}})]2\pi$ and after the fifth and final stage signal spectrum will be located at $[f_{signal}/(32f_{sample\text{Nyquist}})]2\pi$. Thus, obtaining an oversampling ratio of 32. Of course, this assumption is solely correct if each stage interpolates the discrete signal by a factor of 2.

\(^3\)See Section 3.1.3.
In accordance to previous statements, to satisfy the half-band FIR filter design specification of the first interpolator stage, the required filter order was calculated to $N = 52$. The actual estimation of filter impulse response was performed by using MatLab’s function `remezord`.

Since the half-band FIR filter is used for the actual realization of image-rejection filters, the resulting lowpass FIR filter has every other coefficient value equal to zero except for middle one. This is illustrated in Table 4.1 on the next page.

From impulse response summarized in table we can observe that the filter gain is actually increased by a factor of 2, since the middle tap is not equal to 0.5 as it should be for half-band FIR filters. Instead this value is equal to $1^4$. The filter gain was deliberately increased by a factor of 2 in order to eliminate multiplication by 0.5. Consequently when realizing the actual filter, one polyphase branch will contain only delay units as multiplication by 1 do not influence the final result. Hence, the multiplication by 1 will not be realized in hardware. Naturally, we should not forget that round-off noise and scaling estimation will be influenced by filter gain. Depending on the value of the gain, the data word length must be changed accordingly.

In addition, as the FIR filters are not recursive as it is the case with IIR filters, it is always easy to rewrite their transfer functions into corresponding polyphase form. This was previously stated in Section 3.1.2. Here, half of filter coefficients are used for realization of one subfilter $H_{10}(z)$, while the remaining set of filter coefficients are used for realization of the second subfilter $H_{11}(z)$. This was illustrated previously in Fig. 4.3.

Thus, the impulse response $h(n)$ depicted in Table 4.1, is polyphase decomposed into two separate filter banks. Values of $h(2n)$ are used for the subfilter $H_{10}(z)$ and $h(2n + 1)$ are used for the subfilter $H_{11}(z)$ realization. Here, $n$ is equal to $n = \text{length}(h)$.

---

\footnote{Observe values on left side of Table 4.1. The middle value is not 0.5 but is instead this value is equal to 1.}
4.2 Interpolator design using basic design method

<table>
<thead>
<tr>
<th>Impulse response, $H_1(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0) = 0$</td>
</tr>
<tr>
<td>$h(2) = 0$</td>
</tr>
<tr>
<td>$h(4) = 0$</td>
</tr>
<tr>
<td>$h(6) = 0$</td>
</tr>
<tr>
<td>$h(8) = 0$</td>
</tr>
<tr>
<td>$h(10) = 0$</td>
</tr>
<tr>
<td>$h(12) = 0$</td>
</tr>
<tr>
<td>$h(14) = 0$</td>
</tr>
<tr>
<td>$h(16) = 0$</td>
</tr>
<tr>
<td>$h(18) = 0$</td>
</tr>
<tr>
<td>$h(20) = 0$</td>
</tr>
<tr>
<td>$h(22) = 0$</td>
</tr>
<tr>
<td>$h(24) = 0$</td>
</tr>
<tr>
<td>$h(26) = 1$</td>
</tr>
<tr>
<td>$h(28) = 0$</td>
</tr>
<tr>
<td>$h(30) = 0$</td>
</tr>
<tr>
<td>$h(32) = 0$</td>
</tr>
<tr>
<td>$h(34) = 0$</td>
</tr>
<tr>
<td>$h(36) = 0$</td>
</tr>
<tr>
<td>$h(38) = 0$</td>
</tr>
<tr>
<td>$h(40) = 0$</td>
</tr>
<tr>
<td>$h(42) = 0$</td>
</tr>
<tr>
<td>$h(44) = 0$</td>
</tr>
<tr>
<td>$h(46) = 0$</td>
</tr>
<tr>
<td>$h(49) = 0$</td>
</tr>
<tr>
<td>$h(51) = 0$</td>
</tr>
<tr>
<td>$h(53) = 0$</td>
</tr>
</tbody>
</table>

Table 4.1: Impulse response, $h(n)$, of first the half-band FIR filter, $H_1(z)$.  

45
Implementation

Figure 4.4 depicts the resulting magnitude response of the first interpolator stage where the half-band FIR filter is realized using its polyphase form. As Fig. 4.4 illustrates, the required image- and noise attenuation in the stopband range is achieved. The two vertical lines depict the stopband edge, $\omega_s T$, and the passband edge, $\omega_c T$, respectively. Furthermore, the horizontal line indicates the $86 \, dB$ attenuation that must be achieved in the stopband range.

![Figure 4.4: Single stage, half-band FIR filter magnitude response, $OSR = 2$.](attachment:image.png)

To achieve an oversampling ratio of $OSR = 32$, the lowpass filter designed during the initial interpolator stage realization can be reused for the remaining interpolator stages. This is the case, as succeeding interpolator stages are also performing an interpolation by a factor of two. Logically, the previously designed interpolator stage with upsampler and polyphase half-band FIR filter presented in Table 4.1, can be reused for the remaining four interpolator stages.
4.2 Interpolator design using basic design method

The first image-rejection FIR filter will have stopband- and passband edges located at $\omega T_s = (\pi/2) - \alpha \pi$ and $\omega T_c = (\pi/2) + \alpha \pi$, respectively, where $\alpha \pi = 0.10 \pi$. Thus, $\omega T_s = 0.40 \pi$ and $\omega T_c = 0.60 \pi$. The constraint for half-band FIR filter is fulfilled, i.e. $\omega T_s + \omega T_c = \pi$.

After the fifth and final interpolator stage, the stop- and passband edges are moved toward the lower parts of the $\omega T$ axis, as the sampling rate is increased by a factor of 2 after each consequent interpolator stage. The resulting stopband and passband edges for the entire interpolator chain will be equal to $\omega_s T = [(\pi/2 + \alpha \pi)/16] = 0.11780972$ and $\omega_c T = [(\pi/2 \alpha \pi)/16] = 0.078539816$, respectively. Thus, at the output of the fifth interpolator stage, the sampling rate becomes equal to $f_{\text{sample, new}} = 32 f_{\text{sample, Nyquist}}$, where $f_{\text{Nyquist}} = 2 f_{\text{signal}}$. The oversampling ratio becomes equal to 32 at the output of the fifth interpolator block.

The resulting magnitude response of the five interpolator stages cascaded in the interpolator chain is illustrated in Fig. 4.5.

![Figure 4.5: Magnitude response of the interpolation chain, OSR = 32.](image)
In Fig.4.5, horizontal line annotates the stopband attenuation of 86 dB.

As stated previously, for the actual interpolator realization, five identical interpolator stages are cascaded into a single interpolation chain. Each interpolator stage is built by polyphase half-band FIR filters followed by a communicator unit. Thus, the number of multiplications needed for proper interpolator realization will be given by

\[ \text{Number of Multiplications} = \text{Number of stages} \cdot \left( \frac{N}{2} + 1 \right) \]

Since we have increased the filter gain by multiplying each filter impulse response by 2, the actual number of multiplications needed for realization of the interpolator chain will be equal to:

\[ \text{Number of Multiplications} = \text{Number of stages} \cdot \left( \frac{N}{2} \right) \]

\[ \text{Number of Multiplications} = 5 \cdot \left( \frac{52}{2} \right) = 130 \]

Thus, for actual interpolator chain, 130 multiplication units are required. This is quite large number of multiplication units that are needed. In section that follows we will determine if any hardware improvements can be obtained by using linear programming design approach.

### 4.3 Linear Programming

From the previous chapter we concluded that the best solution for implementation of image-rejection filters when trying to obtain a low hardware cost is to realize them as a half-band FIR filters. Thus, the goal of this section will be to obtain an optimal half-band FIR filter that satisfies a given interpolator specification. For this a linear programming approach will be used [1].
4.3 Linear Programming

The linear programming approach can handle linear constraints among the filter coefficients [1]. Here, the passband or stopband ripples can be kept fixed whereas the stopband or passband ripples are minimized. This means that by using linear programming the passband ripple can be kept fixed whereas the stopband ripple is minimized or vice versa. This will result in a FIR filter with lower filter order than if McClellan-Parks-Rabiner algorithm is used [1]. The actual half-band FIR filter will be realized in a Chebyshev sense. By not having equal passband and stopband ripples, one of the requirements for half-band FIR filters is not fulfilled. That is, the values of stopband and passband ripples are not kept equal.

From previous chapter we know that the stopband and passband ripples for half-band FIR filters should be kept equal. This will not be the case for the actual design method. But since we are interested in actual hardware utilization this design method will be used only for simulation purposes.

The actual filter optimization is done by deliberately changing the values of the passband or stopband ripples so that the given interpolation chain specification is fulfilled. This was performed through several iteration steps that are obtained by running the MatLab script found in Appendix(A). Here, the passband ripple was kept constant while the value of stopband ripple was changed to fulfill the stopband attenuation of 86 dB.

The resulting half-band filter impulse response that best satisfies given filter specification is shown in Table 4.2. Here, the passband ripple is set to the, $\delta_{\text{passband}} = 0.05$ and stopband ripple is set to $\delta_{\text{stopband}} = 8e^{-5}$. The stopband attenuation before optimization was calculated to be equal to 80.6306 dB. And after optimization the stopband attenuation is calculated to 86.2407 dB. This is not an exact value of our required stopband attenuation, i.e. 86 dB, but it is the closest result obtained by using the actual design method. Since we are only interested in the number of multiplications required for the actual interpolator realization, this result represents a good enough approximation.
### Impulse response of first image-rejection filter, $H_1(z)$

<table>
<thead>
<tr>
<th>$h(n)$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0)$</td>
<td>0.00057774082079</td>
</tr>
<tr>
<td>$h(2)$</td>
<td>0.00643972940951</td>
</tr>
<tr>
<td>$h(4)$</td>
<td>-0.00026263397892</td>
</tr>
<tr>
<td>$h(6)$</td>
<td>-0.00089205887807</td>
</tr>
<tr>
<td>$h(8)$</td>
<td>0.00202440791367</td>
</tr>
<tr>
<td>$h(10)$</td>
<td>-0.00366405810654</td>
</tr>
<tr>
<td>$h(12)$</td>
<td>0.00597074996468</td>
</tr>
<tr>
<td>$h(14)$</td>
<td>-0.00909355164630</td>
</tr>
<tr>
<td>$h(16)$</td>
<td>0.01326812103153</td>
</tr>
<tr>
<td>$h(18)$</td>
<td>-0.01890154489968</td>
</tr>
<tr>
<td>$h(20)$</td>
<td>0.02677246464568</td>
</tr>
<tr>
<td>$h(22)$</td>
<td>-0.03854427669197</td>
</tr>
<tr>
<td>$h(24)$</td>
<td>0.05857005361823</td>
</tr>
<tr>
<td>$h(26)$</td>
<td>-0.10299707837375</td>
</tr>
<tr>
<td>$h(28)$</td>
<td>0.31726220502921</td>
</tr>
<tr>
<td>$h(30)$</td>
<td>0.31726220502921</td>
</tr>
<tr>
<td>$h(32)$</td>
<td>-0.10299707837375</td>
</tr>
<tr>
<td>$h(34)$</td>
<td>0.05857005361823</td>
</tr>
<tr>
<td>$h(36)$</td>
<td>-0.03854427669197</td>
</tr>
<tr>
<td>$h(38)$</td>
<td>0.02677246464568</td>
</tr>
<tr>
<td>$h(40)$</td>
<td>-0.01890154489968</td>
</tr>
<tr>
<td>$h(42)$</td>
<td>0.01326812103153</td>
</tr>
<tr>
<td>$h(44)$</td>
<td>-0.00909355164630</td>
</tr>
<tr>
<td>$h(46)$</td>
<td>0.00597074996468</td>
</tr>
<tr>
<td>$h(48)$</td>
<td>-0.00366405810654</td>
</tr>
<tr>
<td>$h(50)$</td>
<td>0.00202440791367</td>
</tr>
<tr>
<td>$h(52)$</td>
<td>-0.00089205887807</td>
</tr>
<tr>
<td>$h(54)$</td>
<td>-0.00026263397892</td>
</tr>
<tr>
<td>$h(56)$</td>
<td>0.00643972940951</td>
</tr>
<tr>
<td>$h(58)$</td>
<td>0.00057774082079</td>
</tr>
</tbody>
</table>

**Table 4.2:** Impulse response of $H_1(z)$ calculated by using optimization technique.
4.4 Design Method 3

From the estimated impulse response presented in Table 4.2 we can observe a slight deterioration in the interpolator implementation. Such conclusion is drawn from the resulting impulse responses calculated for the interpolator stage. The estimated filter order is still approximately equal when compared to the design method used in previous section, i.e. Section 4.2.

For the actual interpolator implementation, the number of multiplication blocks that are needed for correct interpolator realization is calculated to be equal to:

\[
\text{Number of Multiplications} = \text{Number of stages} \cdot \left( \frac{N + 2}{2} + 1 \right)
\]

\[
\text{Number of Multiplications} = 5 \cdot \left( \frac{58 + 2}{2} + 1 \right) = 155
\]

Thus, for this filter realization 155 multiplications units are needed.

Furthermore, for the actual interpolator implementation, the filter gain is set to 0.5. There is no need for additional gain increase since no adequate interpolator improvement is discovered as compared to the solution from the previous section. The required number of multiplications is higher as compared to the design method presented in the previous section.

4.4 Design Method 3

In the following section, realization of the interpolator chain will be based on the idea presented in the IEEE transcript [16]. Here, five different image-rejection FIR filters will be designed for every interpolator stage. For this final design method, the regular McClellan-Parks-Rabiner algorithm is used. The MatLab function used for actual calculation of image-rejection filters can be found in Appendix B. Tables below present the impulse response obtained for each individual image-rejection FIR filter that is found in the five stage interpolator realization.
**Impulse response of first image-rejection filter, $H_1(z)$**

<table>
<thead>
<tr>
<th>$h(n)$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0)$</td>
<td>0.00022030971657</td>
</tr>
<tr>
<td>$h(1)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(2)$</td>
<td>0.00068496570327</td>
</tr>
<tr>
<td>$h(3)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(4)$</td>
<td>0.00166555531434</td>
</tr>
<tr>
<td>$h(5)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(6)$</td>
<td>-0.00346435960797</td>
</tr>
<tr>
<td>$h(7)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(8)$</td>
<td>0.00648431644446</td>
</tr>
<tr>
<td>$h(9)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(10)$</td>
<td>-0.01125664848278</td>
</tr>
<tr>
<td>$h(11)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(12)$</td>
<td>0.01847964697098</td>
</tr>
<tr>
<td>$h(13)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(14)$</td>
<td>-0.02916937883389</td>
</tr>
<tr>
<td>$h(15)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(16)$</td>
<td>0.04500677088078</td>
</tr>
<tr>
<td>$h(17)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(18)$</td>
<td>-0.06939146208859</td>
</tr>
<tr>
<td>$h(19)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(20)$</td>
<td>0.11103243246429</td>
</tr>
<tr>
<td>$h(21)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(22)$</td>
<td>-0.20205364905526</td>
</tr>
<tr>
<td>$h(23)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(24)$</td>
<td>0.63316959512440</td>
</tr>
<tr>
<td>$h(25)$</td>
<td>1</td>
</tr>
<tr>
<td>$h(26)$</td>
<td>0.63316959512440</td>
</tr>
<tr>
<td>$h(27)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(28)$</td>
<td>-0.20205364905526</td>
</tr>
<tr>
<td>$h(29)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(30)$</td>
<td>0.11103243246429</td>
</tr>
<tr>
<td>$h(31)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(32)$</td>
<td>-0.06939146208859</td>
</tr>
<tr>
<td>$h(33)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(34)$</td>
<td>0.04500677088078</td>
</tr>
<tr>
<td>$h(35)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(36)$</td>
<td>-0.02916937883389</td>
</tr>
<tr>
<td>$h(37)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(38)$</td>
<td>0.01847964697098</td>
</tr>
<tr>
<td>$h(39)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(40)$</td>
<td>-0.01125664848278</td>
</tr>
<tr>
<td>$h(41)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(42)$</td>
<td>0.00648431644446</td>
</tr>
<tr>
<td>$h(43)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(44)$</td>
<td>-0.00346435960797</td>
</tr>
<tr>
<td>$h(45)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(46)$</td>
<td>0.00166555531434</td>
</tr>
<tr>
<td>$h(48)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(49)$</td>
<td>-0.0068496570327</td>
</tr>
<tr>
<td>$h(50)$</td>
<td>0</td>
</tr>
<tr>
<td>$h(51)$</td>
<td>0.00022030971657</td>
</tr>
<tr>
<td>$h(52)$</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 4.3: Impulse response of first half-band FIR filter, $H_1(z)$.**
4.4 Design Method 3

**Table 4.4:** Impulse response of second half-band FIR filter, $H_2(z)$.

<table>
<thead>
<tr>
<th>$h(n)$</th>
<th>$H_2(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0) = 0$</td>
<td>$h(1) = -0.00156207297295$</td>
</tr>
<tr>
<td>$h(2) = 0$</td>
<td>$h(3) = 0.00823202446016$</td>
</tr>
<tr>
<td>$h(4) = 0$</td>
<td>$h(5) = -0.02694515179199$</td>
</tr>
<tr>
<td>$h(6) = 0$</td>
<td>$h(7) = 0.07021868487182$</td>
</tr>
<tr>
<td>$h(8) = 0$</td>
<td>$h(9) = -0.17219519931157$</td>
</tr>
<tr>
<td>$h(10) = 0$</td>
<td>$h(11) = 0.62218108701557$</td>
</tr>
<tr>
<td>$h(12) = 1$</td>
<td>$h(13) = 0.62218108701557$</td>
</tr>
<tr>
<td>$h(14) = 0$</td>
<td>$h(15) = -0.17219519931157$</td>
</tr>
<tr>
<td>$h(16) = 0$</td>
<td>$h(17) = 0.07021868487182$</td>
</tr>
<tr>
<td>$h(18) = 0$</td>
<td>$h(19) = -0.02694515179199$</td>
</tr>
<tr>
<td>$h(20) = 0$</td>
<td>$h(21) = 0.00823202446016$</td>
</tr>
<tr>
<td>$h(22) = 0$</td>
<td>$h(23) = -0.00156207297295$</td>
</tr>
</tbody>
</table>

**Table 4.5:** Impulse response of third half-band FIR filter, $H_3(z)$.

<table>
<thead>
<tr>
<th>$h(n)$</th>
<th>$H_3(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0) = 0$</td>
<td>$h(1) = 0.01551811020509$</td>
</tr>
<tr>
<td>$h(2) = 0$</td>
<td>$h(3) = -0.10779745130994$</td>
</tr>
<tr>
<td>$h(4) = 0$</td>
<td>$h(5) = 0.59233827295821$</td>
</tr>
<tr>
<td>$h(6) = 1$</td>
<td>$h(7) = 0.59233827295821$</td>
</tr>
<tr>
<td>$h(8) = 0$</td>
<td>$h(9) = -0.10779745130994$</td>
</tr>
<tr>
<td>$h(10) = 0$</td>
<td>$h(11) = 0.01551811020509$</td>
</tr>
</tbody>
</table>

**Table 4.6:** Impulse response of fourth half-band FIR filter, $H_4(z)$.

<table>
<thead>
<tr>
<th>$h(n)$</th>
<th>$H_4(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0) = 0$</td>
<td>$h(1) = -0.06516677274199$</td>
</tr>
<tr>
<td>$h(2) = 0$</td>
<td>$h(3) = 0.56509319606772$</td>
</tr>
<tr>
<td>$h(4) = 1$</td>
<td>$h(5) = 0.56509319606772$</td>
</tr>
<tr>
<td>$h(6) = 0$</td>
<td>$h(7) = -0.06516677274199$</td>
</tr>
</tbody>
</table>
Implementation

<table>
<thead>
<tr>
<th>Impulse response of fifth image-rejection filter, $H_5(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(0) = -0.06315455382372$</td>
</tr>
<tr>
<td>$h(2) = 0.56315001969458$</td>
</tr>
<tr>
<td>$h(4) = 0.56315001969458$</td>
</tr>
<tr>
<td>$h(6) = -0.06315455382372$</td>
</tr>
</tbody>
</table>

Table 4.7: Impulse response of fifth half-band FIR filter, $H_5(z)$.

For the actual design method, number of multiplications that are required for proper interpolator functionality is calculated to be:

- Number of Multiplications, $H_1(z) = 26$
- Number of Multiplications, $H_2(z) = 12$
- Number of Multiplications, $H_3(z) = 6$
- Number of Multiplications, $H_4(z) = 4$
- Number of Multiplications, $H_5(z) = 4$

Naturally, multiplications by the value of 1 are not counted for in the final sum. Hence, the total number of multiplications required for correct realization of the entire interpolator chain is calculated to be equal to:

Total number of multiplications = 52

What can be seen from the presented tables, for this interpolator design method only 52 multiplications are required. As compared to two previously presented design approaches from Sections 4.2 and 4.3, this design alternative gives the best possible interpolator implementation when not only system functionality is of concern but also when small hardware cost is one design constraint.

The actual interpolator implementation can be simplified further. By taking advantage of coefficient symmetry\(^5\), the actual number of

\(^5\)See section 3.1.3
4.4 Design Method 3

multiplications can be further reduced. After performing new calculations only 26 multiplications are required. That is

\[
\text{Total number of multiplications} = \frac{(26 + 12 + 6 + 4 + 4)}{2} = 26
\]

The resulting impulse response of the entire interpolation chain is illustrated in the figure below.

![Magnitude response of interpolation chain, OSR = 32.](image)

**Figure 4.6:** Magnitude response of interpolation chain, $OSR = 32$.

What we can observe in Fig. 4.6 is that the signal attenuation of $86\, dB$ in the stopband range is achieved. This is one main design requirement. Furthermore, both stopband and passband edge constraints that are given by the design specification are fulfilled.
4.5 Chosen Design Method

From previous sections we determined that three different design methods can be successfully used for realization of the actual interpolator system. We further observed that all three design methods were successful in fulfilling design specification given in Section 1.2. That is, the discrete input signal is upsampled 32 times and a stopband attenuation of $86\,dB$ is achieved.

However, when low power consumption and small implementation area is of concern, the design method presented in Section 4.4 gave the best possible solution. This conclusion can be drawn from the fact that for the actual interpolator realization only 26 multiplication units are needed. This is a small number of multiplications as compared to the other two possible design methods.

Hence, the design method presented in Section 4.4 will be our chosen method as it gave best possible results.

4.6 Coefficient rounding

Since the magnitude response of the multistage interpolator realization from Section 4.4 fulfills the given design specification where signal images are attenuated to a required $86\,dB$, next step towards fully functional interpolator design realization will be to round the image-rejection filters coefficient values to some appropriate numerical values so that the constraint imposed on interpolator magnitude response is maintained.

In each of the previously presented design methods, the actual interpolation operation is performed by five successive interpolator stages interconnected into a single interpolation chain. Each interpolator stage is realized as a combination of polyphase lowpass filter and single commutator. Commutator blocks usually do not influence actual system functionality. This is only true if they are clocked properly. On the contrary, when filter coefficients are rounded to some appropriate numerical representation, each stage will have different impact.
4.6 Coefficient rounding

on the overall system magnitude response. This will be the case as the number of stages in the multistage interpolator realization is increased from one to five stages, where each polyphase filter is previously rounded to its appropriate numerical representation. Thus, with each new introduced interpolator stage, the overall interpolator magnitude response will be influenced accordingly. Here, we will compare our chosen design method to that of Section 4.2.

As stated previously in Section 4.2 and Section 4.4, the regular MatLab function remez was used for estimation of individual filter impulse responses. The resulting filter impulse response obtained by remez function, is usually an approximation of an ideal or desired filter, \( H(z)_{\text{ideal}} \). Subsequently, when the designed filter is implemented in hardware, the finite number of bits available for filter coefficient representations further influences the frequency response of the desired filter [1] [17]. Thus, the implemented filter will differ both from the ideal- and desired filter frequency response obtained by MatLab function remezord. In many cases, the initial filter specification often gives some tolerance for coefficient errors deduced by finite filter coefficient word lengths but still every multiplication operation in the filter implementation introduces distortion into the frequency response.

In addition, as filter coefficients do not have equal signs\(^6\), they must be represented as a signed fixed-point rationals. Thus, in order to estimate an ideal frequency response, filter coefficients have to be represented more accurately by using signed fixed-point rationals.

The form of signed fixed point rationals is \( B_i/2^b \), where \( B_i \) and \( b \) are some integer values. The data word length used for filter coefficient representation will be bounded by the relation, \(-2^{M-1} \leq B_i \leq 2^{M-1}-1\), where \( M \) is some integer value. The value \( M \) is used to fix the filter coefficients data word lengths [1] [11]. Thus, by choosing some appropriate value of \( M \), the estimated value \( b'_i \) of the calculated filter coefficient \( b_i \) is than equal to

\(^6\)Impulse response is a mix of positive and negative values.
\[ b'_i = \frac{\text{round}(b_i 2^{M-1})}{2^{M-1}} \]  

(4.1)

In general, \( b'_i \) is only an estimation of \( b_i \). This is because of rounding operation and finite word length imposed by chosen value of \( M \).

The MatLab script used for coefficient approximation is located in Appendix C. This script plots the worst case stopband attenuation as the word length, \( M \), is increased from 15 bits to 22 bits. Simulation starts by plotting the output of the first interpolator stage while changing the word length from 15 to 22 bits. Simulation stops after plotting the output of the last interpolator stage where all stages are interconnected into the actual interpolator chain. Here, the interpolator model from section 4.2 is used. The Fig. 4.7 illustrates resulting plot.

**Figure 4.7:** Stopband attenuation with respect to changing coefficient word length, \( M \).
4.6 Coefficient rounding

From the results plotted in Fig. 4.7, we can see how the actual attenuation in the stopband is influenced by the chosen data word length, \( M \). If the value of \( M \) is chosen to be small, the required attenuation of 86 dB in the stopband will not be achieved as stopband ripples increase with lack of numerical precision. This could be observed for all five cases where the number of interpolator stages in the chain is increased from one to five stages. On the contrary, as the word length is increased from 18 to 22 bits, the difference in worst case stopband attenuation becomes approximately equal. This can be observed in the figure above for the cases when data word length is in range between 18 and 22 bits. Thus, for the actual design method, each polyphase filter found in the interpolation chain must be rounded to the word length representation that is larger than 17 bits. Optimal results are obtained for the case when \( M \) is chosen to be equal or larger than 18 bits.

On contrary, for our chosen interpolator design method presented earlier in section 4.4, each image-rejection FIR filter that is used in the interpolation chain will have different orders. That is, they will have different impulse responses. As such they could be rounded to their individual data word lengths. Depending on the chosen values of \( M \), the overall interpolator frequency response will be affected differently by each interpolator stage. For example, from Table 4.7 we can observe that for lowpass, \( H_5(z) \), filter realization, only five multiplications are required\(^7\). This, on the contrary to lowpass image-rejection filter, \( H_1(z) \) in Table 4.3, is a small number of multiplication operations that must be performed. Thus, impact on stopband attenuation by \( H_5(z) \) will not be as great as compared with filters that have higher orders like filter, \( H_1(z) \) or \( H_2(z) \). Therefore, each interpolator stage in the interpolation chain will be rounded to its individual data word length.

The MatLab script that was used for the actual estimation of individual interpolator data word lengths can be found in Appendix C.

\(^7\)Multiplication by zero value is not included as well as multiplication by one. Multiplication with zero will not be performed since the result is always zero and multiplications by value of one will be realized as a single wire connection.
Fig. 4.6 illustrates the magnitude response of the multistage interpolator chain when coefficient word length of 24 bits, i.e. $M = 24$, is used for rounding each separate interpolator stage. This can be compared with corresponding magnitude response of Fig. 4.8. Here, each interpolator stage is rounded to it’s individual data word length.

![Graph](image-url)

**Figure 4.8:** Magnitude response of the interpolation chain, $OSR = 32$.

The Table 4.8 on the next page summarized the calculated data word length of each separate interpolator stage.
4.7 Scaling

<table>
<thead>
<tr>
<th>Estimated interpolator data word lengths.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_1$</td>
</tr>
<tr>
<td>$H_2$</td>
</tr>
<tr>
<td>$H_3$</td>
</tr>
<tr>
<td>$H_4$</td>
</tr>
<tr>
<td>$H_5$</td>
</tr>
</tbody>
</table>

Table 4.8: Calculated interpolator filter coefficient word lengths.

What we can observe from Fig. 4.8 and Table 4.8 is that considerable hardware savings can be achieved when the interpolator stages are rounded to their individual coefficient word lengths. From Fig. 4.8 we can observe that by rounding interpolator stages to the coefficient word length values shown in Table 4.8, the magnitude response of the whole interpolation chain satisfies the design specifications given in section 1.2. That is, the signal is attenuated below $86 \, dB$ in stopband.

Hence, each individual interpolator stage is rounded to the coefficient word lengths depicted in Table 4.8.

4.7 Scaling

Main arithmetic operations performed inside digital filters are multiplications and additions. Such operations have to be scaled to an extent so that final results at the output of each operation are within a valid numerical range. On the contrary, if results of such operations are not controlled, an overflow will happen. Unwanted overflows will introduce distortion inside the system. As a result, SNR\(^8\) level will decrease. This is an unwanted scenario as we do not want to decrease signal power.

In order for the interpolation chain to work properly each interpolator stage in the multistage realization must be properly scaled. As discussed earlier in the theory part, problems arise when one uses a

\(^8\)SNR stand for Signal to Noise ratio.
Implementation

chain of interpolator stages as signal stationarity is lost. This problem arises due to polyphase realization of interpolator stages. Different samples at the interpolator output will have different statistical properties. This could be explained by the fact that the adder is realized as the commutator in its polyphase representation. Thus, each commutator output will have different statistical probability of overflow. This will result in a variance value that is constantly changing at the output of the commutator. This is the case, as for each new time instant the commutator selects different polyphase path where each path has different statistical possibility for overflow. Thus, variance will be time-varying and periodic with a period of $L$. The parameter $L$ annotates the upsampling ratio.

For the multistage interpolator realization, the scaling process presented earlier in Section 3.3 has been used. The idea behind the chosen method is to rewrite each interpolator substage into its polyphase representation. Thus, the variance at the outputs of the substage becomes constant. As such usual scaling methods can be used. For scaling purposes the safe scaling method is used. A MatLab script used for actual scaling coefficients estimation can be found in Appendix D. The table below summarizes the results obtained at the critical nodes:

<table>
<thead>
<tr>
<th>Critical Nodes</th>
<th>Calculated Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_1$, $c_0$</td>
<td>2.26419067382812</td>
</tr>
<tr>
<td>$H_2$, $c_1$</td>
<td>2.26419067382812</td>
</tr>
<tr>
<td>$H_3$, $c_2$</td>
<td>2.26419067382812</td>
</tr>
<tr>
<td>$H_4$, $c_3$</td>
<td>2.26419067382812</td>
</tr>
<tr>
<td>$H_5$, $c_4$</td>
<td>2.26419067382812</td>
</tr>
</tbody>
</table>

Table 4.9: Calculated values of critical nodes.

Here, critical nodes are the outputs of each interpolator stage. From results summarized in Table 4.9 we can conclude that the each interpolator input must be multiplied by the $1/c_n$ to eliminate possible overflows.
4.8 Roundoff Noise Measurement

During hardware realization, scaling multipliers will not be realized but instead extra bits are added to represent the highest numerical value. For this case, extra two bits are added to the word length as to be able to represent the highest value at the output of each interpolator. This is the case as the worst case value at the output of each interpolator stage is equal to $2^{26}419067382812$. To be able to represent this value in binary form two extra bits are added to the data word length. Thus, each interpolator output is increased by two extra bits.

4.8 Roundoff Noise Measurement

In this section a measurement of round-off noise is discussed. Depending on the calculated round-off noise, number of bits at the interpolator outputs will be changed. The noise measurement method shown earlier in Section 3.4 is used. The data word length of one interpolator block is set to infinite length while the data word length of a second block is set to finite length. As the input signal, a white noise source is chosen. The difference between the two blocks will give the value of the round-off noise of the interpolator system.

\[ H_{Quant}(z) \]
\[ Y_Q(n) \]
\[ H_{Ideal}(z) \]
\[ Y_I(n) \]
\[ x(n) + e(n) \]

**Figure 4.9:** Structure for round-off noise simulation, $M$.

The value of the round-off noise generated at the output of interpolator chain when white noise source is the input signal, should be smaller than the value of the round-off noise that is generated prior to the interpolator block. By doing so we can guaranty that the noise generated inside interpolator chain will not influence already generated noise power at the input of interpolator system. Thus, the variance
Implementation

at the input and the output of the system will be approximately equal.

Thus, the round-off noise variance at the input of the interpolator system is calculated to be equal to $7.761021455128987e^{-11}$. The calculation is performed as follows:

$$
\sigma_e^2 = \frac{2^{-2(B-1)}}{12}
$$

where $B$ annotates the data word length used prior the interpolator chain which in this case is equal to 16. Thus, input variance is equal to:

$$
\sigma_e^2 = \frac{2^{-2(16-1)}}{12} = 7.761021455128987e^{-11}
$$

On the other hand, variance that is generated by interpolator chain is calculated to be equal to $7.7100475754863e^{-13}$ which is approximately 100 time smaller than the input variance. This is only the case when the data word length of each interpolator stage output is set to $M = 20$ bits. Actual roundoff-noise calculation can be found in Appendix (D).

Thus, data word length of each interpolator output should be larger than the 20 bits.

4.9 MCM algorithm

When filters of individual interpolator stages are rounded and scaled, next step towards a functional interpolator implementation is to implement the image-rejection FIR filter of each interpolator stage by using Multiple Constant Multiplication algorithm, (MCM). The main idea in this design phase was to obtain simpler FIR filters. Here, filter multiplications are realized as a combination of adders, substractors and shifts. Hence, an interpolator realization without general multipliers was obtained. From previous discussions we know that every other filter coefficient of the image-rejection filter is equal to zero. Thus,
4.9 MCM algorithm

zero valued multiplications will not be realized. For polyphase half-band FIR filter, the designer needs only to implement one subfilter block as one side only contains delay elements. Thus, further hardware simplification is done. Furthermore, filter coefficient symmetry is also utilized.

For purpose of simpler system implementation, multipliers are realized as shift and add\textsuperscript{9} units. Here, an MCM algorithm was helpful in realizing each filter multiplication in its corresponding shift and add unit. For this an online algorithm generator is used. Table 4.10 summarizes the required number of adders that are needed for the actual implementation of the interpolator chain.

<table>
<thead>
<tr>
<th>The required number of adders.</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image-rejection filter</td>
<td>H\textsubscript{1}</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>H\textsubscript{2}</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>H\textsubscript{3}</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>H\textsubscript{4}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>H\textsubscript{5}</td>
<td>1</td>
</tr>
<tr>
<td>Total number of adders</td>
<td></td>
<td>38</td>
</tr>
</tbody>
</table>

Table 4.10: The required number of adders for the actual interpolator implementation.

From table above we can draw a conclusion that only 38 adders are required for the actual interpolator implementation. This is off course a small number of required adders as compared to the straightforward implementation where no MCM algorithm is used. Later on, the generated filter block structure is used to generate the corresponding vhdl files.

\textsuperscript{9}Since the additions and subtractions have similar complexity, they will be referred as additions.
4.10 FPGA Implementation

In this section a low-level hardware implementation will be discussed. The chapter starts with description of the FPGA board followed by the hardware implementation of the interpolator chain.

4.10.1 Introduction

The chosen FPGA board for this project was the *Altera DE2 Board, (UP3 – 2C35F672C6 family)*. The DE2 board has many features that allow the user to implement a wide range of different designed circuits. Such digital circuits can range from simple circuits to various multimedia projects. All information about the FPGA board and its components presented on this page have been taken from the *DE_user_manual.pdf*. This file is included in the CD of the DE2 package. The Fig. 4.10 illustrates the board in question.

![Altera DE2 Board](image)

*Figure 4.10: Altera DE2 Board.*
4.10 FPGA Implementation

The following hardware components can be found on the DE2 board:

- Altera Cyclone® II 2C35 FPGA device
- Altera Serial Configuration device - EPCS16
- USB Blaster (on board) for programming and user API control; both JTAG and Active Serial (AS) programming modes are supported
- 512-Kbyte SRAM
- 8-Mbyte SDRAM
- 4-Mbyte Flash memory (1 Mbyte on some boards)
- SD Card socket
- 4 pushbutton switches
- 18 toggle switches
- 18 red user LEDs
- 9 green user LEDs
- 50-MHz oscillator and 27-MHz oscillator for clock sources
- 24-bit CD-quality audio CODEC with line-in, line-out and microphone-in jacks
- VGA DAC (10-bit high-speed triple DAC’s) with VGA-out connector
- TV Decoder (NTSC/PAL) and TV-in connector
- 10/100 Ethernet Controller with a connector
- USB Host/Slave Controller with USB type A and type B connectors
- RS-232 transceiver and 9-pin connector
Implementation

- PS/2 mouse/keyboard connector
- IrDA transceiver
- Two 40-pin Expansion Headers with diode protection

In addition to the hardware features mentioned above, the DE2 board has also large software support for standard I/O interfaces. One of these software applications that comes with a DE2 board is the Control Panel. This software facilitates access to various board components. In order to run this application, the corresponding circuit in the DE2 board have to be configured. This is done by downloading the configuration file `DE2_USB_API.sof` into the FPGA. In addition, the software program `DE2_control_panel.exe` on the host computer had to be executed. Both of these applications come with the DE2 board.

Throughout this project, Control Panel was used to read stored data from the SRAM chip on the DE2 board. Furthermore, Control Panel allows users to read the content of the SRAM and store data into a file. This file can later on be used for MatLab simulations.

Figure 4.11 on the next page illustrates the block diagram of the DE2 board.
4.10 FPGA Implementation

Figure 4.11: Altera DE2 Board block diagram.

More detailed hardware information regarding the DE2 board:

Cyclone II 2C35 FPGA

- 33,216 LEs
- 105 M4K RAM blocks
- 483,840 total RAM bits
- 35 embedded multipliers
- 4 PLLs
- 475 user I/O pins
- FineLine BGA 672-pin package

Serial Configuration device and USB Blaster Circuit
Implementation

- Altera’s EPCS16 Serial Configuration device
- On-Board USB Blaster for programming and user API control
- JTAG and AS programming modes are supported

SRAM
- 512-Kbyte Static RAM memory chip
- Organized as 256K x 16 bits
- Accessible as memory for the Nios II processor and by the DE2 Control Panel

SDRAM
- 8-Mbyte Single Data Rate Synchronous Dynamic RAM memory chip
- Organized as 1M x 16 bits x 4 banks
- Accessible as memory for the Nios II processor and by the DE2 Control Panel

Flash Memory
- 4-Mbyte NAND Flash memory (1 Mbyte on some boards)
- 8-bit data bus
- Accessible as memory for the Nios II processor and by the DE2 Control Panel

SD card socket
- Provides SPI mode for SD Card access
- Accessible as memory for the Nios II processor with the DE2 SD Card Drive

Pushbutton switches
4.10 FPGA Implementation

- 4 pushbutton switches
  - Debounced by a Schmitt trigger circuit
  - Normally high. Generates one active-low pulse when the switch is pressed

Toggle switches

- 18 toggle switches for user inputs
  - A switch causes logic 0 when in the DOWN position (closest to the edge of the DE2 board), and logic 1 when in the UP position

Clock inputs

- 50-MHz oscillator
- 27-MHz oscillator
- SMA external clock input

Audio CODEC

- Wolfson WM8731 24-bit sigma-delta audio CODEC
  - Line-level input, line-level output and microphone input jacks
  - Sampling frequency: 8 upto 96 kHz
  - Applications for MP3 players and recorders, PDAs, smart phones, voice recorders, etc.

VGA output

- Uses ADV7181B Multi-format SDTV Video Decoder
  - Supports NTSC-(M,J,4.43), PAL-(B/D/G/H/I/M/N),SECAM
  - Integrates three 54-MHz 9-bit ADCs
  - Clocked from a single 27 MHz oscillator input
Implementation

- Multiple programmable analog input formats: Composite video (CVBS), S-Video(Y/C) and YPrPb components
- Supports digital output formats (8-bit/16-bit): ITU-R BT.656 YCrCb 4:2:2 output + HS, VS and FIELD
- Applications: DVD recorders, LCD TV, Set-top Boxes, Digital TV, Portable video devices

10/100 Ethernet controller

- Integrated MAC and PHY with a general processor interface
- Supports 100Base-T and 10Base-T applications
- Supports full-duplex operation at 10 Mb/s and 100 Mb/s with auto-MDIX
- Fully compliant with the IEEE 802.3u Specification
- Supports IP/TCP/UDP checksum generation and checking
- Supports back-pressure mode for half-duplex mode flow control

USB Host/Slave controller

- Complies fully with Universal Serial Bus Specification Rev.2.0.
- Supports data transfer at full-speed and low-speed
- Supports both USB host and device
- Two USB ports (one type A for a host and one type B for device)
- Provides a high-speed parallel interface to most available processors; supports Nios II with a Terasic drive
- Supports Programmed I/O (PIO) and Direct Memory Access (DMA)

Serial ports

- One RS-232 port
4.10 FPGA Implementation

- One PS/2 port
- DB-9 serial connector for the RS-232 port
- PS/2 connector for connecting a PS2 mouse or keyboard to the DE2 board

IrDA transceiver

- Contains a 115.2-kb/s infrared transceiver
- 32 mA LED drive current
- Integrated EMI shield
- IEC825-I Class 1 eye safe
- Edge detection input

Two 40-pin expansion headers

- 72 Cyclone II I/O pins, as well as 8 power and ground lines, are brought out to two 40-pin expansion connectors
- 40-pin header is designed to accept a standard 40-pin ribbon cable used for IDE hard drives
- Diode and resistor protection is provided

4.10.2 Board Applications

The DE2 board has many different application fields. One can use them for Audio or Video applications, as the board has many audio and video capabilities. It also allows the user to connect to the DE2 board throughout several different I/O ports. One can use USB port, PS/2, IrDA, Ethernet, RS.32, . . . . Some examples of board applications are:

- Playing audio and video input from a DVD player using the VGA output and the Audio CODEC component that are found on the board.
Implementation

- A karaoke machine that uses the Audio CODEC and the microphone-in, line-in and line-out port of the board.

- Nios II process that reads music data stored in the SD-card and uses the Audio CODEC to play the music.

- Using the Philips ISP1362 chip and the Nios II processor to implement a USB mouse movement detector.

Since the goal of this project was to implement and test an interpolator system, several components on the DE2 board had to be used. Two main hardware components used thought this project were 24-bit CD-quality Audio CODEC and asynchronous 512-Kbyte SRAM memory chip. Previous to the interpolator implementation, the Audio CODEC had to be configured such that it outputs 16-bits audio data. The 16-bit audio data was required to test the interpolator circuit.

Several other board components, such as internal clocks, toggle switches, USB connections and Control Panel were also needed for implementation and testing of the multistage interpolator. Furthermore, components like LED’s are used to facilitate the simulation process of the interpolator system.

The Fig. 4.12 illustrates the way how the interpolator testing was performed.

![Figure 4.12: Interpolator test structure.](image)

An audio signal from a CD is sent via the line-level input port, to the audio interface. In the audio interface, a single audio channel is
4.10 FPGA Implementation

sampled and digitized to its corresponding 16-bit binary data representation. Resulting data is consequently sent to the interpolator block where the signal is upsampled and filtered. As the SRAM memory is organized as 256Kx16 bits, of resulting 20 bits that are generated by the interpolator block only the 16 most significant bits are saved into the SRAM memory. Thus, resulting interpolator data must be truncated. Four least significant bits are discarded/truncated.

Furthermore, for simulation reasons both data from the audio interface and interpolator block are simultaneously saved into the SRAM memory. This gives us the possibility to compare the interpolator output data with the reference input data generated by the audio interface.

4.10.3 Audio CODEC Interface

In this subsection a small introduction to the WM8731/WM8731L audio interface is done.

The Audio CODEC included with the Altera DE2 board is a low power stereo 24-bit ΔΣ-audio CODEC that has an integrated headphone driver. Mainly, the interface is designed for portable MP3 players and recorders, CD and mini-disc recorders, PDAs and smart phones. The block diagram of audio CODEC is shown in Fig. 4.13 on the next page.
The audio interface has two different input ports. There is a stereo line-level input and a single microphone input. Both inputs are provided with their separate volume and mute functions. These functions are internally controlled by the control interface of the Audio CODEC, allowing volume and mute control.

The interface uses two 24-bit sigma delta converters, ADCs and DACs, together with oversampling interpolator and decimator filters to generate a digitalized output signal. Output data is signed 2’s complement. Furthermore, the CODEC sampling frequency can be chosen between values of 8 KHz to 96 KHz. In addition, by using the control interface the digital audio output can be programed to 16, 24 or 32 bits. This interface configuration is only possible in I2S and left justified mode. If the interface is programed to output 16 bit data, the least significant bits are truncated from 24 bit data.

In this project the audio interface was configured as a slave component, meaning that all CODEC control signals had to be generated by the DE2 board. The default values of the audio output data word length and sampling frequency are set to 24 bits and 41.22 KHz, re-
4.10 FPGA Implementation

respectively. Thus, these default values had to be changed to reflect our specification requirements. Furthermore, in the default mode the main clock frequency of the interface is set to $12.288 \, MHz$ and since the Audio CODEC was configured as a slave component, a master clock signal had to be generated outside the audio interface.

The external clock that is taken from the DE2 board is the $\text{clock}_50$ signal located at pin N2 that generates a clock with frequency of $50 \, MHz$. To get the required audio clock frequency, this clock signal had to be divided by four to generate a clock with $12.5 \, MHz$ clock frequency, i.e. $f_{\text{clock}} = 50 \, MHz / 4 = 12.5 \, MHz$. This is not the exact clock frequency required by the audio interface but it was good enough for the testing purposes. The filter accuracy was somewhat decreased with this but no significant change in the audio signal was detected while checking the final results.

Additional clock signals that had to be generated in order to control the audio interface were:

* ADCLRC, an alignment clock that controls whether left or right channel is present on the ADCDAT output

* DACLRC, an alignment clock that controls whether left or right channel is present on the DACDAT output

* BCLK, is a bit shift clock which on falling edge indicates if the data is ready to be read/write. The signal frequency of this signal is set to $f_{\text{BCLK}} = f_{\text{clock}} / 4$.

All clock signals are generated by the following VHDL code:

```vhdl
signal counter std_logic_vector(9 downto 0);
begin
if (rising_edge(clk)) then
if (reset = '0') then
counter <= (others => '0');
else
counter <= counter + 1;
end if;
```
Furthermore, the left justified mode was implemented in this project. This is not the default mode for the Audio CODEC so the default interface configuration had to be changed in the control interface. The left justified mode is the easier mode to program the interface by and it was good enough for this project.

In this mode, the MSB is available on the first falling edge of the BCLK clock, followed by an ADCLRC or DACLRC transition. The timing of the left justified mode is shown in Fig. 4.14.

![Figure 4.14: Left justified mode.](image)

**Audio CODEC configuration**

The control interface of the Audio CODEC can be configured with the 3-wire or 2-wire serial control mode. By default the setting for serial control mode of the Audio CODEC component can be selected by setting the state of the MODE pin which is located on the Audio CODEC. But on the DE2 board this pin is grounded. Thus, the 2-wire serial control interface is selected by default for the Audio CODEC configuration.

The clock, SCLK, that is used for the control of serial data input must have a maximal frequency of 400 kHz. This clock was generated by dividing the 50 MHz clock by 128 to generate the SCLK of
4.10 FPGA Implementation

![Diagram of 2-wire serial interface mode for the Audio CODEC configuration.](image)

Figure 4.15: 2-wire serial interface mode for the Audio CODEC configuration.

The initial condition for starting the Audio CODEC configuration operation is the falling edge on the SDIN while SCLK is kept high. This can be observed in Fig. 4.15. The nextcomning seven bits determines which device that will receive data. Since the CSB state is set to ground by default, the address is set to "0011010". After the address, the R/W bit determines the direction of data transfer. In this case a '0' indicates the write operation.

If the device recognizes given address and R/W bit, the device pulls down the SDIN low during the ninth clock cycle of SCLK, acknowledging the data transfer. After the acknowledgment signal, two bit vectors are followed. These two bit vectors contain the configuration data. Bit vectors are separated with yet another acknowledgment signal. First bit vector, B[15:9], contains the control address bits and the second bit vector, B[8:0], contains the control data bits.

The rising edge of SDIN indicates the stop condition. For proper stop condition, the rising edge on SDIN must occur when SCLK is high or otherwise a faulty condition is detected and the device jumps back to the idle state. Also, if a start condition is detected at any point during the data transfer, the device will jump back to the idle condition. After a complete control operation, the audio CODEC returns back to the idle state and waits for another start condition.
The vhdl-code that was used to generate the SDIN signal is shown below:

case cntr is

-- Starting condition.
when 0 => SDIN <= start;
when 64 => SDIN <= ack;

-- CS address, i.e. 0011010.
when 192 => SDIN <= address(6);
when 448 => SDIN <= address(5);
when 704 => SDIN <= address(4);
when 960 => SDIN <= address(3);
when 1216 => SDIN <= address(2);
when 1472 => SDIN <= address(1);
when 1726 => SDIN <= address(0);

-- R/W set to '0'.
when 1984 => SDIN <= rw;

-- Acknowledging
when 2240 => SDIN <= ack;

-- Control Address
when 2496 => SDIN <= control(15);
when 2752 => SDIN <= control(14);
when 3008 => SDIN <= control(13);
when 3264 => SDIN <= control(12);
when 3520 => SDIN <= control(11);
when 3776 => SDIN <= control(10);
when 4032 => SDIN <= control(9);
when 4288 => SDIN <= control(8);

-- Acknowledging
when 4544 => SDIN <= ack;

-- Control Data
when 4800 => SDIN <= control(7);
when 5056 => SDIN <= control(6);
4.10 FPGA Implementation

when 5312 => SDIN <= control(5);
when 5568 => SDIN <= control(4);
when 5824 => SDIN <= control(3);
when 6080 => SDIN <= control(2);
when 6336 => SDIN <= control(1);
when 6592 => SDIN <= control(0);

−− Acknowledging and finishing by indicating stop condition.
when 6848 => SDIN <= ack;
when 7232 => SDIN <= stop;

−− End
when others => null;
end case;

Since the default configuration values of the Audio CODEC did not satisfy the required design specification given for this project, they had to be altered. This was done by changing the default values stored in the 11 registers shown in Fig. 4.16, by using the previous VHDL-code. These registers are 16-bits wide data registers of which 7 bits are used to indicate the control address and 9 bits for the control data.

The register map is shown in Fig. 4.16.
Initially, the *mute* option had to be deactivated and the *left to right channel* input value and *mute data control* had to be activated for the *left line in*. The corresponding register address in the register map is the $R0$, i.e. "00000000". This corresponds to the $B[15 : 9]$ in the 2-wire configuration process previously presented. To change the default data value stored in this register, bit $B7$ had to be set to '0' and bit $B8$ had to be set to '1'. Other data bits stored in register $R0$ are kept unchanged. By performing the configuration as presented, the *mute option* was turned off and the *right line in* volume and mute options were set to the same configuration values as that of the *left line in*.

The corresponding control sequence was set to:

\[
\text{control} := '0000000100010111'
\]

There is also an *internal bypass* option activated in the default configuration of the audio CODEC. Thus, this default option had to be changed also, i.e. option had to be deactivated. Furthermore, the *DAC select* option had to be activated. Thus, the default value

---

### Figure 4.16: Register map.

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>B15</th>
<th>B14</th>
<th>B13</th>
<th>B12</th>
<th>B11</th>
<th>B10</th>
<th>B9</th>
<th>B8</th>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 (00h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R1 (02h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R2 (04h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R3 (06h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R4 (08h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R5 (0Ah)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R6 (0Ch)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R7 (0 Eh)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R8 (10h)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R9 (12h)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R15 (1 Eh)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.10 FPGA Implementation

stored in the analog audio path register \( R4 \), i.e. "0000100", had to be changed. Bit \( B3 \) is set to '0' and bit \( B4 \) to '1'.

The corresponding control sequence was set to:

\[
control := '0000100000010010'
\]

Furthermore, the default value stored in register \( R6 \), i.e. "0000110", also had to be changed. This register saves the data for the audio CODEC power down control. Initially, the power off mode, \( B7 \) is set to '0'. Also, line input, ADC, DAC, and outputs power down had to be deactivated. This is done by setting \( B0 = B2 = B3 = B4 = '0' \).

The corresponding control sequence was set to:

\[
control := '0000110000000010'
\]

The final configuration changes performed for the audio CODEC was the format of the digital audio interface and the data output. Here, left justified mode was chosen together with the 16-bit data output. The corresponding register is \( R7 \), i.e. "000111", where left justified mode was selected by switching the bit \( B0 \) to '1' and value of bit \( B1 \) to '0'. For the 16-bit data output, bits \( B3 \) and \( B2 \) of the same register were switched to '0' and '0', respectively.

The corresponding control sequence was set to:

\[
control := '0000111000001001'
\]

After this the Audio Codec was configured to satisfy the requirement specification given previously in section 1.2.
Consequently, the 'Audio_Codec_Sturcture.vhd' file was written and simulated in the ModelSim. This file reads the bit-serial data from the ADCs and consequently sends the 16-bits parallel digital audio data to the interpolator block and the SRAM memory. Two files build up the "Audio_Codec_Structure.vhd" file and they are "clk_gen.vhd" and "codec_interface.vhd". The 'clk_gen.vhd" generates the required clocks mclk, DA-Clrclk, ADClrclk, belk, while the 'codec_interface.vhd' takes care of the serial data from the ADC converters. When the shift-register is full, i.e. all 16-bits are read to the shift register, audio data is sent out to the interpolator block and SRAM-memory.

All blocks are clocked by the master clock, which is set to 50 MHz. They have also same global reset signal that resets blocks to their default values. The switch button, KEY0 or pin G26, is used for this purpose. This switch is debounced by default so there will be no problems with unwanted glitches. Furthermore, a reset switch provides a low logic level when the switch is pressed and a high logic level when the switch is depressed.

The allocated pin names that were set for the Audio CODEC interface are shown in Table 4.11.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>FPGA pin number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCLRCLK</td>
<td>PIN_C5</td>
<td>ADC left/right clock.</td>
</tr>
<tr>
<td>ADCDAT</td>
<td>PIN_B5</td>
<td>ADC serial data.</td>
</tr>
<tr>
<td>DACLRCLK</td>
<td>PIN_C6</td>
<td>DAC left/right clock.</td>
</tr>
<tr>
<td>DACDAT</td>
<td>PIN_A4</td>
<td>DAC serial data.</td>
</tr>
<tr>
<td>CLK</td>
<td>PIN_A5</td>
<td>Master chip clock.</td>
</tr>
<tr>
<td>BCLK</td>
<td>PIN_B4</td>
<td>Bit-stream clock.</td>
</tr>
</tbody>
</table>

Table 4.11: Allocated Audio Codec pins.
4.10 FPGA Implementation

4.10.4 SRAM memory

The SRAM memory included with the DE2 board is a high-speed asynchronous CMOS static random access memory. This memory is organized as an array of $256K$ words by 16 bits. Furthermore, this circuit has a fully static operation meaning that no clock or refreshing cycle is needed for its proper functionality. In order for data to be stored correctly, the SRAM write cycle switching characteristics must be fulfilled. The corresponding write cycle switching characteristics are shown in Fig. 4.17.

![AC WAVEFORMS WRITE CYCLE NO. 1 (CE Controlled, $OE$ is HIGH or LOW)](image)

Figure 4.17: SRAM write cycle.
The allocated SRAM pin description is shown in Table 4.12:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>FPGA pin number</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0-A17</td>
<td>Address Inputs</td>
</tr>
<tr>
<td>I/O0-I/O15</td>
<td>Data Inputs/Outputs</td>
</tr>
<tr>
<td>CE</td>
<td>Chip Enable Input</td>
</tr>
<tr>
<td>OE</td>
<td>Output Enable Input</td>
</tr>
<tr>
<td>WE</td>
<td>Write Enable Input</td>
</tr>
<tr>
<td>LB</td>
<td>Lower-Byte Control (I/O0-I/O7)</td>
</tr>
<tr>
<td>UB</td>
<td>Upper-Byte Control (I/O8-I/O15)</td>
</tr>
</tbody>
</table>

**Table 4.12**: SRAM pin description.

This component was used to save data both from the interpolator block and the audio interface. Thus, it provides an ability to compare generated interpolator data with the actual input data from the audio interface. Since the interpolator block operates at 32 time higher sampling frequency than the audio CODEC\(^\text{10}\), the data generated by the interpolator block was always written first to the SRAM memory. From previous we know that the audio interface operates at $f_{\text{sample\_codec}} = 50 \, MHz/1024 = 48.828125 \, KHz$, while the interpolator block operates with the $f_{\text{sample\_interpolator}} = 32 \ast f_{\text{sample\_codec}} = 1.5625 \, MHz$ sampling rate. Thus, there will be time instants when data from both blocks have to be written simultaneously into the SRAM memory. This memory write hazard was solved by postponing the write cycle from the audio interface. When the interpolator data is written, only then data from the audio interface is allowed to be written into the memory. Such write operation was possible since there was enough time between two consequent write requests from the interpolator block.

The memory had to be divided into two memory blocks, as two different data sequences are saved into the SRAM memory simultane-

---

\(^{10}\)Oversampling rate should be equal to 32.
4.10 FPGA Implementation

ously. The SRAM had to be divided in a such a way so that sufficiently long data sequences from both blocks were saved. The accumulated data was later used for the frequency spectrum calculation.

The required memory write control signals were generated by the 'SRAM_control.vhdll' file. This file also resolves the memory write hazard as discussed in previous text. The main part in the "SRAM_control.vhdll" file is the Finite State Machine which is often used for the generation of memory control signals for the memory write/read operations. To perform simulations on the stored binary data, Control Panel was used. As describe previously, the Control Panel is included with the DE2 board to facilitate the communication with different board input/output ports.
Chapter 5

Simulation Results

In this section the simulation results are presented. All simulations throughout this section were performed on the saved data from the SRAM memory.

Initially, an audio sinus signal with $1\, KHz$ frequency was applied to the DE2 board. Here, the audio interface converted the audio signal to its corresponding digitized 16-bit data representation. This data together with the data from the interpolator block was stored in the SRAM memory. Later on, the data stored in the SRAM memory was extracted by using the Control Panel. This software allows the user to read the stored data from the memory and to save it to a file. Thus, two files representing data from the interpolator and the audio interface had to be converted from binary to their corresponding numerical representation. An online converter was found to be suitable for the purpose of converting data from the binary to the hexadecimal representation [18]. After this, further data conversion was performed where hexadecimal data was converted to the corresponding bit data representation.

The generated files initially had to be processed to allow proper data loading into the MatLab program. For this Linux bash terminal and programs sed and awk were used. The commands used for this purpose are presented in following text:

```
bash$ sed ’s//g’ < file.txt > /tmp2/file2.txt
```
Simulation Results

```
bash$ awk 'c=split($0, s); for(n=1; n<=c; n+=4) print s[n+1] s[n] s[n+3]' $ 1 < /tmp2/t2.txt > /tmp2/bin.txt
```

Finally, the processed data was loaded into *Matlab* for further data processing.

The Fig. 5.1 illustrates a sinus signal that was extracted from the SRAM memory. This is the data generated by the audio interface. The Fig. 5.2 represents the resulting plot of the interpolator block. From this pictures we can observe that interpolator block generates a sinus signal with higher resolution as it operates at 32 times higher sample frequency than the audio interface.

![Figure 5.1: The output from the audio interface.](image1.png)

![Figure 5.2: The output from the interpolator block.](image2.png)
Figures below represent the corresponding amplitude spectrum of both signals plotted in figures 5.1 and 5.2. The vertical line lines indicates the sample frequencies of $f_{\text{codec}} = 48.828125 \text{ KHz}$ and - $f_{\text{interpolator}} = 32 \times f_{\text{codec}} = 1.5 \text{ MHZ}$.

![Figure 5.3](image1)

**Figure 5.3:** The amplitude spectrum of the sinus signal from the audio interface.

![Figure 5.4](image2)

**Figure 5.4:** The amplitude spectrum of the sinus signal from the interpolator block.

From the amplitude spectrum plots from the Fig. 5.3 and the Fig. 5.4 we can determine that the interpolator block is upsampling an input audio signal 32 times as required by the specification. Thus, the main goal of the requirement specification was fulfilled.
Furthermore, the corresponding SNR values that were calculated on the data from both Audio CODEC and interpolator block are presented in table below.

<table>
<thead>
<tr>
<th>Signal to Noise Ratio</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR prior the interpolator block</td>
<td>60.203848770764203 dB</td>
</tr>
<tr>
<td>SNR after the interpolator block</td>
<td>60.960911023247718 dB</td>
</tr>
</tbody>
</table>

**Table 5.1:** Calculated SNR values.

From the results summarized in the table above we can see that the interpolator block do not contribute with additional noise to already present noise power at the input of the interpolator block. The actual SNR value at the output of the interpolator block is somewhat better then the input SNR value. This improvement in SNR at the output of interpolator block could be explained by the fact that the Audio CODEC do not operate at its default sample frequency. Thus, the ADC digital filters inside the Audio CODEC do not perform filtering properly.
Chapter 6

Conclusion

6.1 Final Thoughts

In this project several optimization techniques have been explored and merged together for the realization of an interpolator circuit. The main goal was to perform an interpolation operation with lowest possible hardware complexity and utilization. Thus, the upsampling factor was deliberately chosen to be a power of two. As a result, half-band FIR filters were used for the realization of image-rejection filters. Thus, half of the filter coefficients did not have to be realized in hardware. Furthermore, by exploiting the half-band FIR filter symmetry only one side of the remaining filter coefficients were realized in the hardware. Furthermore, multiple constant multiplication, \( (MCM) \), algorithm was used for the realization of the filter multipliers. Thus, multiplication operations found inside the half-band FIR filters were realized with the help of an MCM algorithm as a shift and add units. This resulted in further interpolator simplification. The communicator components found in each interpolator stage were realized as a simple two-port multiplexers. Here, five different clock signals had to be generated as they were used as multiplexer select signals.

Thus, the interpolator realization presented in this thesis gives one possible realization when a small implementation area and low power consumption are two of the design requirements. This implementation is naturally only possible when the interpolator factor is a power of two. This is the main requirement as the half-band FIR filters could
be used for the realization of every image-rejection FIR filters.

### 6.2 Further Work

For further work, additional simulations can be performed. Initially, one can calculate noise generated by the audio interface and interpolator block when no signal is applied to the DE2 board. Furthermore, one can calculate noise generated when a sinus signal is applied to the input of the DE2 board. Here, SNR can also be estimated. Finally, additional simulations can be performed on the audio signal where music played from the CD is applied to the DE2 board.

On chip level realization, several possible implementation problems can be found. One of the problems that could influence proper interpolator realization is the use of several clock signals. These clock signals must be generated for selecting the port of interpolator multiplexers. Hence, if interpolator block is to be realized on a single VLSI chip, extra attention to the clock nets must be made. Also, clock skews must be carefully controlled or otherwise system malfunction can occur. On contrary, by having large clock skews different interpolator stages could sample wrong data values. This is naturally undesirable behavior. Furthermore, if clock signals are drawn in close vicinity of each other they must be shielded from each other. Preferably different metals can be used for different clock paths. The clock jitter can further negatively influence proper functionality of the interpolator circuit. Furthermore, analog parts on the chip must be shielded from the clock nets or otherwise they can be influenced by the constant clock nets switching. If analog parts are available on the chip separate analog power source must be available for their power supply.
Bibliography


På svenska

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Uppphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlaget hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet — or its possible replacement — for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Jasko Bajramovic