Σχεδίαση και υλοποίηση ενός 24-bit ψηφιακού σε αναλογικό μετατροπέα βασισμένο στη ΣΔ διαμόρφωση

Τσιλιγιάννης Γιώργος

ΑΜ: 2002030053

Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Οκτώβριος 2008

Εξεταστική επιτροπή: Αναπληρωτής Καθηγητής Πνευματικάτος Δ.

Καθηγητής Δόλλας Α.

Καθηγητής Καλαϊτζάκης Κ.

Η διπλωματική αυτή είναι αφιερωμένη στους γονείς μου. Πολλές ευχαριστίες στον κύριο Διονύσιο Πνευματικάτο, τον κύριο Κωνσταντίνο Παπαδά και τον κύριο Δημήτρη Μητροβγένη για τη βοήθεια που μου παρείχαν κατά τη διάρκεια της εκπόνησης της διπλωματικής μου εργασίας

Περίληψη

Η διπλωματική αυτή εργασία αναφέρεται στη θεωρητική μελέτη, στη σχεδίαση και στην υλοποίηση του

ψηφιακού μέρους ενός μετατροπέα υψηλής ανάλυσης ψηφιακού σε αναλογικό σήμα. Πιο συγκεκριμένα πρόκειται για ένα μετατροπέα, 24-bit με εύρος συχνοτήτων από 0.1 mHz έως 1 KHz με λόγο ισχύος σήματος πρός ισχύ θορύβου (SNR) μεγαλύτερο των 120 dB

Συνήθως, μετατροπείς υψηλής ακρίβειας συναντώνται σε εφαρμογές ήχου με διαφορετικό, όμως, εύρος συχνοτήτων. Σε εφαρμογές μικρότερης ακρίβειας χρησιμοποιούνται αρχιτεκτονικές καθοδήγησης ρεύματος (current steering) ή ανακατανομής φορτίου (charge redistribution). Για την αρχιτεκτονική του μετατροπέα προτιμήθηκε η ΣΔ τοπολογία λόγω των πλεονεκτημάτων που έχει σε σχέση με άλλες. Η υψηλή ανάλυση μπορεί να επιτευχθεί με την αύξηση του αρχικού ρυθμού δειγματοληψίας του σήματος εισόδου και τη διαμόρφωση του θορύβου κβάντισης που προστίθεται από τη ΣΔ διαμόρφωση. Στη συνέχεια, το κβαντισμένο σήμα οδηγεί έναν εσωτερικό μετατροπέα ψηφιακού σήματος σε αναλογικό σήμα λίγων bits. Τέλος, το αναλογικό σήμα φιλτράρεται από ένα χαμηλοπερατό φίλτρο έτσι ώστε να ανακτηθεί το αρχικό επιθυμητό σήμα.

Η υλοποίηση μιας τέτοιας διάταξης μπορεί να χρησιμοποιηθεί σε εφαρμογές όπως γεννήτριες αναλογικού σήματος, μηχανισμούς ανάδρασης για έλεγχο κίνησης ακριβείας π.χ κάποιου ρομποτικού βραχίονα καθώς και πλήθος άλλων εφαρμογών.

Τα βήματα που ακολουθήθηκαν για τη σχεδίαση και την υλοποίηση του μετατροπέα ήταν τα εξής: Αρχικά μελετήθηκε η σχετική βιβλιογραφία καθώς και σχετικές δημοσιεύσεις. Στη συνέχεια, με βάση τις προδιαγραφές του μετατροπέα καθορίστηκε η αρχιτεκτονική και δημιουργήθηκε ένα μοντέλο σε SIMULINK /MATLAB το οποίο να πληρεί τις απαιτήσεις της σχεδίασης. Έπειτα, γράφηκε ο κώδικας σε VHDL (RTL μοντέλο) με βάση το μοντέλο του SIMULINK /MATLAB. Ο έλεγχος για τη λειτουργικότητά του RTL έγινε συγκρίνοντας τα αποτελέσματα της εξομοίωσης του κώδικά της VHDL με αυτά του SIMULINK/MATLAB. Τέλος, έγινε σύνθεση σε FPGA προκείμενου να επιβεβαιωθεί η σχεδίαση και να γίνει εκτίμηση της πολυπλοκότητας. Παρακάτω αναφέρονται συνοπτικά τα μέρη στα οποία έχει χωριστεί η σχεδίαση.

Ο μετατροπέας χωρίζεται σε δύο μέρη: το ψηφιακό και το αναλογικό. Το ψηφιακό μέρος έχει χωριστεί περεταίρω σε τρία εσωτερικά μέρη: τον παρεμβολέα (interpolator), τον ΣΔ διαμορφωτή (modulator) και τον αλγόριθμο DWA.

Ο interpolator είναι το πρώτο μπλοκ και πραγματοποιεί υπερδειγματοληψία στο αρχικά δειγματοληπτημένο σήμα στα 6 KHz κατά 256. Κατά την υπερδειγματοληψία, δημιουργούνται αντίγραφα της εικόνας του φάσματος του σήματος εισόδου στο σήμα εξόδου. Για την αφαίρεση αυτών των εικόνων χρησιμοποιούνται χαμηλοπερατά φίλτρα. Όπως θα γίνει κατανοητό στη συνέχεια, η αρχιτεκτονική των πολλαπλών σταδίων (multistage) είναι προτιμότερη γιατί καταλήγει σε φίλτρα χαμηλότερης τάξης και άρα χαμηλότερης υπολογιστικής πολυπλοκότητας.

Με βάση τα προηγούμενα ο interpolator διαιρείται σε τέσσερα στάδια κάθε ένα από τα οποία περιέχει ένα φίλτρο. Τα στάδια που συνθέτουν τον interpolator πραγματοποιούν υπερδειγματοληψία κατά x2, x2, x2 και x32 . Προκειμένου το σήμα εξόδου να έχει γραμμική φάση, τα φίλτρα είναι πεπερασμένης κρουστικής απόκρισης (FIR). Πιο συγκεκριμένα στο πρώτο στάδιο χρησιμοποιείται ένα ισοκυματικό (equiripple) FIR φίλτρο 44ης τάξης. Στο δεύτερο στάδιο και στο τρίτο στάδιο χρησιμοποιούνται Half Band φίλτρα 30ης και 18ης τάξης αντίστοιχα και τέλος χρησιμοποιείται ένα SINC φίλτρο δεύτερης τάξης. Για την ελαχιστοποίηση των υπολογιστικών πράξεων που εκτελούνται στα φίλτρα χρησιμοποιείται η αρχιτεκτονική των πολυφασικών συνιστωσών και η λογική του πολλαπλασιαστή-συσσωρευτή (MAC).

Η έξοδος του intrepolator τροφοδοτείται στην είσοδο ενός ΣΔ διαμορφωτή 3ης τάξης που είναι το επόμενο μπλοκ. Το σήμα εισόδου του διαμορφωτή είναι 24-bit, ενώ η έξοδος είναι το κβαντισμένο σήμα των 5-bit. Η ΣΔ διαμόρφωση αξιοποιεί την υπερδειγματοληψία για να “απλώσει” την ισχύ του θορύβου κβάντισης σε μεγαλύτερο εύρος συχνοτήτων και στην συνέχεια να τον διαμορφώσει κατάλληλα ώστε να βρίσκεται εκτός της επιθυμητής ζώνης συχνοτήτων. Στην πραγματικότητα, πρόκειται για ένα είδος υψιπερατού φίλτρου για το θόρυβο, ενώ το σήμα εισόδου εμφανίζεται στην έξοδο με μια μικρή καθυστέρηση.

Για την αποφυγή προβλημάτων αστάθειας που εμφανίζονται σε αυτή την αρχιτεκτονική προτιμήθηκε η λύση ενός κβαντιστή πολλών επιπέδων.

Τέλος, πρέπει να αναφέρουμε ότι κατά τη διάρκεια της διαμόρφωσης, μπορεί να δημιουργηθούν ανεπιθύμητοι τόνοι στο φάσμα των συχνοτήτων του μετατροπέα που μας ενδιαφέρει. Οι τόνοι αυτοί είναι αποτέλεσμα περιοδικότητας στην έξοδο η οποία δημιουργείται είτε με μια σταθερή είσοδο, είτε με είσοδο η οποία έχει πολύ μικρή συχνότητα. Για να αποφύγουμε τη γέννηση τέτοιων παλμών προσθέτουμε στο σήμα εξόδου πριν το στάδιο της κβάντισης, ένα ψευδό-τυχαίο σήμα ώστε να αναιρέσουμε τη περιοδικότητα που εμφανίζεται στην έξοδο. Το σήμα αυτό δημιουργείται χρησιμοποιώντας ένα Γραμμικό Αναδρομικό Καταχωρητή Ολίσθησης 35 bit..

Το μπλοκ του διαμορφωτή αποτελείται από δύο ολοκληρωτές που δεν εισάγουν καθυστέρηση στο σήμα εισόδου και ένα ολοκληρωτή που εισάγει έναν κύκλο καθυστέρησης. Η τοπολογία του

διαμορφωτή είναι της ανάδρασης του λάθους κβάντισης κατά την οποία η έξοδος του κβαντιστή εκτεταμένη κατά 19 bit, προστίθεται μαζί με το σήμα εισόδου κάθε ολοκληρωτή. Στις αριθμητικές πράξεις του διαμορφωτή, χρησιμοποιείται λογική πρόβλεψης υπερχείλισης ώστε να μην έχουμε εισαγωγή λάθους.

Στο τελευταίο στάδιο του μετατροπέα, υλοποιείται ένας αλγόριθμος Δυναμικής Αντιστοίχησης Στοιχείων. Με τον αλγόριθμο αυτό, πετυχαίνουμε το περιορισμό του λάθους που εισάγεται στο αναλογικό σήμα εξόδου εξαιτίας της διαφοράς μεταξύ των στοιχείων αντιστοίχησης (είτε πηγές ρεύματος είτε πυκνωτές) που υλοποιούν το αναλογικό μέρος.

Ο αλγόριθμος που χρησιμοποιείται στο συγκεκριμένο μετατροπέα, είναι αυτός του Μέσου Όρου Βάρους Δεδομένων (Data Weighted Averaging). Στον αλγόριθμο αυτό το σήμα εξόδου μετατρέπεται από ένα σήμα 5-bit σε ένα σήμα 32-bit με βάση τη κωδικοποίηση θερμομέτρου. Ο αλγόριθμος αυτός δουλεύει ως εξής: κάθε φορά που υπάρχει κάποια νέα είσοδος, ένας δείκτης δείχνει τη θέση του πρώτου στοιχείου που θα ενεργοποιηθεί ενώ το πλήθος των στοιχείων εξαρτάται από τη κωδικοποίηση θερμομέτρου της τρέχουσας εισόδου. Στην επόμενη είσοδο, ο δείκτης δείχνει ως πρώτο στοιχείο το επόμενο μετά το τελευταίο στοιχείο της προηγούμενης εισόδου. Με αυτό τον τρόπο το λάθος μετά από διαδοχικές εισόδου μειώνεται.

Abstract

In this thesis, the design of a ΣΔ Digital to Analog Converter is presented. The concept of the thesis is presented, the theory of operation is explained, high level models are developed, the architecture of

the digital part and the development procedure is described.

This document has the following structure: In *Chapter 1,* an introduction to ΣΔ converters is given. The State-of-the-Art of ΣΔ converters is presented and the Top Level Architecture of the DAC is described. *Chapter 2* contains the signal processing theory of the interpolation block taking into consideration performance specifications, along with RTL implementation and corresponding verification of the results. *Chapter 3* describes the concept of ΣΔ modulation along with RTL implementation and corresponding verification of the block.. In *Chapter 4* the analysis of the Dynamic Element Matching algorithm (DWA) along with the RTL implementation and verification of the block are exposed. In *Chapter 5* the synthesis procedure to the FPGA device is described. Finally, in *Chapter 6* a conclusion is made for the work performed and future extensions of this project are mentioned.

Table of Contents

1 Introduction

This thesis concerns the implementation of the digital part of a 24-bit high resolution Digital to Analog Converter at 1 KHz with a signal to noise ratio greater than 120 dB. The content of the thesis was chosen because of the small number of applications with similar performance. For this activity, the ΣΔ architecture was preferred, because of its inherent advantages against other architectures like current steering topologies which are not well suited for high resolution converters. More specifically, high resolutions can be achieved by combining oversampling of the input signal and shaping of the quantization noise inherent in the ΣΔ modulator.

In order to improve the stability margin and relax the requirements of the analog post filter, the converter uses the multibit ΣΔ topology. A multibit architecture implies using an internal DAC which has though the disadvantage of mismatch error in the analog part. Therefore, a Dynamic Element Matching algorithm is used to reduce these errors and achieve high dynamic performance.

For the design and implementation of this thesis, a certain procedure was followed. In the beginning, the proper literature and several publications were studied. Thereafter, a SIMULINK/MATLAB theoretical model was designed to follow the converters specifications. Then the RTL model was implemented in order to follow the SIMULINK model. The functionality of the RTL model was verified by comparing the results with those of the SIMULINK model. Finally, the RTL model was synthesized to a Cyclone-II FPGA of Altera.

1.1 State of the Art

This chapter focuses on the State-OfThe-Art of the major parts of a ΣΔ converter that includes an interpolation stage, a noise shaping loop (ΣΔ modulator), and a Dynamic Element Matching algorithm.

1.1.1 Interpolation

As far as the interpolation stage is concerned, several criteria have to be taken into consideration. These criteria depend on the target implementation specifications. The basic trade-offs for the design of the interpolator are the area of the design, the speed and the performance, the type of filtering, the upsampling factor, that are unique for each application. The arithmetic representation of the filter will determine the accuracy when the filter is implemented in hardware

One of the most important parameters determining the complexity of the filters is the order and type of the filter. The order of the filter is mainly affected by the upsampling factor and the bandwidth. When the upsampling factor is high, the order of the interpolation filters increases. Therefore, multistage architectures are preferred in most applications to decrease the order of the filters with the benefit of better performance and smaller complexity.

A parameter that also affects the design of the interpolator is the type of filters that have to be used in each stage. The filters can be either FIR or IIR. In applications such as high quality digital audio, where linear phase and stability are important, FIR filters are mainly used. However, FIR filters have the disadvantage that they need bigger number of taps than IIR do, to maintain the same required pass-band ripple and stop-band attenuation.

Examples of implementations taking into considerations the above issues can be found on literature.

1.1.2 ΣΔ modulation

The specifications of the converter, i.e sampling frequency,bandwidth, and dynamic performance specify the oversampling ratio, the order of the modulator and the number of quantization levels.

In several applications where ΣΔ modulation is used, many different topologies of the ΣΔ modulator exist each one of them offering different advantages.

One of the fundamental consideration for the design of the ΣΔ modulator is, if the quantization is single bit or multi-bit. Single bit modulators have the advantage of of low complexity, low cost implementation and offer perfect linearity. Unfortunately, single bit modulators have reduced dynamic performance for a certain oversampling ratio and order. Even by raising the order of the single bit modulator stability problems will exist and the shaping of quantization noise will be quite poor. On the other hand, multibit modulators can achieve bigger stability margin for the same oversampling ratios than the single bit modulators, but they may suffer from non linearities. Despite their increased complexity, they are usually preferred

Another issue concerning the ΣΔ modulators is the order and the structure of the modulator. By raising the order, the quantization noise shaping is improved but the modulator may experience overflow of the quantizer, which results in oscillations. In practice, the stability of the modulator depends on the

order, the range of the input signal and the resolution of the quantizer.

A ΣΔ modulator can follow either the error feed forward and feedback structure or the cascade structure.

The error feedback loop topology can be implemented by a chain of integrators with a distributed feedback of the quantization error. This topology can be easily altered by placing bias at the feedback network or distributing selectively the feedback. Error feedback topologies may experience stability problems when the level of the quantizer is small. The advantage of these topologies is high SNR.

On the other hand, MASH structures are an alternative solution when high order of error shaping is required without stability problems. The basic idea of a MASH topology is the cascade of ΣΔ modulators of smaller order. This topology increases the complexity of the analog design and may suffer from filtered noise leakage between the cascaded modulators. To shape this error, MASH architectures require the use of extra analog processing. In Figure 1.1 both the error feedback loop and MASH topologies are displayed.

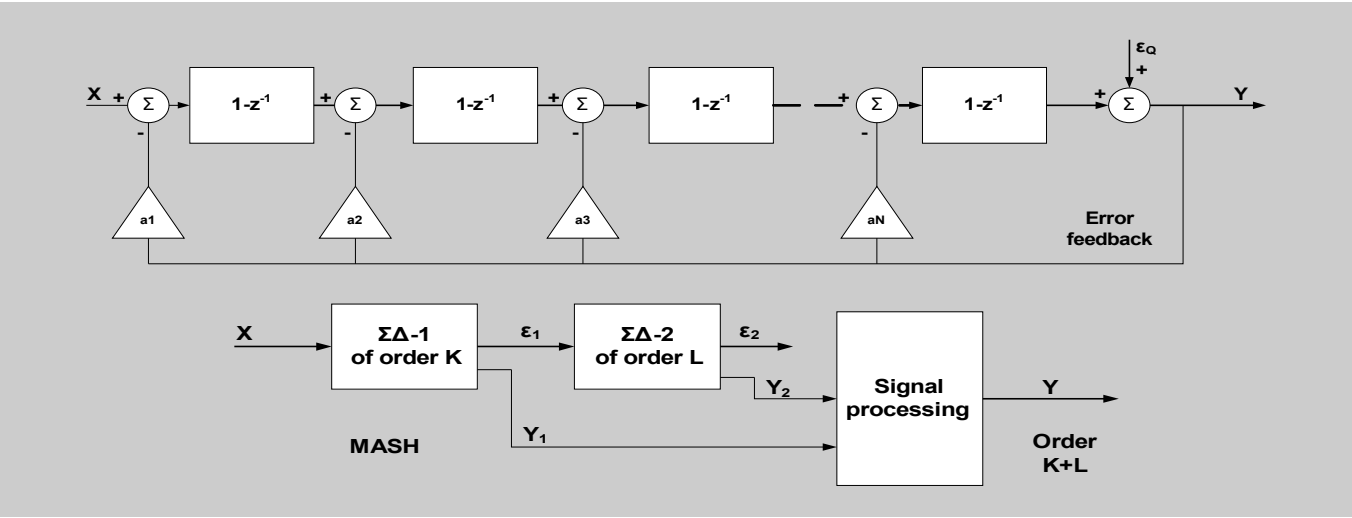

Figure 1.1: Error feedback and MASH architecture topologies

Both MASH and error feedback structures are used in ΣΔ modulators, depending on the specifications. The choice of the type of ΣΔ modulator is easier for a DAC than for an ADC, because complex transfer functions can be implemented with high accuracy.

1.1.3 Dynamic Element Matching

As described before the analog part suffers from non-linearity due to the element mismatch. The Dynamic Element Matching algorithm is a fundamental part of the DAC and is used to correct the mismatch of the analog part when the latter is implemented in silicon.

Architectures based on current steering or charge redistribution are commonly used for the internal DAC. The thermometer code or the binary code topology can be chosen depending on the application specifications. The binary topology is preferred when the resolution of the quantizer is high and the simplicity of the implementation is critical. Thermometer code topologies on the other hand are preferred when the linearity of the output is important. For more special applications, segmented topologies combining both thermometer and binary code are used.

For the thermometer topology, there are many algorithms used to increase the matching of the analog elements. Depending on the complexity limitations, there are several types of dynamic element matching algorithms, such as the butterfly randomization, the Individual Level Averaging (ILA), the Galton tree structure, the Data Weighted Averaging (DWA) and others.

The most performant DEM technique is the DWA algorithm along with its alternative versions, such as the rotated and the partial DWA.. The DWA algorithm is preferred because the performance of the algorithm is promising and the complexity is small. DWA contributes in the element mismatch error shaping better than any other algorithms.

1.2 Top level architecture

The converter shall achieve 130 dB signal to noise ratio for signals from 0.1 mHz up to 1kHz. The sampling rate is performed at 6 kHz and the resolution of the input is 24-bit. The oversampling ratio is 256 that results in a master clock frequency of 1.536 MHz .

The DAC is divided in two main blocks: the digital part and the analog part. The digital part is divided into three major internal blocks: the interpolator, the ΣΔ modulator and the DWA block and the analog part comprises the internal DAC. The top level schematic of the DAC as well as the I/O interface of each block are displayed in Figure 1.2 and Table 1.1 respectively.

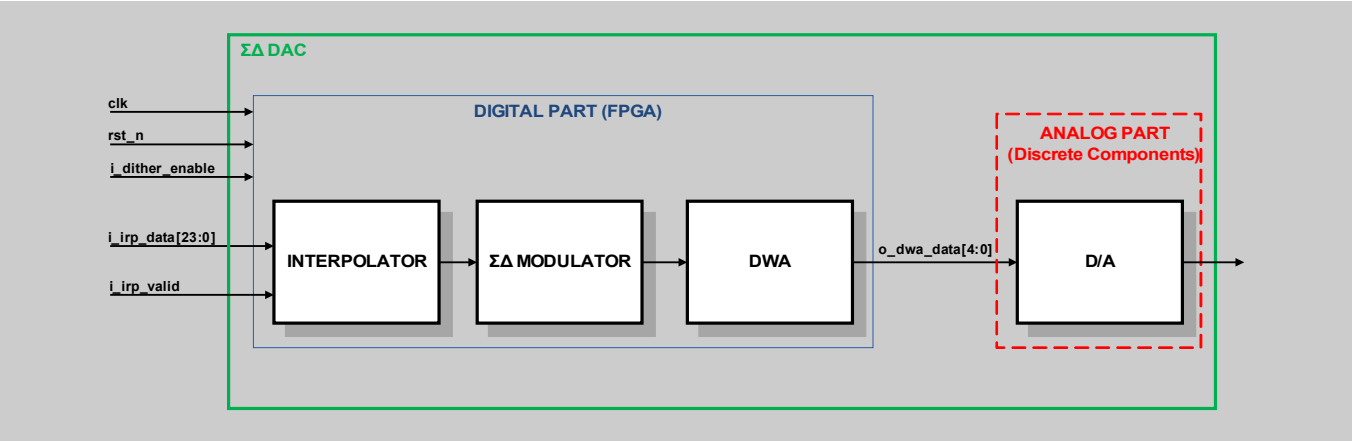

Figure 1.2: Top level schematic

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock at 1.536 MHz
rst_n	in	Active low asynchronous reset
Digital Part		
i_dither_enable	in	Dither enable control signal from external block
i_irp_data (23:0)	in	24-bit input data
i_irp_valid	in	Indicates a new sampled 24-bit input data at 6 KHz
o_dwa_data(31:0)	out	32-bit data output
Analog Part - DAC		
o_dwa_data(31:0)	in	32-bit input data from DWA block
i_out	out	Output current

Table 1.1: ΣΔ DAC Pin List

The ΣΔ DAC as shown in the top level schematic diagram, receives five input signals. The clk and rst_n signals are fed to all the components of the digital part. The i_dither_enable signal is forwarded to the ΣΔ modulator block and the i_irp_valid and i_irp_data are the interpolator inputs.

The first stage of the chain performs upsampling of the initial signal. The basic functionality of the interpolator is the insertion of zero value samples between two consecutive samples of the input signal. Because of image replicas of the initial input signal when upsampled, the interpolator will have to suppress them by means of digital filtering. The interpolator block includes a multistage filtering topology. The output of the interpolator as displayed in Figure 1.2 is fed to the ΣΔ modulator.

The next stage of the converter performs ΣΔ modulation. The ΣΔ modulator block basically shapes the noise introduced in the loop by quantizing differential signals. Internally, in the ΣΔ modulator block, a pseudo-random signal generator Is used, which is controlled externally, used for removing unwanted frequency tones. The quantized output of the ΣΔ modulator block is fed to the DWA block which is the last component of the digital part. This block implements a dynamic element matching algorithm preventing the appearance of mismatch errors in the analog signal. The DWA block codes the quantized output of the modulator in thermometer format and performs cyclic rotation of the elements used in the analog part in order to shape the mismatch error.

For the simulation and demonstration of the converter's functionality, an external analog part is implemented which is connected with the FPGA output. The analog block includes a number of current steering sources driven by each bit of the DWA output. The summation of the currents is the final

analog output which will be displayed in a waveform monitor.

1.3 References

[1] Multirate Digital Signal Processing, Ronald E. Crohiere, Lawrence R. Rabiner, 1983, p129-181

[2] Delta-Sigma Data Converters: Theory, Design and Simulation, Steven R. Norsworthy, Richard Schreier, Gabor C. Temes, First edition, 1996, p: 309-316,406-446

[3] *Data Converters*, Franko Maloberti, First edition, 2007, p: 1-73, 253-298,374-391

[4] *Analog Integrated Circuit Design*, David Johns, Ken Martin, First edition, 1996, p: 531-551

[5] A Multibit Delta-Sigma Audio DAC with 120dB Dynamic Range, Ichiro Fujimori, Tetsuro Sugimoto, IEEE Journal of Solid-State Circuits, VOL.35 NO 8, August 2000.

[6] A 14-bit, 10-Msamples/s DAC Converter Using Multibit ΣΔ modulation, Katayoun Falakshahi, Chih-Kong Ken Yang, Bruce A. Wooley, IEEE Journal of Solid-State Circuits, VOL.34, NO 5 May 1999.

[7] Principles of Data Conversion Systems, Behzad Razavi, IEEE Press p. 45-63.

2 Interpolator

The interpolator is the stage before the ΣΔ modulator and its functionality is to increase the signal rate and filter the image replicas that occur because of upsampling. This chapter describes the principles of the interpolation stage. The background theory of interpolation is presented along with different filter architectures, simulations and implementation considerations.

2.1 Interpolation theory

The benefit of ΣΔ modulation is that it shapes the quantization noise of an oversampled signal to higher frequencies. In order to increase the rate of the signal, an interpolation stage has to be used. Digital data initially sampled at a low rate are interpolated and then fed to the ΣΔ modulator.

The interpolation procedure combines signal upsampling, as well as filtering of the upsampled signal, which contains image replicas of the initial signal. These images need to be removed. The suppression of the images is possible by using low pass filtering. Figure 2.1 Illustrates these two stages.

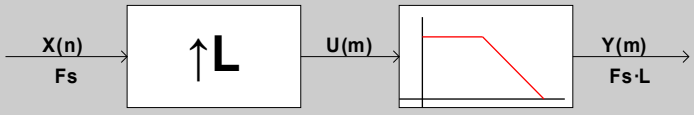

Figure 2.1: Upsampling and filtering

The input signal X(n) initially sampled at Fs is upsampled by L. This means that (L-1) zeros will be inserted between two successive samples resulting in signal U(m) with frequency Fs·L. This signal is then processed by a filter with a low pass characteristic.

2.1.1 Upsampling

The frequency of a signal can be increased by inserting zero values between two successive samples. When upsampling by L, (L-1) zeros are inserted between two successive samples. This can be described by the following equation:

$$U(m)=\begin{cases} x(\frac{m}{L}), & m=0,\pm L,\pm 2L,... \\ 0, & otherwise \end{cases} \quad (1)$$

Figure 2.2 depicts an example of a sinusoidal signal initially sampled at frequency Fs and then upsampled by x2. After the upsampling the frequency of the signal is Fs*L where L=4.

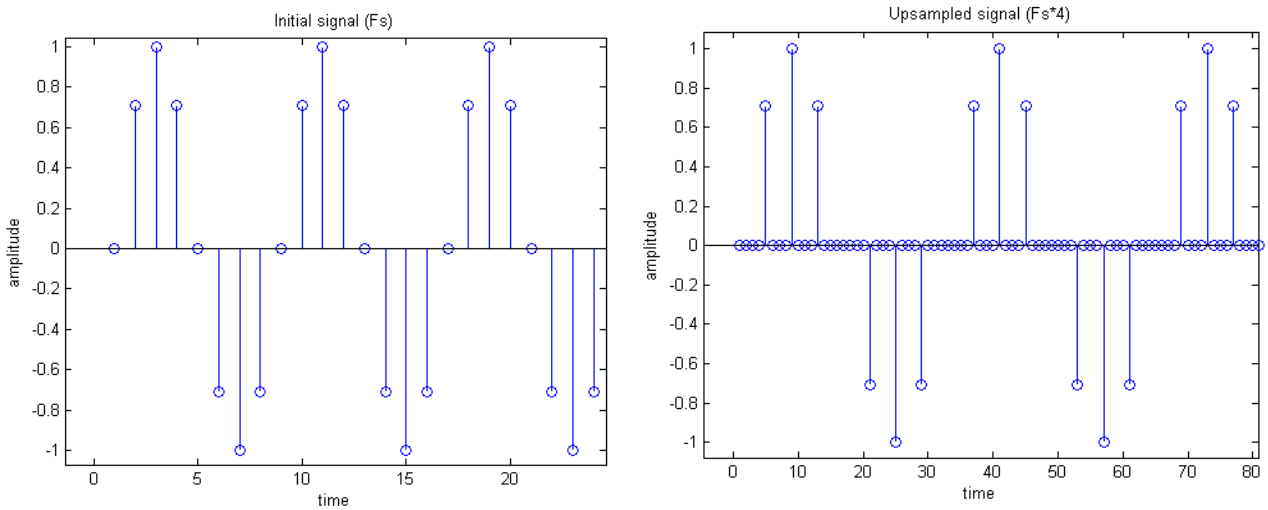

Figure 2.2: Input signal and upsampled signal for L=4

At the frequency domain the upsampled signal will have (L-1) replicas of the spectrum of the initial signal located at multiples of Fs up to (L-1)Fs that have to be removed. Figure 2.3 depicts the power spectrum of the signal shown in Figure 2.2 before and after the upsampling.

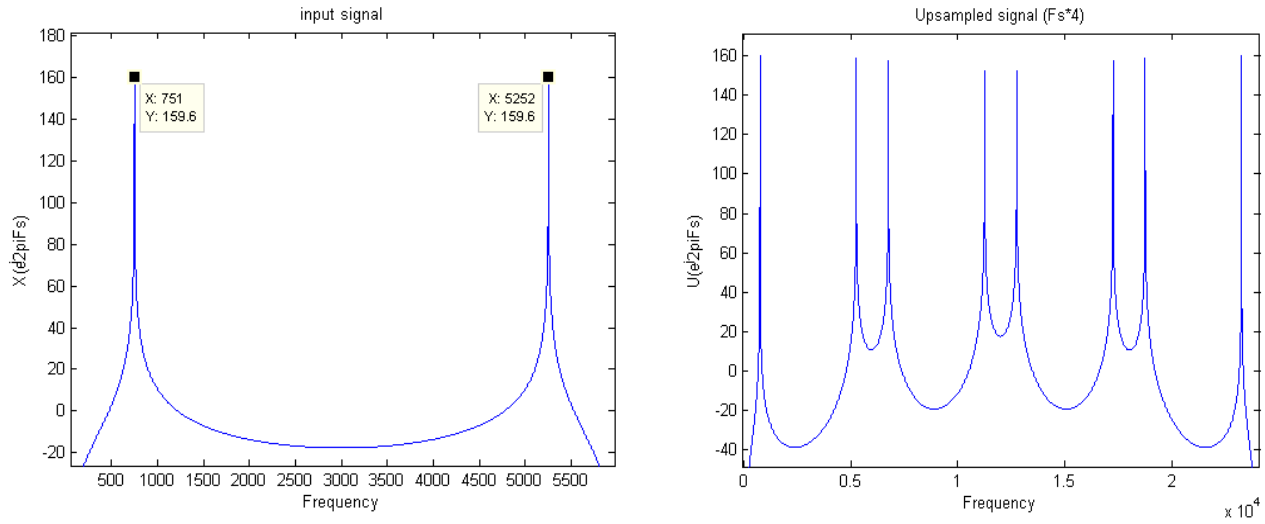

Figure 2.3: Spectrum of input and upsampled signal

The initial signal has a spurious at f_o=750 Hz and its symmetric is located at $L*F_s-f_o$.

Equation (2) gives the location of images for different values of k.

T: Sampling period

Xc: Fourier transform of continuous time signal

ω: Analog frequency

$$X(e^{j\omega})=\frac{1}{T}\sum_{-\infty}^{\infty}X_c\left(j\left(\frac{\omega}{T}-\frac{2\pi k}{T}\right)\right) \quad (2)$$

2.1.2 Low pass filtering

To obtain the desired interpolated signal the image replicas have to be suppressed. This is possible by using low pass filtering which results in the final interpolated version of the initial signal. The filter should be designed in order to preserve the band of interest and suppress the undesired images.

Figure 2.4 depicts the power spectra of the upsampled signal of Figure 2.3 after passing through a low pass filter.

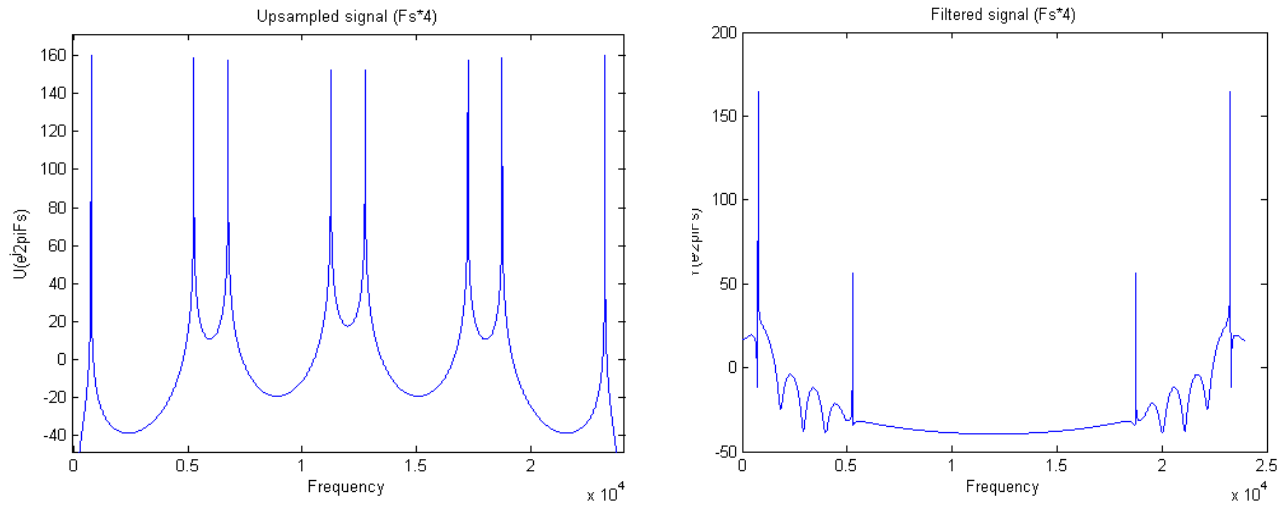

Figure 2.4: Magnitude response of the upsampled and filtered signal

2.1.3 Evaluation of different filters

Two major families of filters can be considered: FIR and IIR. In the case of an FIR filter, the output depends only on the current and previous values of the input signal. The equation describing FIR filters is:

$$y(n)=b_0 x(n)+b_1 x(n-1)+...+b_M x(n-M) \quad (3)$$

where M is the order of the FIR filter and b_i are the coefficients of the filter for i=0,...,M.

On the other hand, in the case of an IIR filter, the output depends on the current and previous values of the input and previous values of the output. The equation describing IIR filters is:

$$y(n)=b_0 x(n)+b_1 x(n-1)+...+b_M x(n-M)+a_1 y(n-1)+...+a_M y(n-M) \quad (4)$$

where b_i and a_i are the coefficients of the filter and M is the order of the IIR filter.

The basic advantage of FIR is that FIR can achieve linear phase (the phase response of the filter is a linear function of the frequency), while IIR filters can only approximate it. However, in general IIR results in lower order filters for the same requirements.

The previous equations result in filters of different complexity. In a hardware implementation this is translated in computational overhead, increase of area and power consumption. However, IIR filters require higher accuracy, which is translated to more bits for the representation of the coefficients because they need to overcome the round-off noise and guarantee stability. In the next section, subcategories of FIR and IIR filters will be presented along with their main attributes.

2.1.3.1 IIR

There are several types of IIR filters that could be used at the design of a low pass filter, such as Chebyshev type-I and II, Butterworth and Elliptic. The basic properties of those filters will be presented.

Chebyshev type-I have equal ripple behavior in the pass-band and Chebyshev type-II have equal ripple behavior in the stopband. Both Chebyshev type-I and type-II filters provide the smallest step response settling time of the IIR filters considered here.

Butterworth filters have the most flat response in the pass-band among their counterparts, which is translated in the smallest possible ripple. The price paid for using this filter, will be slower roll-off which will result to higher order to meet a specific stopband requirement. Butterworth filters result in higher order comparing with the other IIR filters.

Elliptic filters have equiripple behavior, which means that the ripple in both passband and stopband is independently adjustable. This property makes the elliptic filter to have faster transition in gain between both bands.

2.1.3.2 FIR

The output of the FIR filters depends on the current and previous values of the input signal. Their basic feature is that they have linear phase in the passband. Moreover, they are stable but they need very high orders to meet the specifications. FIR filters which have the property of linear phase have symmetric coefficients which usually results in optimized topologies. There are many types of FIR filters but Half Band filters and equiripple filters will be considered here among others.

Half Band filters have their passband and stopband symmetric around Fs/4 where Fs is the desired rate after the interpolation. The beginning of the stopband and the end of the passband are equally located around Fs/4. Half band filters have all even coefficients equal to zero except the central one h(0) which is 0.5. This results in reduced complexity since the multiplications with the zero coefficients are omitted. Half Band filters use the Parks-McLellan algorithm, for the calculation of the coefficients.

Equiripple (or Kaiser) filters have the smallest deviation from the ideal filter when compared to other FIR filters of the same order. This type of filter is suitable for achieving minimum ripple in the passband.

In Table 2.1 and Table 2.2 different types of IIR and FIR are compared respectively for the same specifications (passband and stopband frequency, passband ripple, stopband attenuation). The filters are compared for a x256, x128 and x2 interpolation factor.

IIR type	Initial sampling rate (KHz)	Interpolation	Fs (kHz)	Fpass (kHz)	Fstop (kHz)	Passband ripple (dB)	Stopband attenuation (dB)	Order
Butterworth	6	256	6*256	1.01	3	1	130	15
Chebyshev	6	256	6*256	1.01	3	1	130	10
Elliptic	6	256	6*256	1.01	3	1	130	7
Butterworth	6	128	6*128	1.01	3	1	130	15
Chebyshev	6	128	6*128	1.01	3	1	130	10
Elliptic	6	128	6*128	1.01	3	1	130	7
Butterworth	6	2	6*2	1.01	3	1	130	12
Chebyshev	6	2	6*2	1.01	3	1	130	9
Elliptic	6	2	6*2	1.01	3	1	130	7

Table 2.1: Comparison of IIR filters for different upsampling factors

FIR type	Initial sampling rate (KHz)	Interpolation	Fs (kHz)	Fpass (kHz)	Fstop (kHz)	Passband ripple (dB)	Stopband attenuation (dB)	Order
Equiripple	6	256	6*256	1.01	3	1	130	3026
Half Band	6	256	6*256	1.01	766	1	130	6
Equiripple	6	128	6*128	1.01	3	1	130	1513
Half Band	6	128	6*128	1.01	384	1	130	6
Equiripple	6	2	6*2	1.01	3	1	130	22

Table 2.2: Comparison of FIR filters for different upsampling ratios

The order N of a low pass FIR filter can be approximated by equation (5).

$$N \approx \frac{D_\infty(\delta_p, \delta_s)}{\Delta F / f} \tag{5}$$

where :

$$D_\infty(\delta_p, \delta_s) = \log_{10}\delta_s[a_1(\log_{10}\delta_p)^2 + a_2\log_{10}\delta_p + a_3 + a_4\log_{10}\delta_p + a_4\log_{10}\delta_p + a_6] \tag{6}$$

where:

f :is the sampling frequency at which the filter is referred

ΔF: is the difference between the stopband and the passband frequency of the filter

δ_p and δ_s are the required ripples in the passband and stopband respectively in linear scale

and a_1=0.005309, a_2=0.07114, a_3=-0.4761, a_4=0.00266, a_5=-0.5941 and a_6=0.4278.

From (5) we can conclude that the order of the filter increases, while the transition zone ΔF is shortened. This explains why a multistage implementation by cascading interpolation stages is preferred.

Therefore, in order to reduce the complexity of the filter, a multistage implementation of the interpolator with multiple filters of low order is preferred.

2.1.3.3 SINC filter

The SINC (Sinus Cardinal) filter is a special case of FIR filters, which has certain attributes that have to be discussed. It is a practical solution for linear interpolation. In practice, SINC filters will reject the images created by the upsampling process, by letting the zeros of the transfer function to match with the images that have to be suppressed. The transfer function of a SINC filter is given by the equation :

$$H(z)=\left(\frac{1}{M}\frac{1-z^{-M}}{1-z^{-1}}\right)^{K} \tag{7}$$

where K is the order of the filter and M is the order of the interpolation. The frequency response of the filter is given by equation (8).

$$|H(e^{j\omega})|=\left[\frac{1}{M}\frac{\sin(\omega/2)}{\sin(\omega/2\mathrm{M})}\right]^{K} \tag{8}$$

where

$$\omega=2\pi \mathrm{f}/F_s \tag{9}$$

with Fs being the initial sampling frequency. The images of the interpolation filter are located at $k\cdot Fs\pm f_0$ where f_0 is the signal bandwidth.

The model shown in Figure 2.5 emulates a SINC filter.

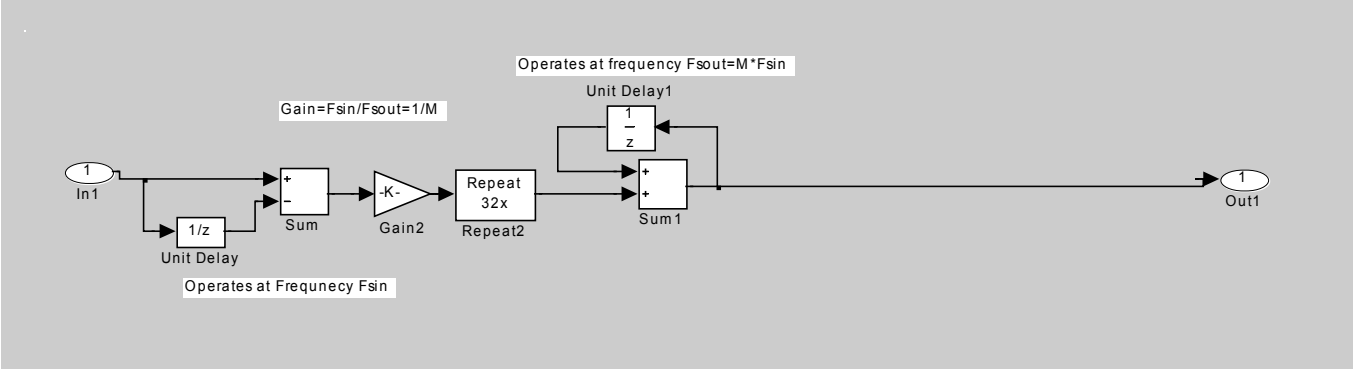

Figure 2.5: SINC filter of oversampling ratio x32 and 2nd order

When applying a sinusoidal signal to a SINC filter and a x4 upsampling is used, we can see that 3 samples are inserted between two successive samples of the initial signal.

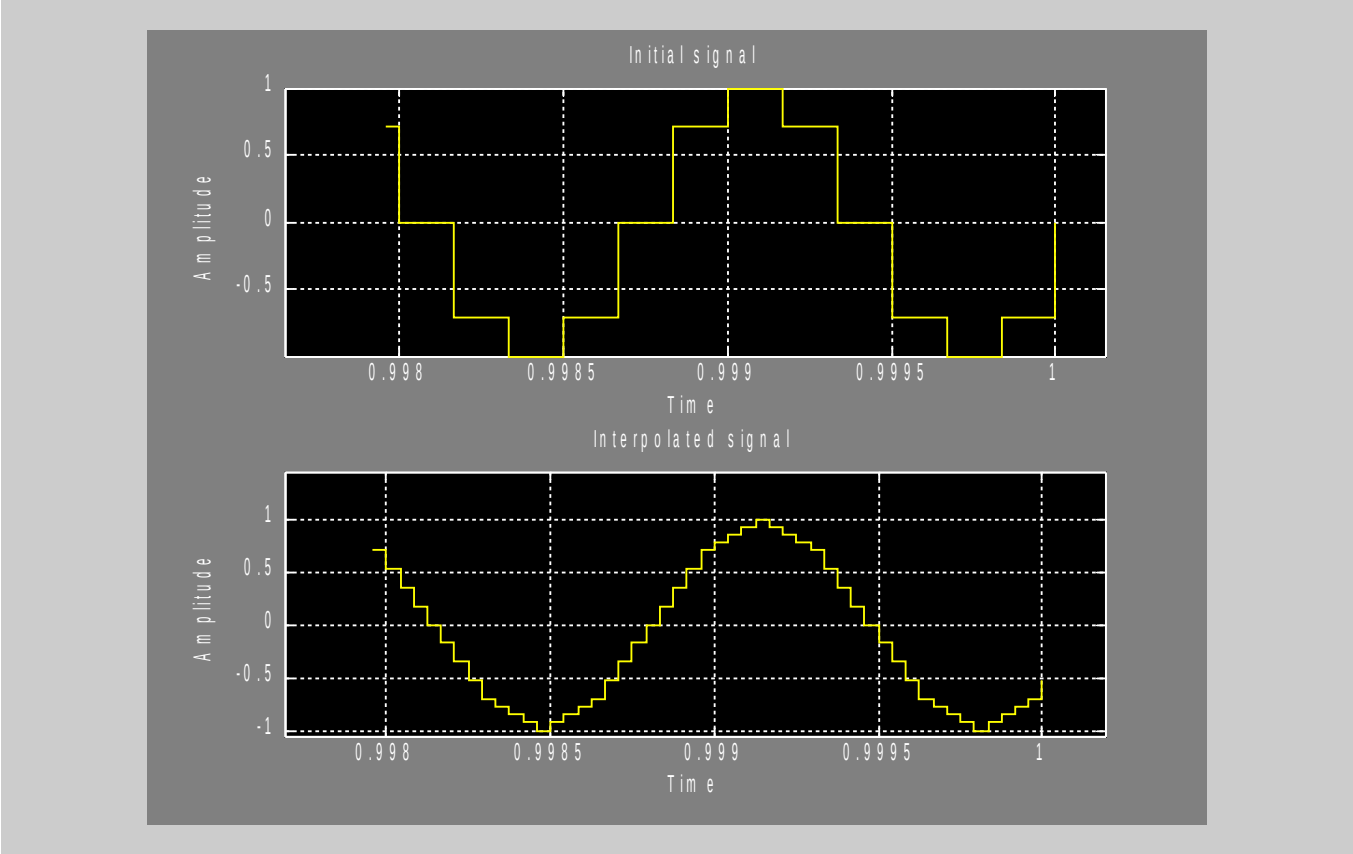

Figure 2.6: Linear interpolation by x4

The magnitude response of a x32 filter is shown in Figure 2.7. We can observe that although the images are not sufficiently attenuated, they are out of the band of interest.

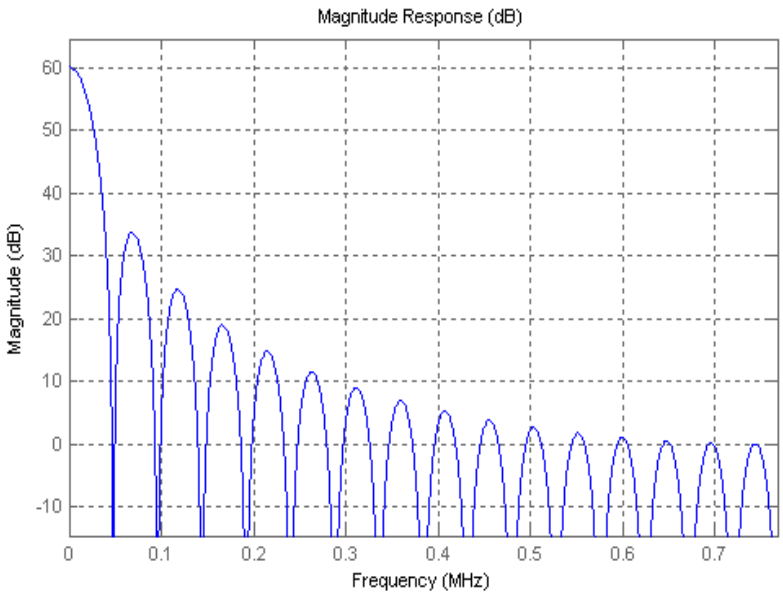

Figure 2.7: CIC filter Frequency response

The impulse response of the CIC filter is shown in Figure 2.8.

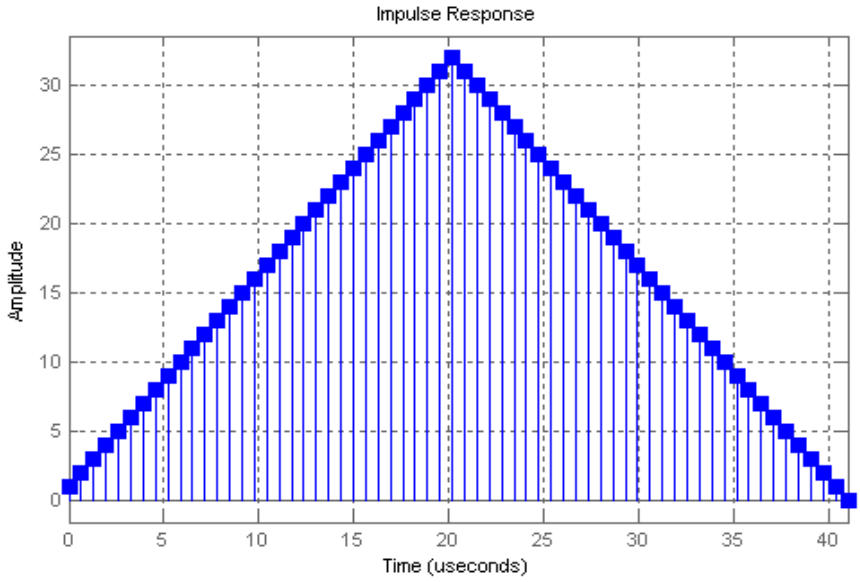

Figure 2.8: Impulse response of CIC Filter

2.1.4 Partitioning

In order to avoid complex implementations in terms of computation and power consumption the multistage approach is preferred.

The interpolation procedure will be divided in more than one stages. This means that we have to cascade upsamplers and low pass filters. The product of the interpolation factors (Li) of each interpolation sub-block has to be equal to the desired oversampling ratio (L). This partitioning is shown in Figure 2.9.

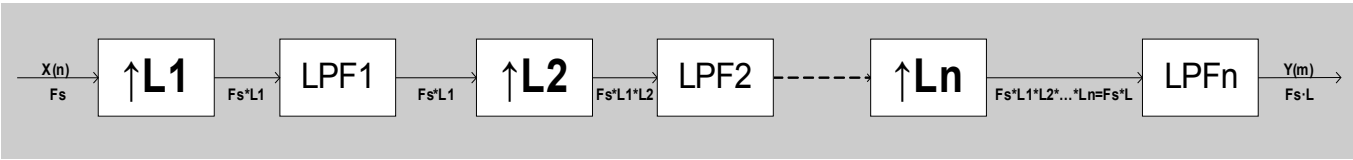

Figure 2.9:Multistage conversion

2.1.5 Polyphase Architecture

The implementation of the interpolation stage can be facilitated by using a polyphase implementation. An example illustrating how the polyphase topology works in case of interpolation by x2 is presented: The transfer function of a third order (3rd) filter is given by:

$$H(z)=h(0)+h(1)z^{-1}+h(2)z^{-2}+h(3)z^{-3} \tag{10}$$

The input signal x(n) is sampled at Fs. Then it is upsampled by x2, which means that a zero is inserted between two successive samples. The upsampled signal is w(m) at 2·Fs. The same holds for y(m) as illustrated in Figure 2.10. The filtered output is then:

$$y(m)=h(0)w(m)+h(1)w(m-1)+h(2)w(m-2)+h(3)w(m-3) \tag{11}$$

where h(i), i=0,...,3 are the coefficients of the filter.

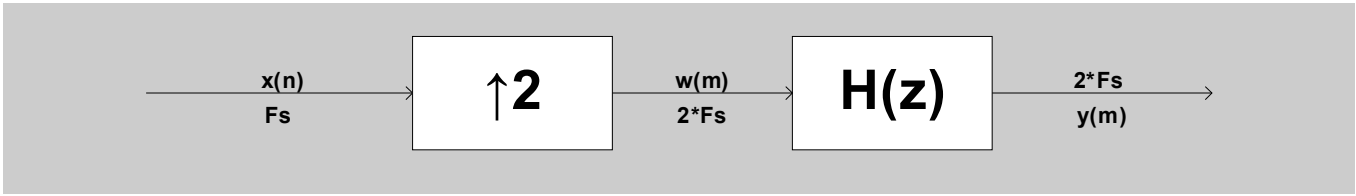

Figure 2.10:Upsampling by x2 and low pass filtering

The output results of the direct implementation are shown in Table 2.3.

n	x(n)	m	w(m)	y(m)
n=0	x(0)	m=0	w(0)=x(0)	y(0)=h(0)x(0)
		m=1	w(1)=0	y(1)=h(1)x(0)
n=1	x(1)	m=2	w(2)=x(1)	y(2)=h(0)x(1) + h(2)x(0)
		m=3	w(3)=0	y(3)=h(1)x(1) + h(3)x(0)
n=2	x(2)	m=4	w(4)=x(2)	y(4)=h(0)x(2) + h(2)x(1)
		m=5	w(5)=0	y(5)=h(1)x(2) + h(3)x(1)

Table 2.3: interpolation process

Equation (10) can be rewritten as:

$$H(z)=h(0)+h(2)z^{-2}+z^{-1}(h(1)+h(3)z^{-2}) \tag{12}$$

and by setting

$$E_0(z)=h(0)+h(2)z^{-1} \tag{13}$$

$$E_1(z)=h(1)+h(3)z^{-1} \tag{14}$$

we have:

$$H(z)=E_0(z^2)+E_1(z^2)z^{-1} \tag{15}$$

The outputs before upsampling are:

$$w_0(n)=h(0)x(n)+h(2)x(n-1) \tag{16}$$

$$w_1(n)=h(1)x(n)+h(3)x(n-1) \tag{17}$$

This results in the topology of Figure 2.11.

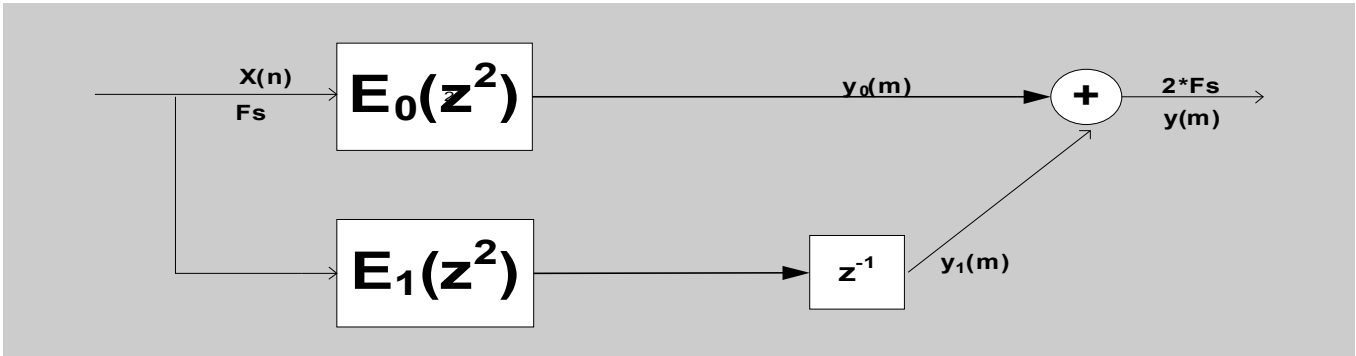

Figure 2.11:Polyphase interpolation

By using the Noble identity shown in Figure 2.12 the topology of Figure 2.11 is reduced to the topology of Figure 2.13.

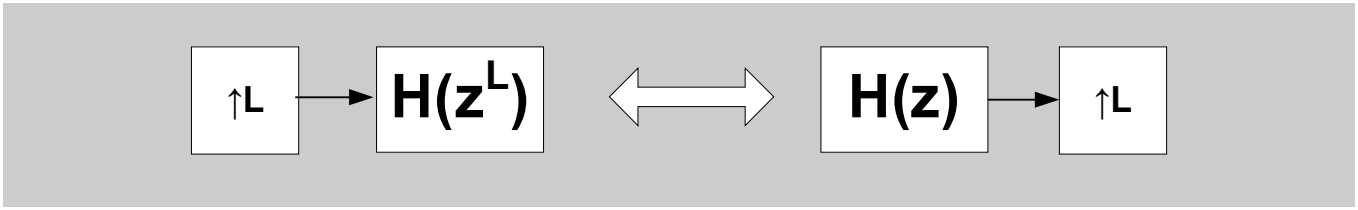

Figure 2.12:Noble identity

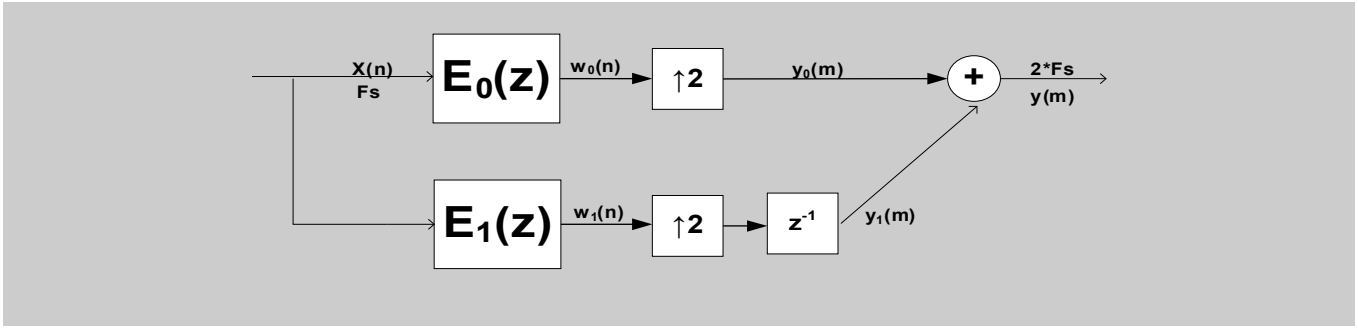

Figure 2.13:Polyphase interpolation

The equations describing the output of the topology of Figure 2.11 after the interpolation are given by Table 2.4:

n	**0**		**1**	
m	**0**	**1**	**2**	**3**
$y_0(m)$	$w_0(0)$	0	$w_0(1)$	0
$y_1(m)$	0	$w_1(0)$	0	$w_1(1)$
$y(m)=y_0(m)+y_1(m)$	$w_0(0)$	$w_1(0)$	$w_0(1)$	$w_1(1)$

Table 2.4: Polyphase implementation

The output y(m) is then given by the sum of the output of the two branches. The topology of Figure 2.13 can be reduced to the topology of Figure 2.14 where the adder and the delay element have been replaced by a switch working at the fast frequency 2·Fs.

From Table 2.4 we can observe that we can take the desirable output y(m) by using this architecture. As we can see the output sampled by the switch is equivalent to sampling the value directly from $y_0(m)$ or $y_1(m)$.

The benefit of this topology is that only the switch has to operate at the increased frequency whereas the other part operates at Fs.

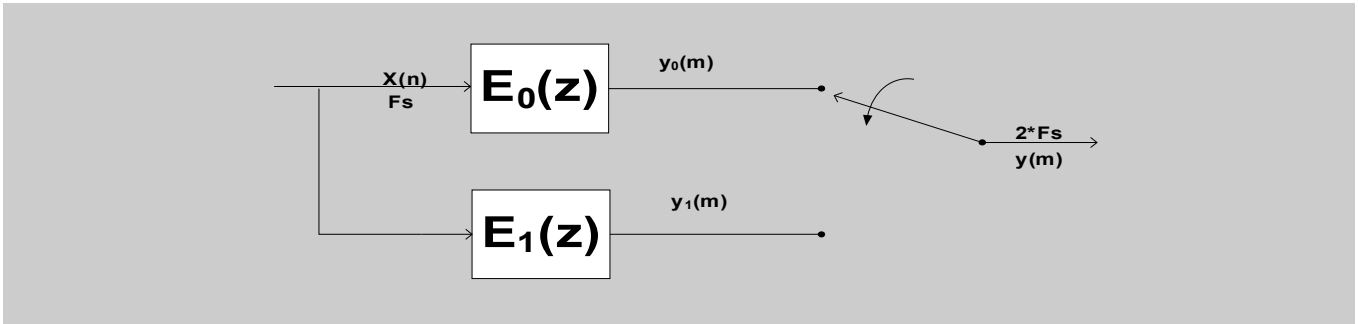

Figure 2.14:Simplified polyphase interpolation filter

2.1.6 Scaling of Filter coefficients

When filtering a signal it is possible to have overflow in the output since multiple multiplications and additions take place. The output must not exceed a given value. In order to calculated the maximum value at the output of the interpolator the method that will be described below can be used.

In terms of presenting the overflow precaution, a simple model of an N tap filter will be used:

It is obvious that the maximum absolute output of the filter occurs when each input at the time $|x(n-i)| = x_{max}$ and the sign of x(n-i) is the same with the sign of the coefficient bi.

Thus, the maximum output value is:

$$y_{MAX}[n] = x_{MAX} \sum_{i=0}^{N-1} |b_i| \tag{18}$$

Assuming that $x_{MAX}=1$ we want to have $y_{MAX}=1$ as well. Therefore, we need to multiply the input signal by a scaling factor sc in order to guarantee that y_{MAX} does not to exceed 1. The equation describing this condition is:

$$|y_{MAX}| = |x_{MAX}| = 1 \geq sc \times x_{MAX} \sum_{i=0}^{N-1} |b_i| \tag{19}$$

Where:

$$sc \leq \frac{1}{\sum_{1=0}^{N-1} |b_i|} \tag{20}$$

The absolute sum of the filter's coefficient is equivalent to the L1 norm of the vector of the coefficients.

2.2 Architecture

This section will present the architecture for the implementation of the interpolation filters taking into consideration the previous analysis.

In the current application, the desired interpolation is 256 in order to achieve SNR greater than 120dB. The selection of the OSR is made according to the maximum SNR performance equation of a third order modulator described in the ΣΔ modulator chapter. To succeed this order of interpolation four stages of upsampling will have to be implemented.

As mentioned in 2.1.3.3 FIR filters are stable and have linear phase response, therefore, FIR filters will be used for the first three stages. Each stage will increase the frequency by x2. A programmable SINC will be used for the last stage to increase the frequency by x32. Figure 2.15 depicts the interpolator's multistage implementation. This topology of filters is preferred for high oversampling ratio D/ADAC converters.

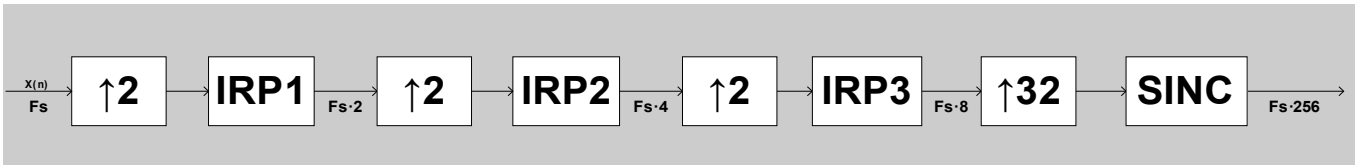

Figure 2.15:Multistage Interpolator by 256

The schematic of the interpolator in its multistage version shown in Figure 2.15 contains several filters. Table 2.5 depicts the filters attributes

Filter	Filter type	Initial sampling rate (KHz)	Interpolation	Fs (kHz)	Fpass (kHz)	Fstop (kHz)	Passband ripple (dB)	Stopband attenuation (dB)	Order
IRP1	FIR Equiripple	6	2	6*2	1.01	3	1	130	44
IRP2	FIR Half Band	6*2	2	6*4	3	9	1	130	30
IRP3	FIR Half Band	6*4	2	6*8	3	21	1	130	18
SINC	SINC	6*8	32	6*8*32	1.01	N/A	1	N/A	N/A

Table 2.5: FIR filter metrics

The filters described above are the ones used in the SIMULINK model for the interpolator. The model of the interpolator along with the filters attributes (magnitude and phase response, impulse response etc.) will be presented in the following figures.

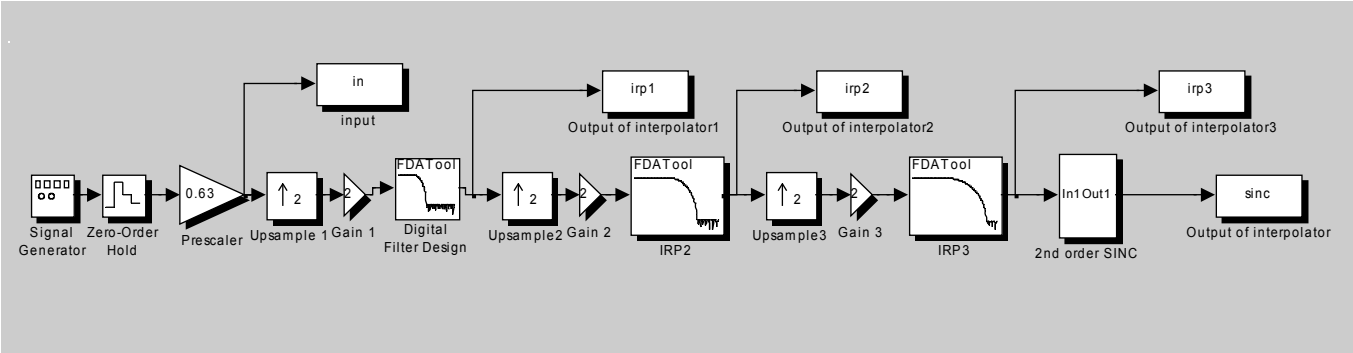

Figure 2.16:SIMULINK model of the interpolator

2.2.0.1 IRP1 Linear Phase Equiripple Low Pass Filter

The equations describing the IRP1 filter are given below. We can see how the polyphase architecture is implemented in a low pass FIR filter of 44th order.

$$H(z)=h(0)+h(44)z^{-44}+h(1)z^{-1}+z^{-43}h(43)=h(0)+h(44)z^{-44}+z^{-1}[h(1)+z^{-42}h(43)] \quad (21)$$

By setting

$$E_0(z)=h(0)+...+h(44)z^{-22} \quad (22)$$

$$E_1(z)=h(1)+...+h(43)z^{-21} \quad (23)$$

We have:

$$H(z)=E_0(z^2)+E_1(z^2)z^{-1} \quad (24)$$

$$y_0(n)=h(0)x(n)+...+h(44)x(n-22) \quad (25)$$

$$y_1(z)=h(1)x(n)+...+h(43)x(n-21) \quad (26)$$

The model specified in the SIMULINK tool for the IRP1 is using double precision floating point arithmetic. Instead. the quantization of the filter coefficients as well as the fixed point modeling will be applied to the model, to reach the filter specifications. Figure 2.17 displays the differences between the real model and a low accuracy fixed point model and Figure 2.18 presents the final fixed model and the real one.

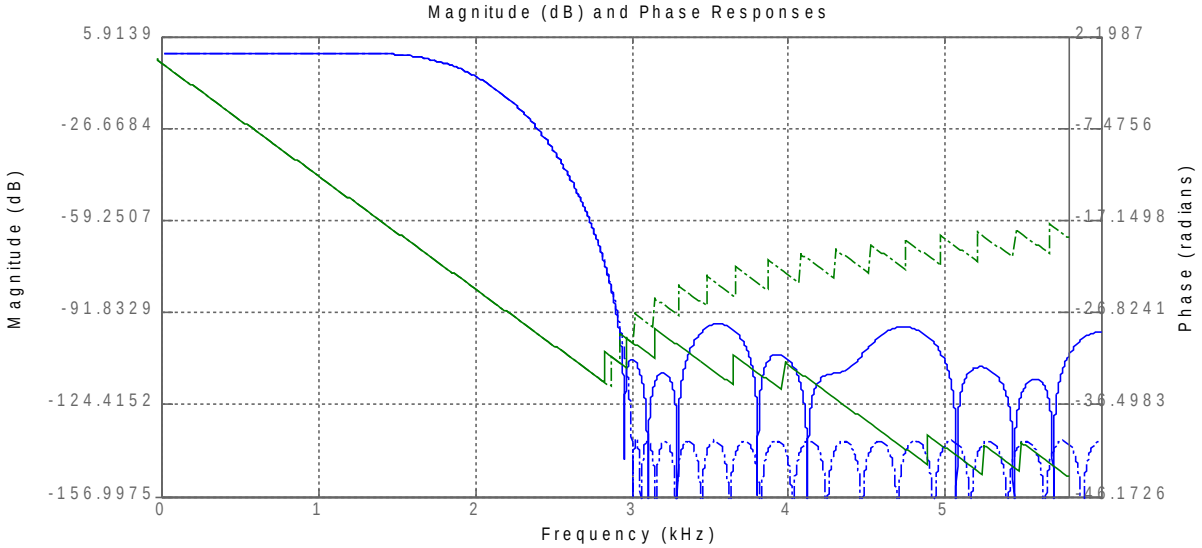

Figure 2.17:Magnitude and phase response of low accuracy fixed point model vs real model of IRP1

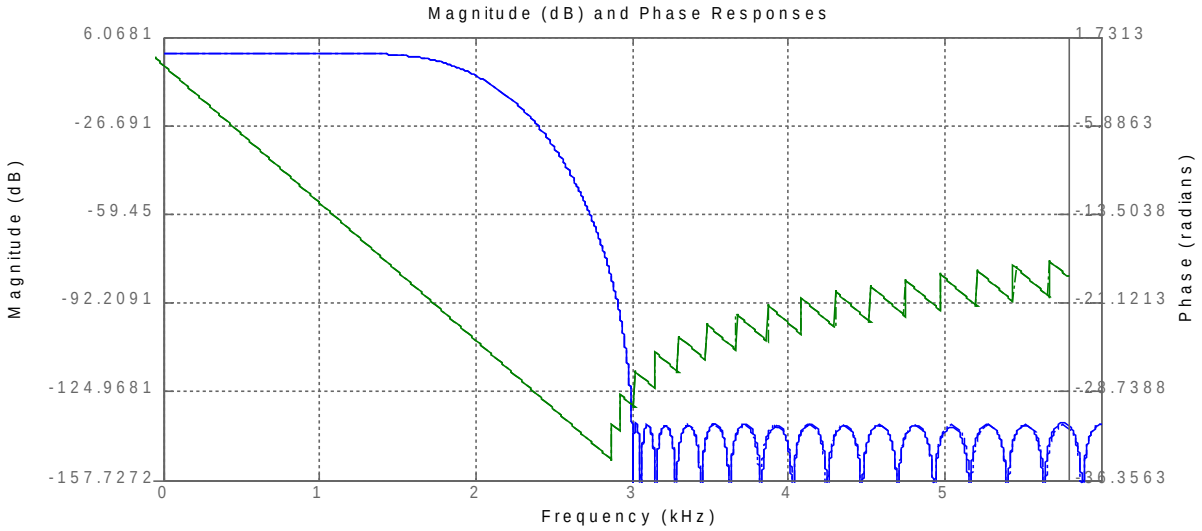

Figure 2.18:Magnitude and phase response of final fixed point model vs real model of IRP1

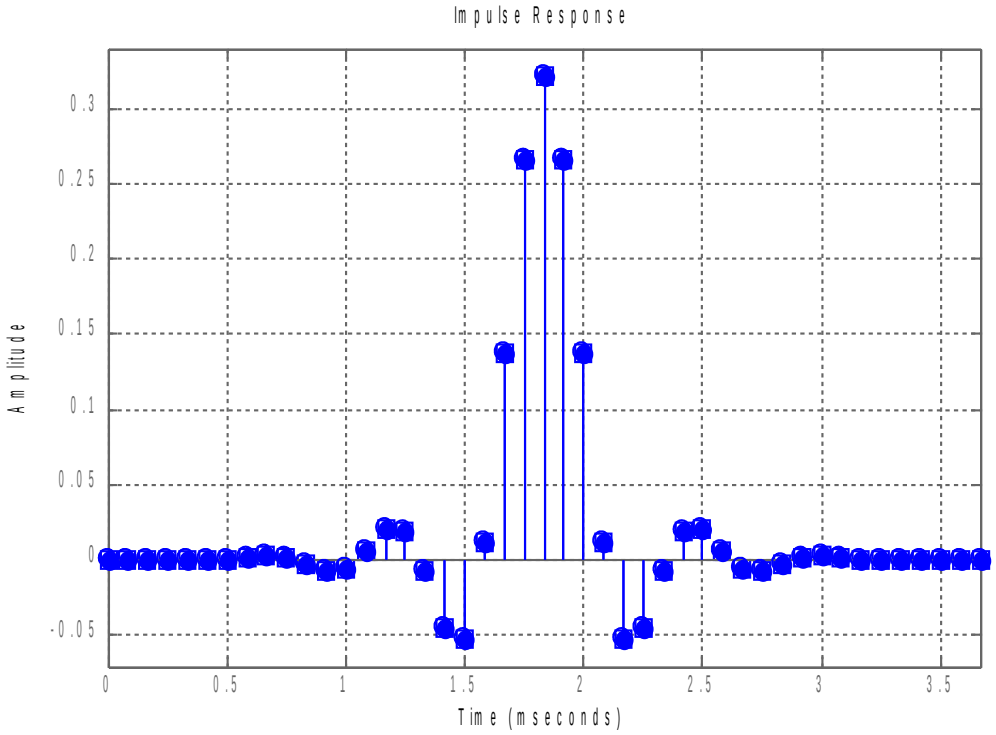

Figure 2.19:Impulse response of low pass equiripple filter

In Figure 2.20 we can also see the details of the fixed point arithmetic used in the model of IRP1.

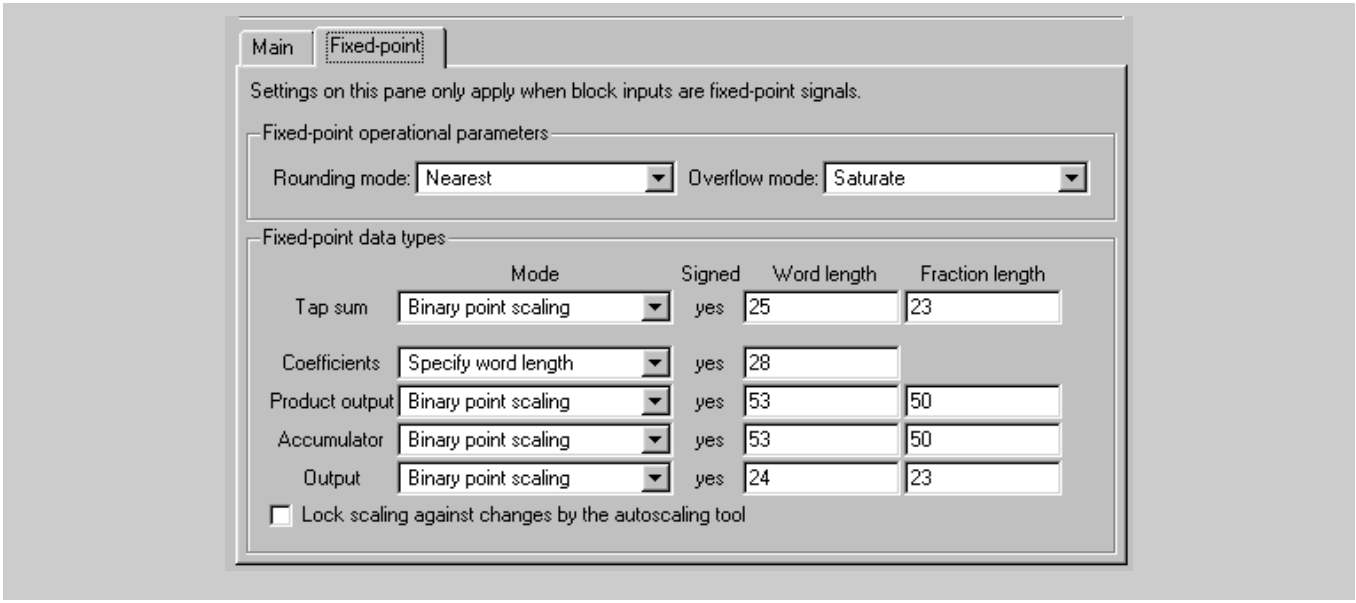

Figure 2.20:Fixed point arithmetic of IRP1

2.2.0.2 IRP2 Half Band Filter

The equations describing the IRP2 filter are given below. We can see how the polyphase architecture is implemented in a low pass FIR filter of 30th order.

$$H(z)=h(0)+h(30)z^{-30}+h(1)z^{-1}+z^{-29}h(29)=h(0)+h(30)z^{-30}+z^{-1}[h(1)+z^{-28}h(29)] \quad (27)$$

By setting

$$E_0(z)=h(0)+...+h(30)z^{-15} \tag{28}$$

$$E_1(z)=h(1)+...+h(29)z^{-14} \tag{29}$$

We have:

$$H(z)=E_0(z^2)+E_1(z^2)z^{-1} \tag{30}$$

$$y_0(n)=h(0)x(n)+...+h(30)x(n-15) \tag{31}$$

$$y_1(n)=h(1)x(n)+...+h(29)x(n-14) \tag{32}$$

because the half band filters have half of their coefficients set to zero, equation (32) becomes:

$$y_1(n)=h(15)x(n-7) \tag{33}$$

As already described in the IRP1 filter, the model presented is using the double precision floating point arithmetic to generate the filter. In the RTL model the design applied will have to follow a fixed point arithmetic to reduce the complexity. In Figure 2.21 there are displayed the performance differences of a low accuracy model and the real one and in Figure 2.22 the differences of the applied fixed point model and the real one.

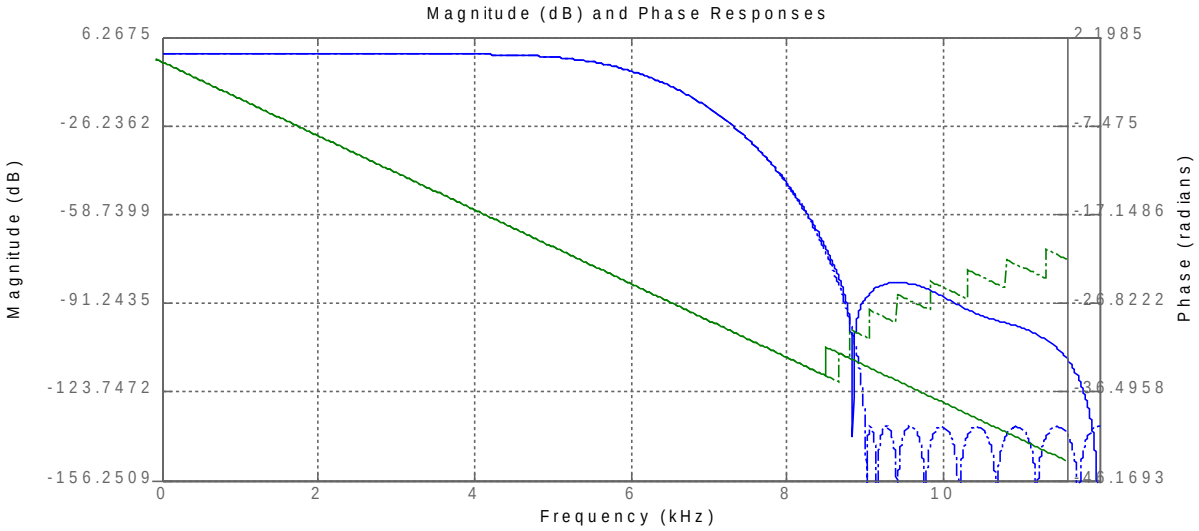

Figure 2.21:Magnitude and phase response of low accuracy fixed point model vs real model of IRP2

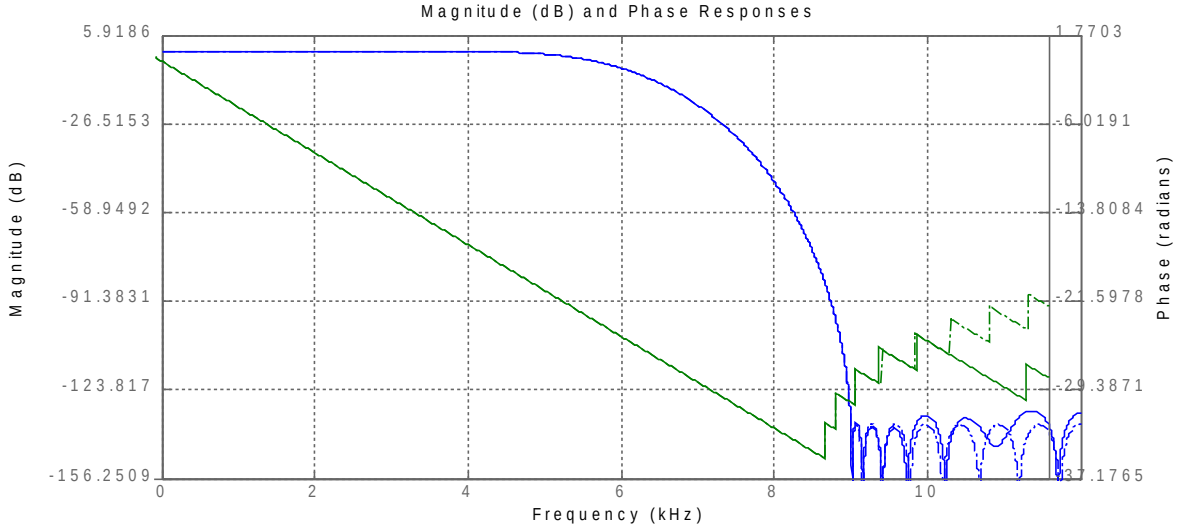

Figure 2.22:Magnitude and phase response of final fixed point model vs real model of IRP2

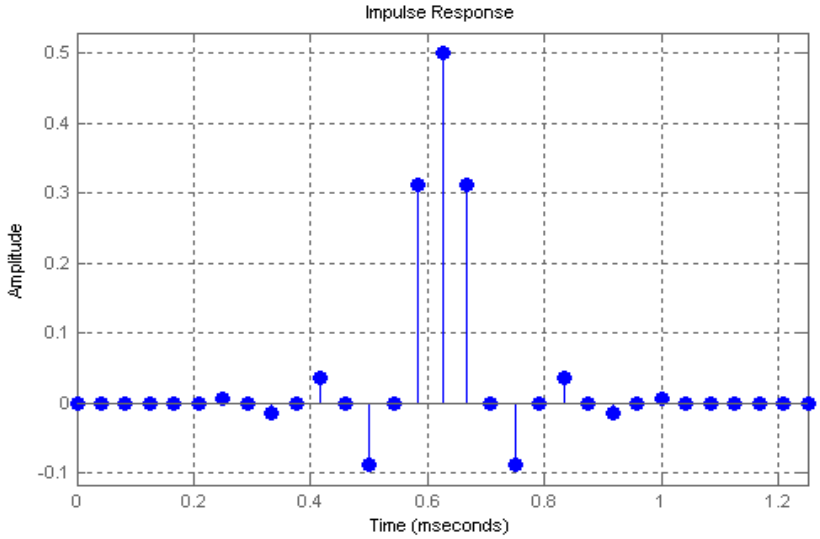

Figure 2.23:Impulse response of first half band filter (IRP2)

Below in Figure 2.24 there are depicted the arithmetic details of the fixed point model concerning the filter coefficients as well as the precision of the operations inside the filter.

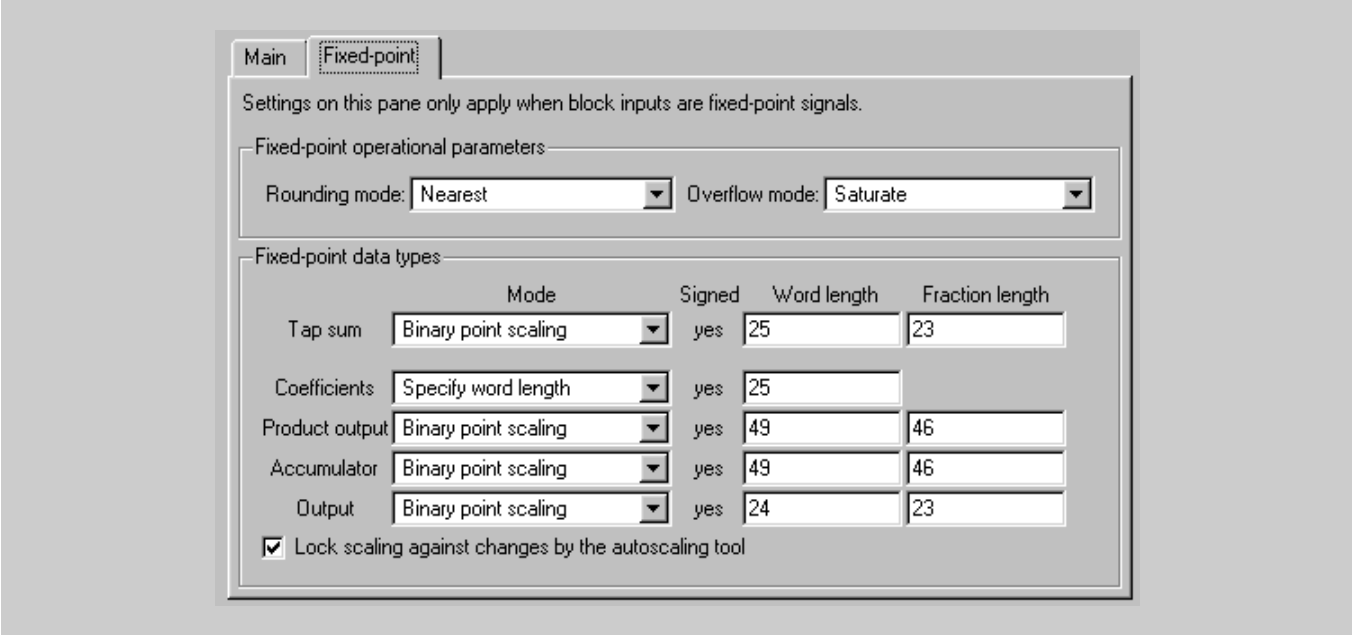

Figure 2.24:Fixed point arithmetic of IRP2

2.2.0.3 IRP3 Half Band Filter

The equations describing the IRP2 filter are given below. We can see how the polyphase architecture is implemented in a low pass FIR filter of 30th order.

$$H(z)=h(0)+h(18)z^{-18}+h(1)z^{-1}+z^{-17}h(17)=h(0)+h(18)z^{-18}+z^{-1}[h(1)+z^{-16}h(17)] \quad (34)$$

By setting

$$E_0(z)=h(0)+...+h(18)z^{-9} \quad (35)$$

$$E_1(z)=h(1)+...+h(17)z^{-8} \quad (36)$$

We have:

$$H(z)=E_0(z^2)+E_1(z^2)z^{-1} \quad (37)$$

$$y_0(n)=h(0)x(n)+...+h(18)x(n-9) \quad (38)$$

$$y_1(n)=h(1)x(n)+...+h(17)x(n-8) \quad (39)$$

because the half band filters have half of their coefficients set to zero, equation (39) becomes:

$$y_1(n)=h(9)x(n-4) \quad (40)$$

When designing the real model, double precision floating point arithmetic is used for the IRP3 filter. In order to follow the design architecture of the previous filters, fixed point arithmetic will have to be applied as in IRP3 filter. Figure 2.25 displays the differences between a low accuracy fixed point model and the real one and Figure 2.26 displays the differences between the applied fixed point model and the real one.

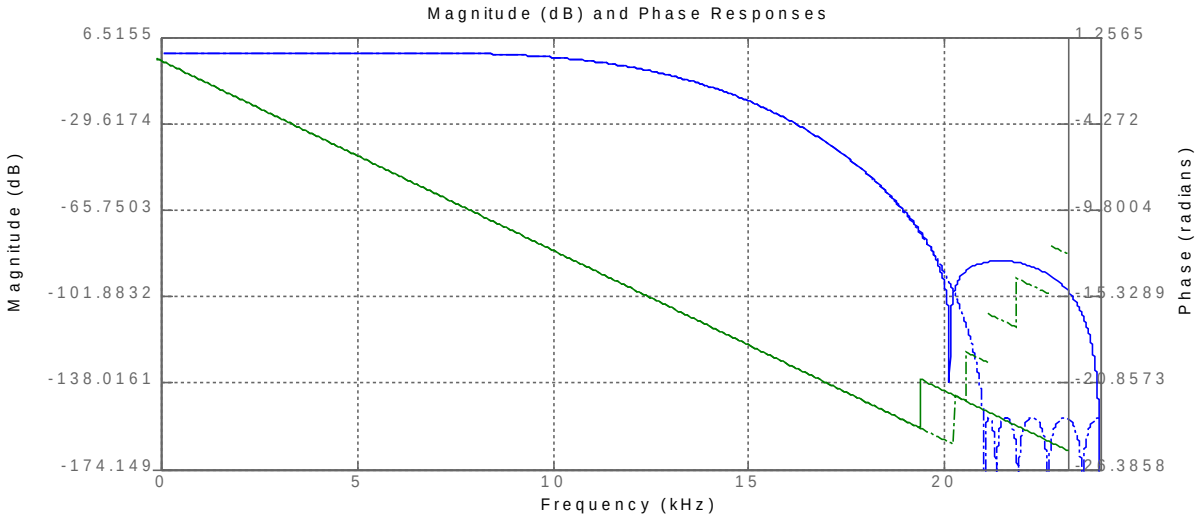

Figure 2.25:Magnitude and phase response of the low accuracy fixed point model vs the real model of IRP3

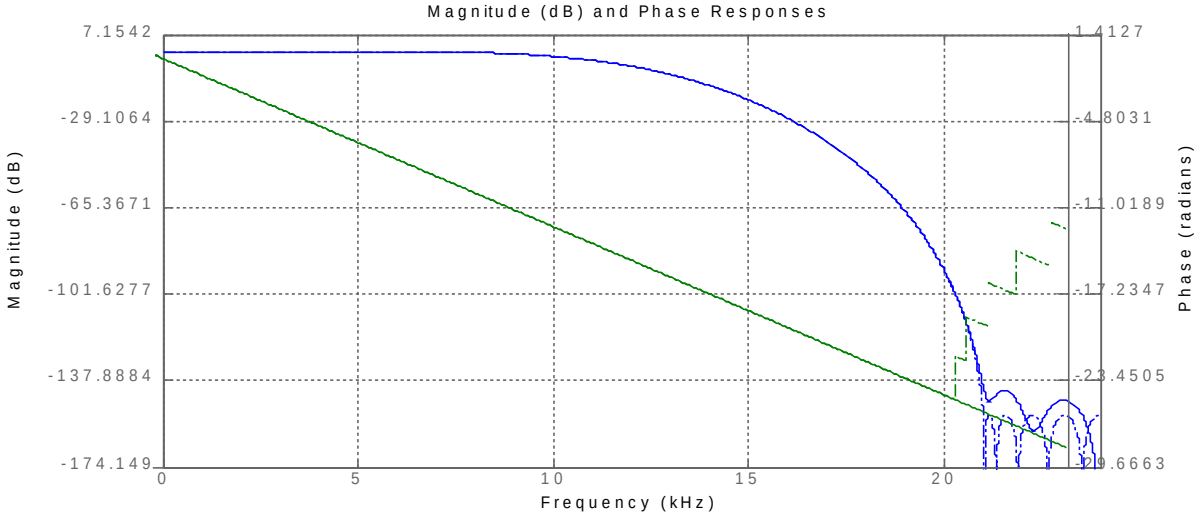

Figure 2.26:Magnitude and phase response of applied fixed point model vs the real model of IRP3

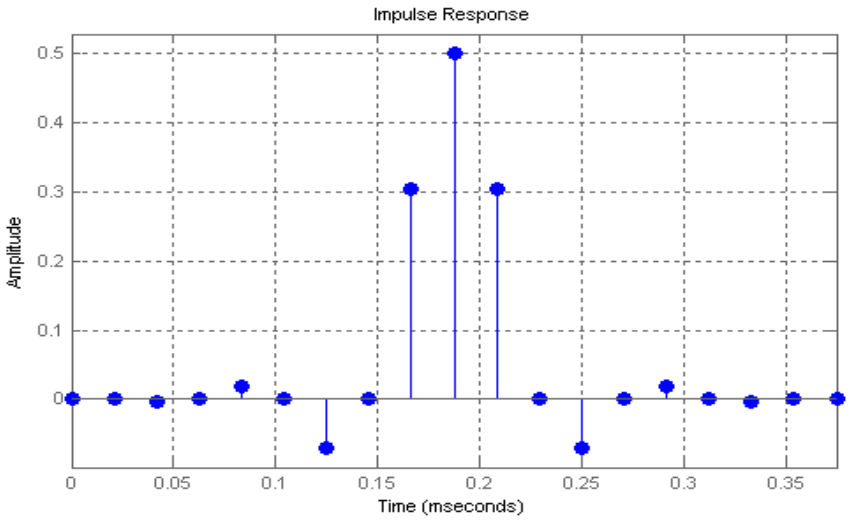

Figure 2.27:Impulse response of second half band filter (IRP3)

Below in Figure 2.28 the details of the fixed point arithmetic applied to the model is presented.

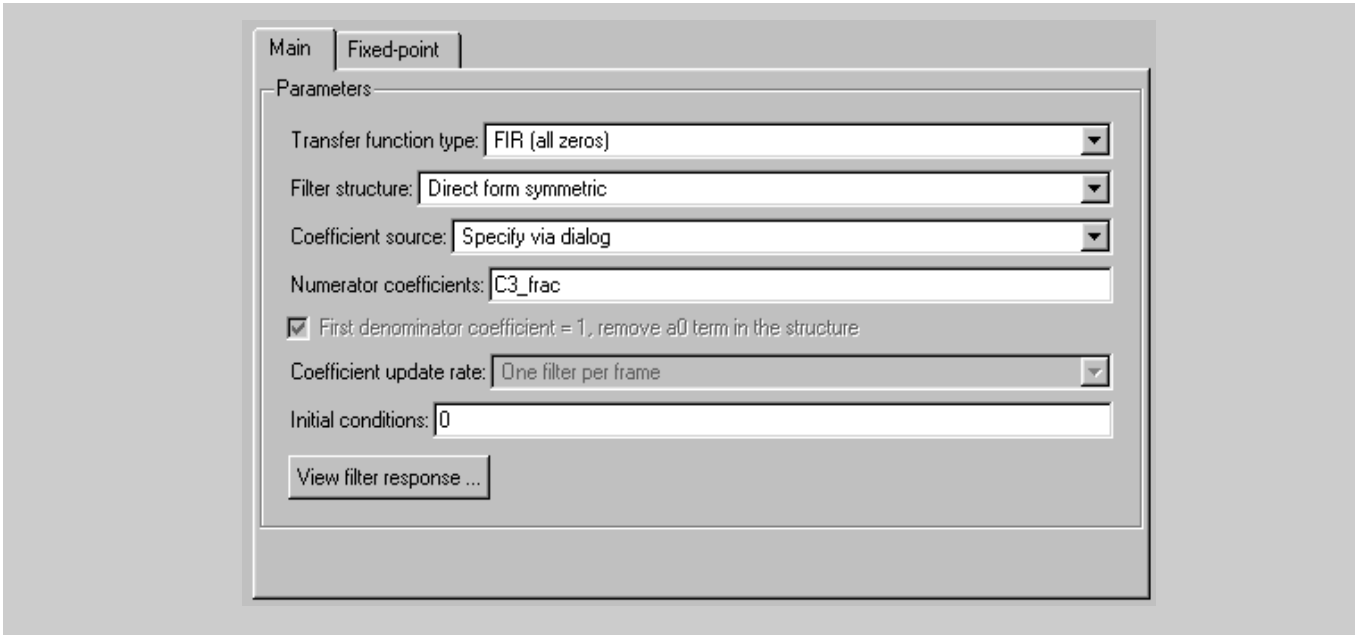

Figure 2.28:Fixed point arithmetic details of IRP3

As we can see from the figures depicting the frequency and the phase response, the phase is linear in the pass-band.

The previous figures depict the phase, frequency and magnitude responses of the filters that will be used by the interpolator. The 2nd order SINC filter that will be used is given by the model in Figure 2.5 which uses fixed point arithmetic.

2.3 MATLAB simulation results

For each filter described on the previous section, there will be presented the input-output results of the SIMULINK model in the time and frequency domain.

IRP1 results

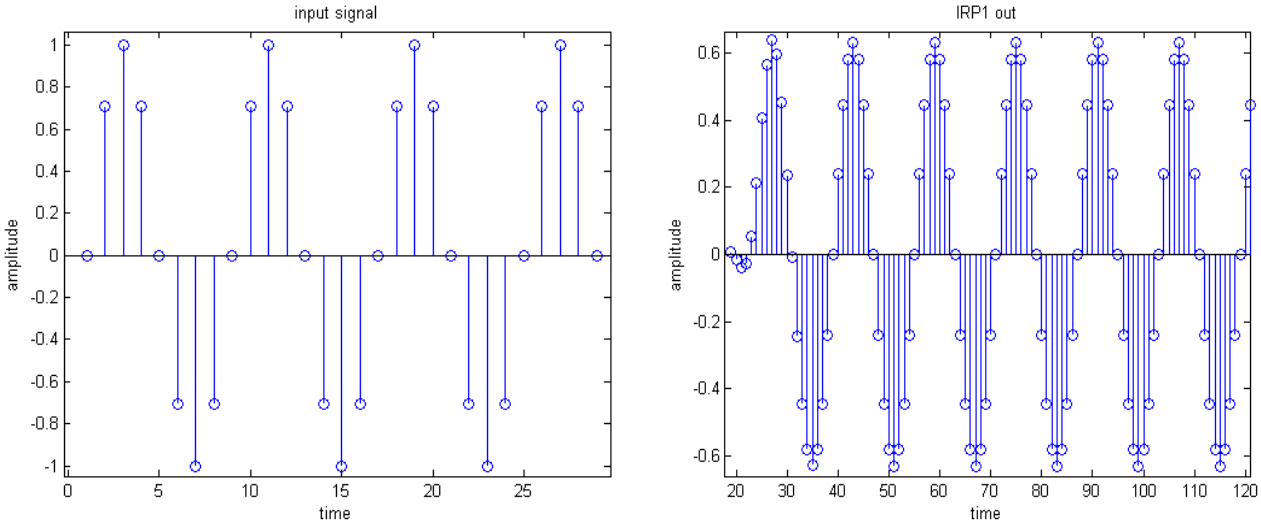

Figure 2.29: Time response of input signal and output of first filter (IRP1)

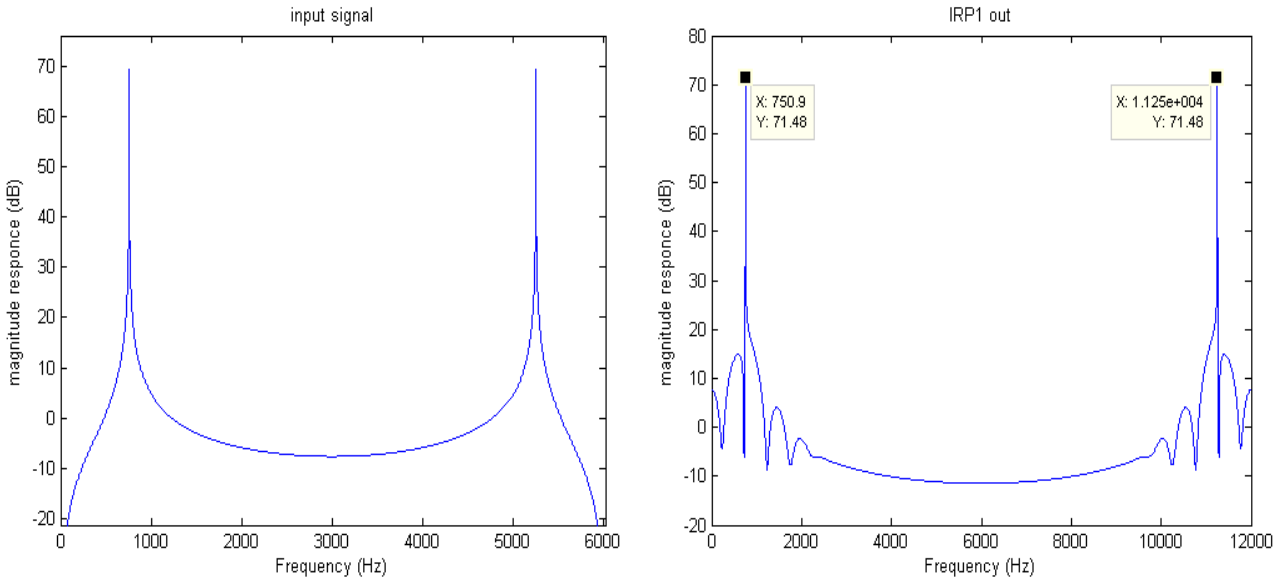

Figure 2.30: Frequency response of input signal and output of first filter (IRP1)

IRP2 results

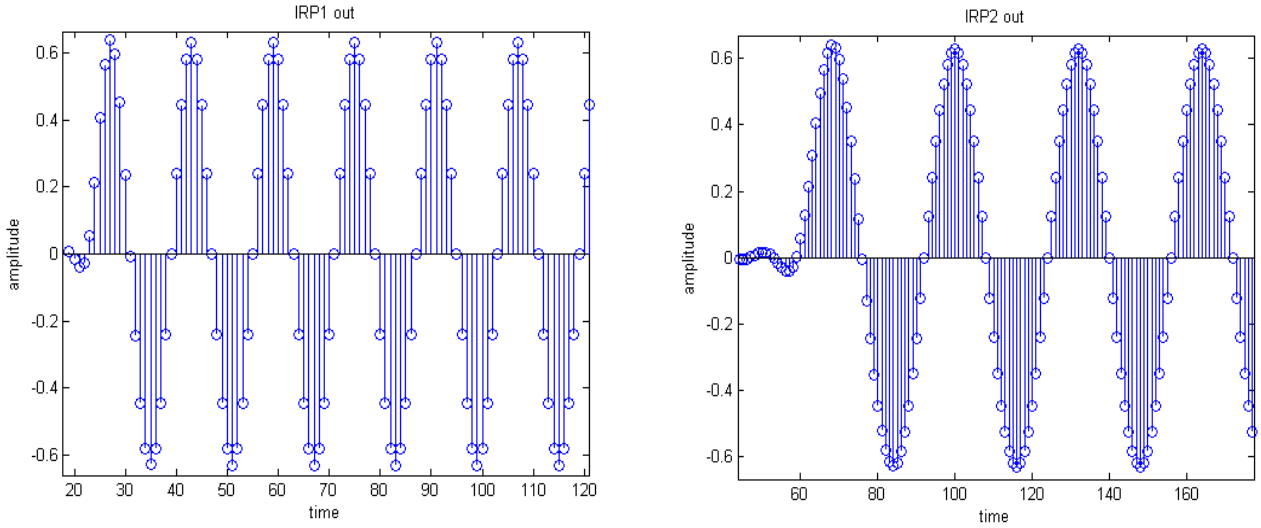

Figure 2.31: Time response of IRP1 and output of second filter (IRP2)

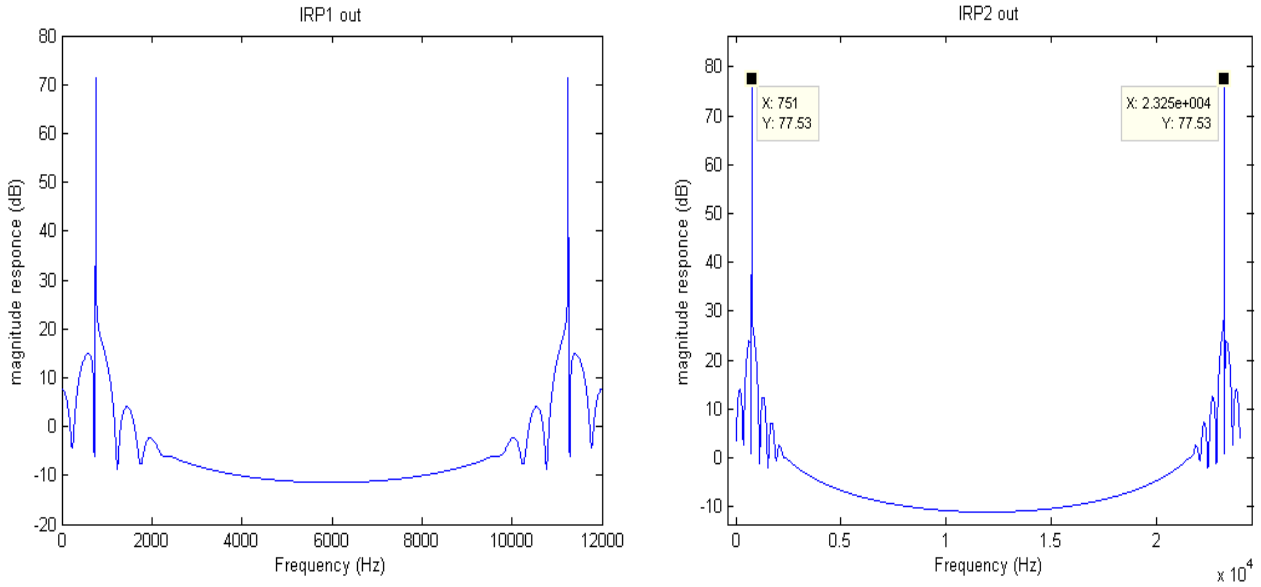

Figure 2.32: Frequency response of IRP1 and output of second filter (IRP2)

IRP3 results

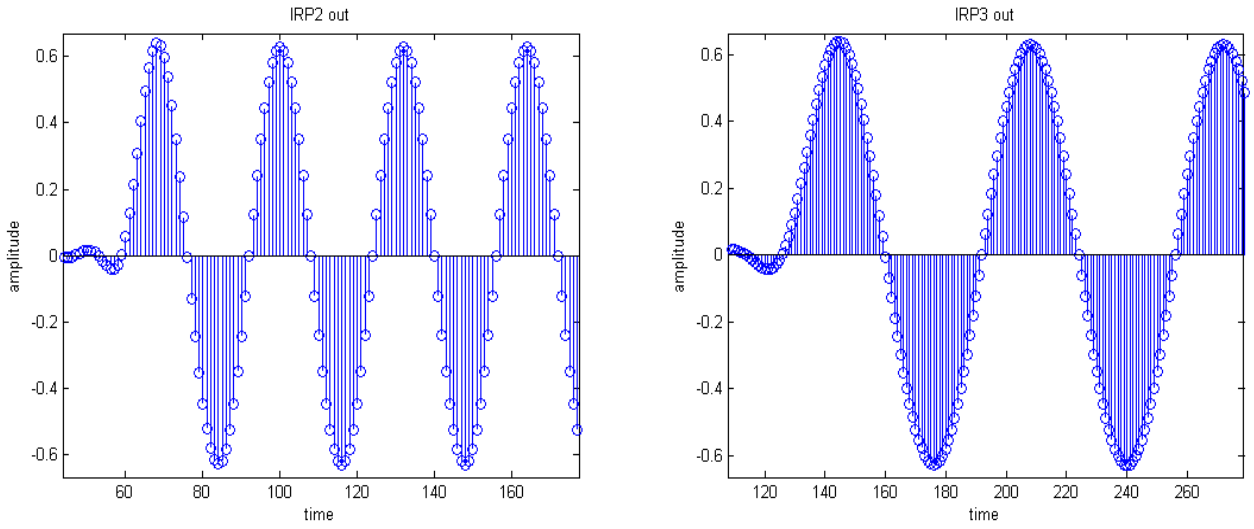

Figure 2.33: Time response of IRP2 and output of third filter (IRP3)

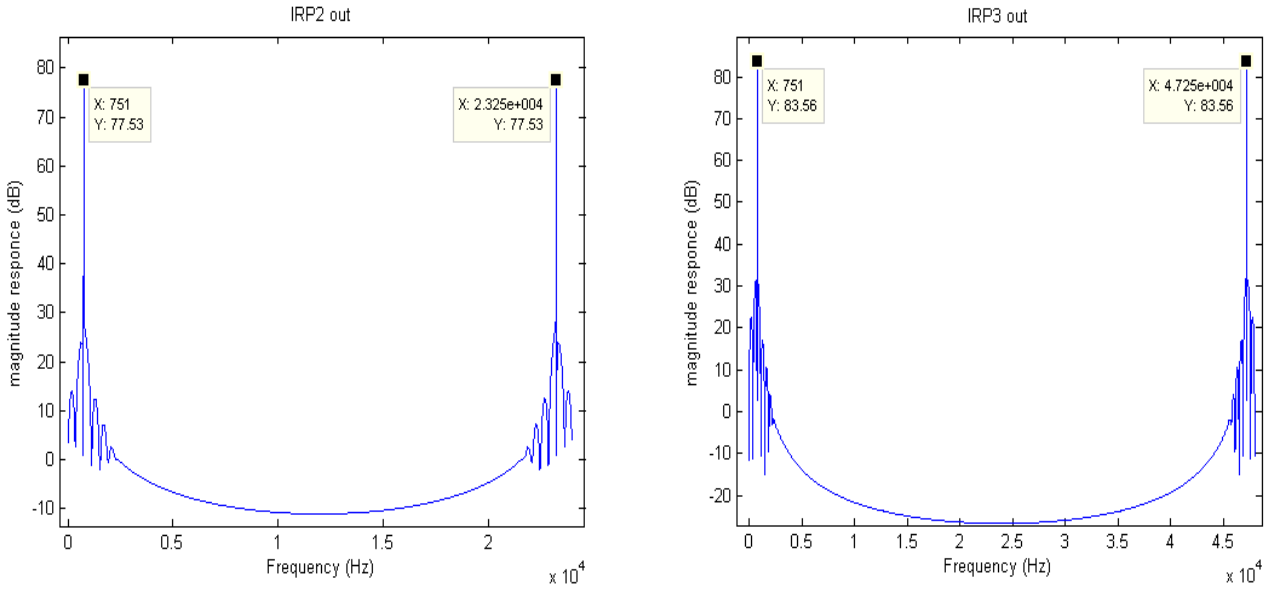

Figure 2.34: Frequency response of IRP2 and output of third filter (IRP3)

SINC results

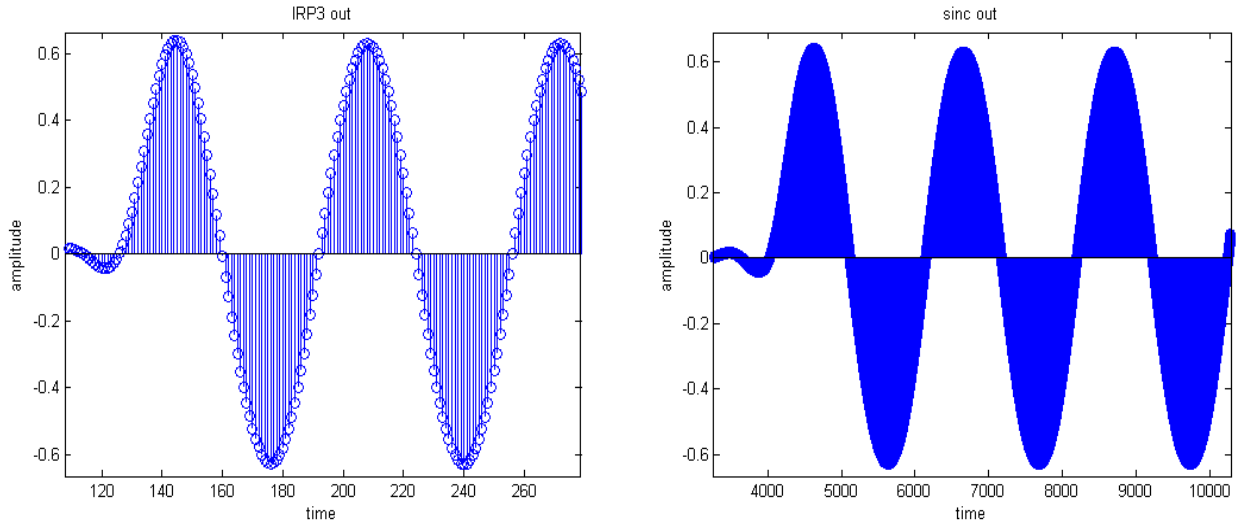

Figure 2.35: Time response of IRP3 and output of SINC filter

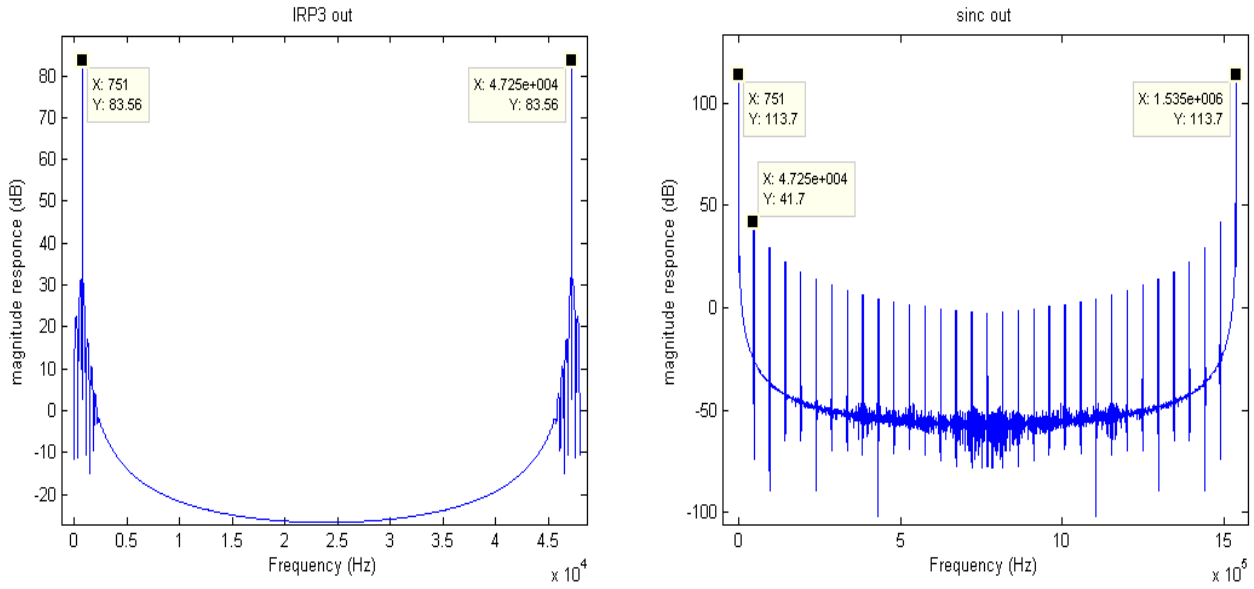

Figure 2.36: Frequency response of IRP3 and output ofSINC filter

The final output of the interpolator equals to the output of the SINC filter. These simulations are performed using the model of Figure 2.16.

2.4 H/W Implementation

This chapter will present the implementation details of the interpolator. RTL architecture diagrams, I/O interfaces and detailed descriptions of the filters will be shown.

2.4.1 Procedural diagram

Figure 2.37 shows the procedural diagram of the interpolator. The interpolator is implemented using the multistage approach to achieve reduced complexity. The interpolator consists of a low pass filter (IRP1), two half band filters (IRP2 and IRP3) and a sinc filter (SINC) as already mentioned in the architecture paragraph.

The filters will be implemented by using the polyphase architecture except from the last filter (SINC).This architecture exploits the symmetry of the coefficients and leads to a low complexity implementation.

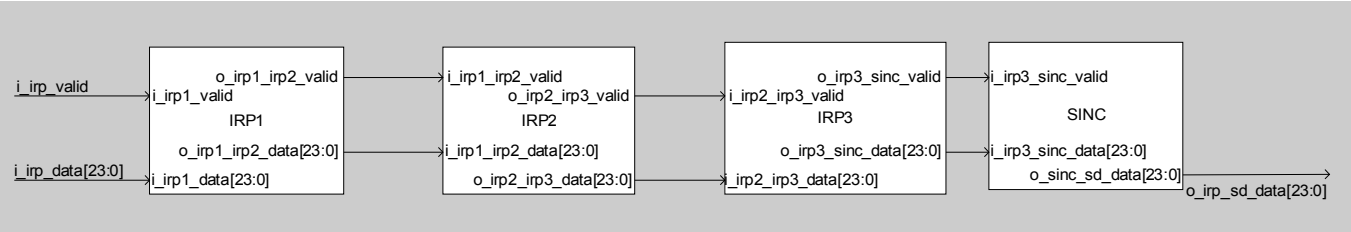

Figure 2.37: Interpolator Procedural Diagram

In Figure 2.38 the I/O interface block is presented showing the interconnection signals.

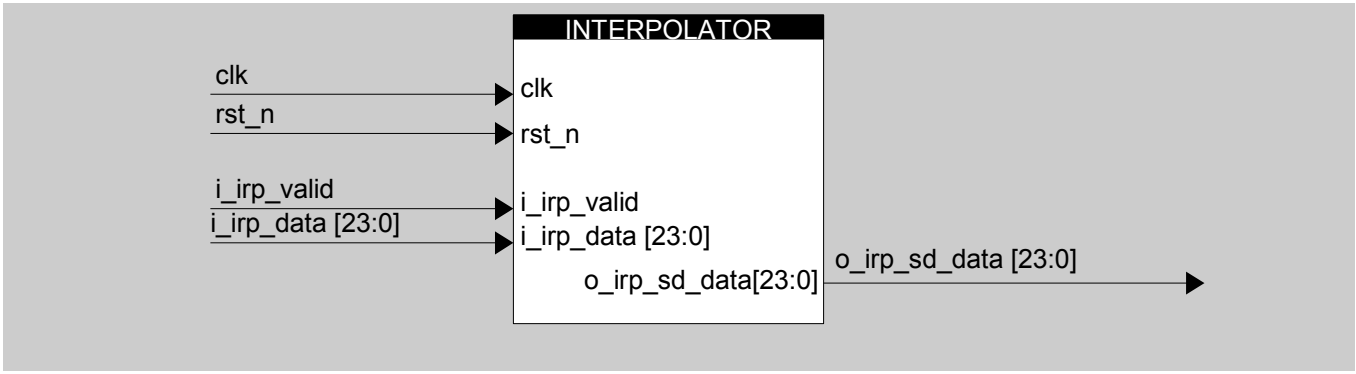

Figure 2.38: INTERPOLATOR I/O Block Diagram

Table 2.6 shows the pin list of the interpolator with all the I/O signals involved.

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock
rst_n	in	Active low asynchronous reset
CONTROL		
i_irp_valid	in	Indicates that a new 24-bit input data sampled at Fs
Input interface		
i_irp_data (23:0)	in	24-bit input data at Fs
ΣΔ modulator interface		
o_irp_sd_data (23:0)	out	24-bit output data at 256Fs

Table 2.6: Interpolator Pin List

In the following paragraphs the design details of the filters contained in the interpolator will be presented.

2.4.2 Linear phase equiripple filter (IRP1)

The pin list along with the interface diagram of IRP1 will be shown in Table 2.7 and Figure 2.39 respectively.

Signal	I/O	Description
Clocks and reset		
clk	in	Master clock at 256Fs
rst_n	in	Active low asynchronous reset
CONTROL		
i_irp1_valid	in	Indicates that a new 24-bit input data is sampled at the input at Fs rate
Input interface		
i_int_data [23:0]	in	24-bit input data
IRP2 filter interface		
o_irp1_irp2_data[23:0]	out	24-bit output interpolated data at 2Fs
o_irp1_irp2_valid	out	Indicates that a 24-bit output data is ready at the output at 2*Fs rate

Table 2.7: IRP1 Pin List

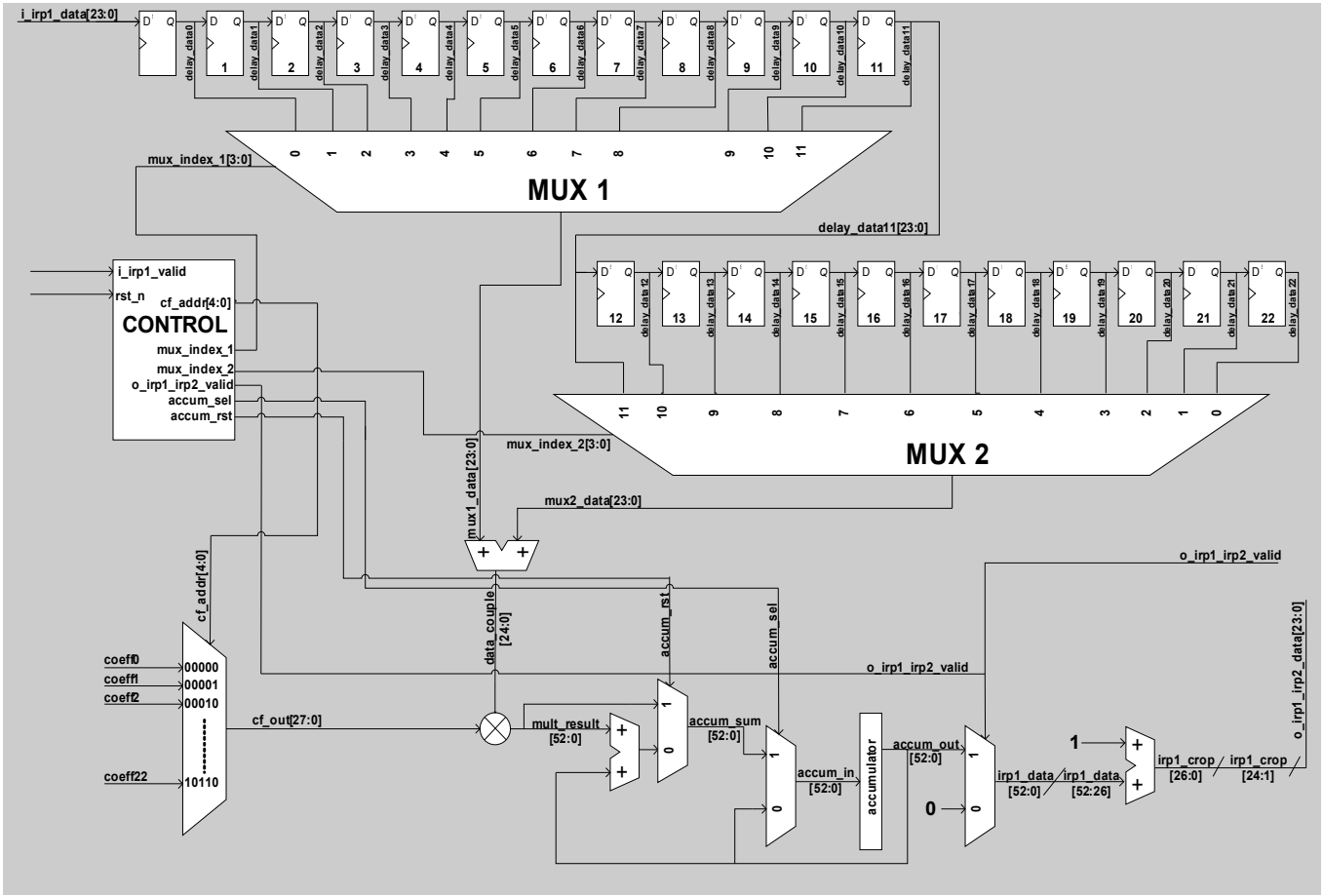

Figure 2.39: IRP1 Interface Diagram

Description

The first block of the interpolator is the IRP1 filter that interpolates the input signal *i_irp1_data* by a factor of x2.

The coefficients of the filter are chosen by a multiplexer 22-to-1. The coefficients of each filter are given in Appendix C. The values of the coefficients are calculated assuming that there are 24 bits for the input signal, 27 bits for the representation of the coefficients. Moreover, the prescaling factor of 0.63 is applied to the coefficients so the output of the overall interpolation stage remains bounded.

There are two branches that will perform the polyphase interpolation. The one that corresponds to the differential equation:

$$y_0(n)=h(0)\,x(n)+\ldots+h(44)\,x(n-22) \tag{41}$$

and the other one that corresponds to the differential equation:

$$y_1(n)=h(1)\,x(n)+\ldots+h(43)\,x(n-21) \tag{42}$$

Since the filters are symmetric it is convenient to add the input data and then multiply the result of the addition with the appropriate coefficient. We use the Multiply-Accumulator architecture in order to make the additions.

For each couple of the data that correspond to the symmetric coefficients the proper index *mux_index1* and *mux_index2* fetches the values from the delay line and the signal *cf_addr* fetches the proper coefficient from the multiplexer. Then the two input signals $y_0(n)$ and $y_1(n)$ are summed.

After the summation, the coefficient corresponding to that couple is chosen from the coefficient multiplexer and multiplied with it. The result of each multiplication is placed in the signal *mux_result,* which is 53-bits long according to the multiplication rules. The signal *mux_result* is then added to the current value of the accumulator output, the *accum_out* signal.

The IRP1 block as mentioned before, upsamples the input signal by x2, while the master clock is at 256Fs. Therefore the input data of the interpolator arrives every 256 clock cycles. Consequently, the IRP1 block shall output data every 128 clock cycles.

When all the computations from the first branch of IRP1 filter are finished, the output data wait for 128 clock cycles and then they are transmitted to the next filter. While the output data wait in the accumulator, the *accum_in* signal is connected with the *accum_out* signal through a multiplexer.

When the signal irp*1_irp2_valid* is set to '1' the accumulator's data are driven to the output after their quantization. The quantization takes place because the accumulators data are 53-bit and the output data of the IRP1 has to be 24-bit. To achieve the right quantization we take the 27 MSB's of the *accum_out* and we sum '1' placing them to the *irp1_crop* signal. The *o_irp1_irp2_data* is the 25 MSBs of the *irp1_crop* signal.

The procedure described above occurs two times for each input data. After the second filter drives its output to the IRP2 filter, the accumulator's input becomes the multipliers output without the addition of the accumulators output, in order for the previous data to be flushed.

The IRP1 block is using an internal control that assigns the proper values to the signals *mux_index_1* , *mux_index_2* , *cf_addr, o_irp1_irp2_valid, accum_sel and accum_rst* which control the multiplexers as well as the coefficients address of the RF block. The control block is implemented with an FSM.

Table 2.8 shows the values of the signals at each state of the FSM.

state	mux_index1	mux_index2	cf_addr	o_irp1_irp2_valid	accum_sel	accum_rst	nx_state
state0	0	0	-1	'0'	'1'	'0'	state0 if i_valid='0' else state1
state1	0	0	0	'0'	'1'	'1'	state2
state2	1	1	2	'0'	'1'	'0'	state3
state3	2	2	4	'0'	'1'	'0'	state4
state4	3	3	6	'0'	'1'	'0'	state5
state5	4	4	8	'0'	'1'	'0'	state6
state6	5	5	10	'0'	'1'	'0'	state7
state7	6	6	12	'0'	'1'	'0'	state8
state8	7	7	14	'0'	'1'	'0'	state9
state9	8	8	16	'0'	'1'	'0'	state10
state10	9	9	18	'0'	'1'	'0'	state11
state11	10	10	20	'0'	'1'	'0'	state12
state12	11	11	22	'0'	'1'	'0'	state13
state13	0	0	-1	'1' if counter=127 else '0'	'0'	'0'	state14 if counter=127 else state13
state14	0	1	1	'0'	'1'	'1'	state15
state15	1	2	3	'0'	'1'	'0'	state16
state16	2	3	5	'0'	'1'	'0'	state17
state17	3	4	7	'0'	'1'	'0'	state18
state18	4	5	9	'0'	'1'	'0'	state19
state19	5	6	11	'0'	'1'	'0'	state20
state20	6	7	13	'0'	'1'	'0'	state21
state21	7	8	15	'0'	'1'	'0'	state22
state22	8	9	17	'0'	'1'	'0'	state23
state23	9	10	19	'0'	'1'	'0'	state24
state24	10	11	21	'0'	'1'	'0'	state25
state25	0	0	-1	'1' if counter=256 else '0'	'0'	'0'	state0 if counter=256 else state25

Table 2.8: IRP1 FSM state description table

The FSM has 25 states. state0 is the idle state at which the IRP1 is reset when *rst_n* signal is set to '0' or when IRP1 waits for the *i_irp1_valid* signal to rise and fetch the next input data. When the *i_irp1_valid* signal rises, the state jumps to state1 and the IRP1's calculations begin. When the first branch of the filter finishes its calculations at state12, the FSM enters state13 waiting for the counter to count a total of 127 cycles from the rise of the i_valid signal and then drives its output to the IRP2 filter. After that, the FSM flushes the accumulator with the *accum_rst* signal as described previously and enters state14 where the other branch of the filter begins its calculations. When the other half of the filter finishes, the FSM enters state25 and waits for the counter to receive the value 256 so that IRP1 can drive the output of the second half of the filter. After that the FSM jumps to state0 waiting for the next input to arrive.

Table 2.9 shows the coefficient and input data relation.

mux_index1	mux_index2	inputs	cf_addr	i_irp1_valid	o_irp1_irp2_valid
0	0	x(n)+x(n-22)	h0	**1**	0
1	1	x(n-1)+x(n-21)	h2	0	0
2	2	x(n-2)+x(n-20)	h4	0	0
3	3	x(n-3)+x(n-19)	h6	0	0
4	4	x(n-4)+x(n-18)	h8	0	0
5	5	x(n-5)+x(n-17)	h10	0	0
6	6	x(n-6)+x(n-16)	h12	0	0
7	7	x(n-7)+x(n-15)	h14	0	0
8	8	x(n-8)+x(n-14)	h16	0	0
9	9	x(n-9)x(n-13)	h18	0	0
10	10	x(n-10)+x(n-12)	h20	0	0
11	11	x(n-11)+x(n-11)	h22/2	0	1
0	1	x(n)+x(n-21)	h1	0	0
1	2	x(n-1)+x(n-20)	h3	0	0
2	3	x(n-2)+x(n-19)	h5	0	0
3	4	x(n-3)+x(n-18)	h7	0	0
4	5	x(n-4)+x(n-17)	h9	0	0
5	6	x(n-5)+x(n-16)	h11	0	0
6	7	x(n-6)+x(n-15)	h13	0	0
7	8	x(n-7)+x(n-14)	h15	0	0
8	9	x(n-8)+x(n-13)	h17	0	0
9	10	x(n-9)x(n-12)	h19	0	0
10	11	x(n-10)+x(n-11)	h21	0	1
0	0	x(n)+x(n-22)	h0	**1**	0
1	1	x(n-1)+x(n-21)	h2	0	0
2	2	x(n-2)+x(n-20)	h4	0	0

Table 2.9: IRP1 signal and time table

2.4.3 Half Band filter (IRP2)

The pin list along with the interface diagram of IRP1 will be shown in Table 2.10 and Figure 2.40 respectively.

Signal	I/O	Description
Clocks and reset		
clk	in	Master clock at 256Fs
rst_n	in	Active low asynchronous reset
CONTROL		
i_irp1_irp2_valid	in	Indicates that a new 24-bit input data is sampled at the input at 2*Fs rate
Input interface		
i_irp1_irp2_data [23:0]	in	24-bit input data from IRP1 at 2Fs
IRP3 interface		
o_irp2_irp3_data[23:0]	out	24-bit output data
o_irp2_irp3_valid	out	Indicates that a 24-bit output data is ready at the output at 2*2*Fs rate

Table 2.10: IRP2 Pin List

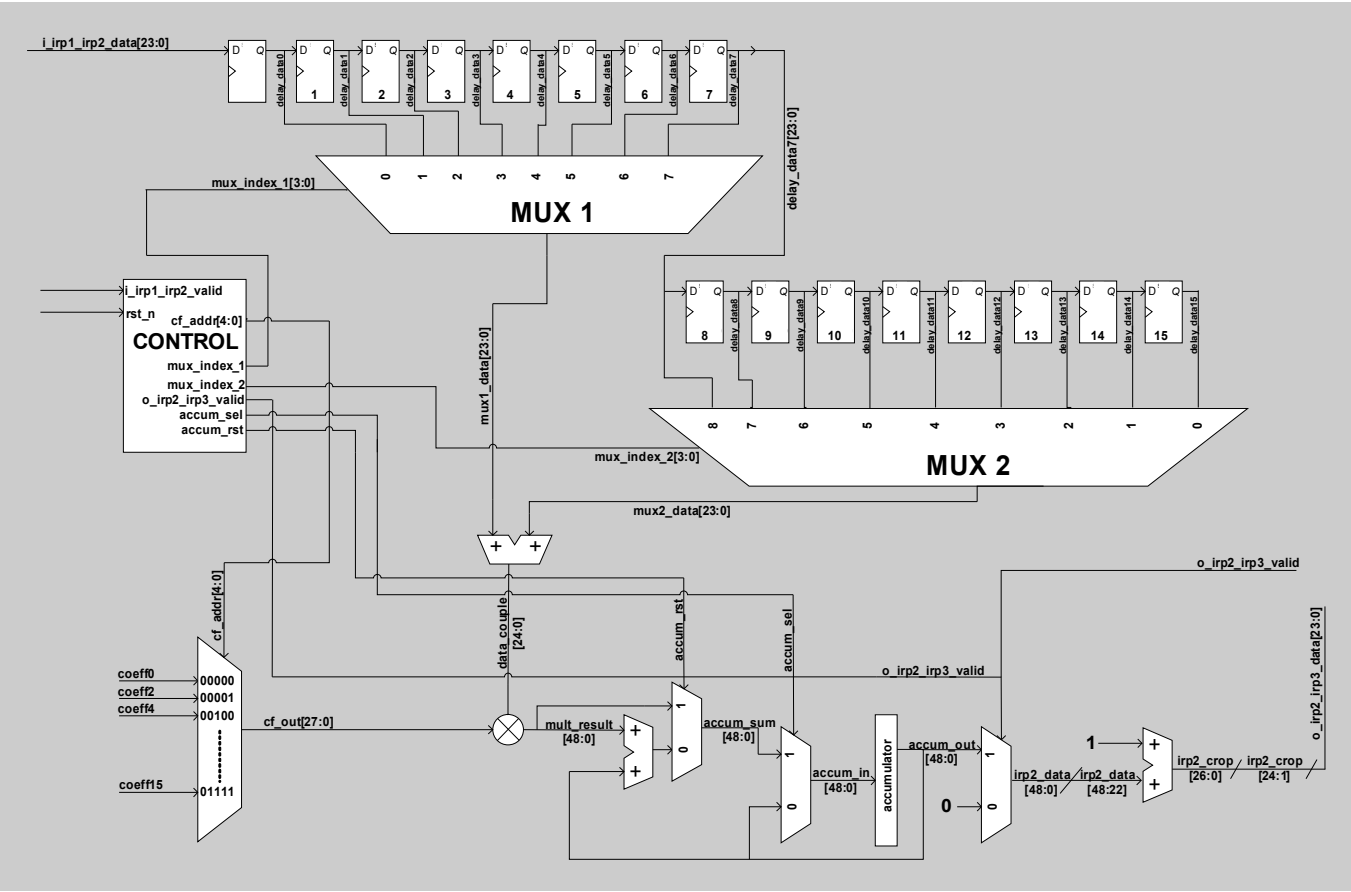

Figure 2.40: IRP2 Interface Diagram

Description

The second block of the interpolator is the IRP2 filter that interpolates the signal provided by IRP1 *i_irp1_irp2_data* by a factor of x2.

There are two polyphase filters that will perform the interpolation. The one that corresponds to the differential equation:

$$y_0(n)=h(0)x(n)+...+h(30)x(n-15) \tag{43}$$

and the other one that corresponds to the differential equation:

$$y_1(n)=h(1)x(n)+...+h(29)x(n-14) \tag{44}$$

Since the filters are symmetric it is convenient to add the input data and then multiply the result of the addition with the appropriate coefficient. The IRP2 filter's architecture is basically the same with the IRP1. The difference between IRP1 and IRP2 filters is that the second one is a Half Band filter. This comprises that all the even coefficients are set to zero except from the center one.

This property of having the even coefficients set to zero can be exploited by performing half of the multiplications. This is successed by using eleven states on the FSM of IRP2 from which the first nine are used to calculate the first half of the filter as well as the idle state, similar to the IRP1 approach, and the other two to calculate the second half of the filter by making the multiplication with the non zero even coefficient.

IRP2 filter as described previously has a total of 30 coefficients from which half of them are symmetric and half of them are zero except from the center one. Therefore the filter has nine non zero coefficients which are stored in the register file block. The first half of the filter calculates the first output with the odd non zero coefficients and the second half of the filter calculates the output of the non zero even coefficient.

The FSM of the IRP2 filter is described on the Table 2.11

state	mux_index1	mux_index2	cf_addr	o_irp2_irp3_valid	accum_sel	accum_rst	nx_state
state0	0	0	-1	'0'	'1'	'0'	state0 if i_valid='0' else state1
state1	0	0	0	'0'	'1'	'1'	state2
state2	1	1	2	'0'	'1'	'0'	state3
state3	2	2	4	'0'	'1'	'0'	state4
state4	3	3	6	'0'	'1'	'0'	state5
state5	4	4	8	'0'	'1'	'0'	state6
state6	5	5	10	'0'	'1'	'0'	state7
state7	6	6	12	'0'	'1'	'0'	state8
state8	7	7	14	'0'	'1'	'0'	state9
state9	0	0	-1	'1' if counter=63 else '0'	'0'	'0'	state10 if counter=63 else state9
state10	7	8	15	'0'	'1'	'1'	state11
state11	0	0	-1	'1' if counter=128 else '0'	'0'	'0'	state0 if counter=128 else state11

Table 2.11: IRP2 FSM state description table

The FSM of the IRP2 filter works with the same way that the FSM of the IRP1 filter does. The only difference between them is that the FSM of IRP2 has 11 states instead of 25. As we can see the second filter performs only one multiplication and waits for the counter to count 128 clock cycles.

A more detailed analysis of the input data and the coefficient relation is shown in the Table 2.12 above.

mux_index1	mux_index2	inputs	cf_addr	i_irp1_irp2_valid	o_irp2_irp3_valid
0	0	x(n)+x(n-15)	h0	**1**	0
1	1	x(n-1)+x(n-14)	h2	0	0
2	2	x(n-2)+x(n-13)	h4	0	0
3	3	x(n-3)+x(n-12)	h6	0	0
4	4	x(n-4)+x(n-11)	h8	0	0
5	5	x(n-5)+x(n-10)	h10	0	0
6	6	x(n-6)+x(n-9)	h12	0	0
7	7	x(n-7)+x(n-8)	h14	0	1
7	8	x(n-7)+x(n-7)	h15	0	1
0	0	x(n)+x(n-15)	h0	**1**	0
1	1	x(n-1)+x(n-14)	h2	0	0
2	2	x(n-2)+x(n-13)	h4	0	0

Table 2.12: IRP2 signal and time table

2.4.4 Half Band filter (IRP3)

The pin list along with the interface diagram of IRP1 will be shown in Table 2.13 and Figure 2.41 respectively.

Signal	I/O	Description
Clocks and reset		
clk	in	Master clock at 256Fs
rst_n	in	Active low asynchronous reset
CONTROL		
i_irp2_irp3_valid	in	Indicates that a new 24-bit input data is sampled at the input at 4*Fs rate
Input interface		
i_irp2_irp3_data [23:0]	in	24-bit input data from IRP2 at 4Fs
SINC interface		
o_irp3_sinc_data[23:0]	out	24-bit output interpolated data connected with SINC filter
o_irp3_sinc_valid	out	Indicates that a 24-bit output data is ready at the output at 8*Fs rate

Table 2.13: IRP3 Pin List

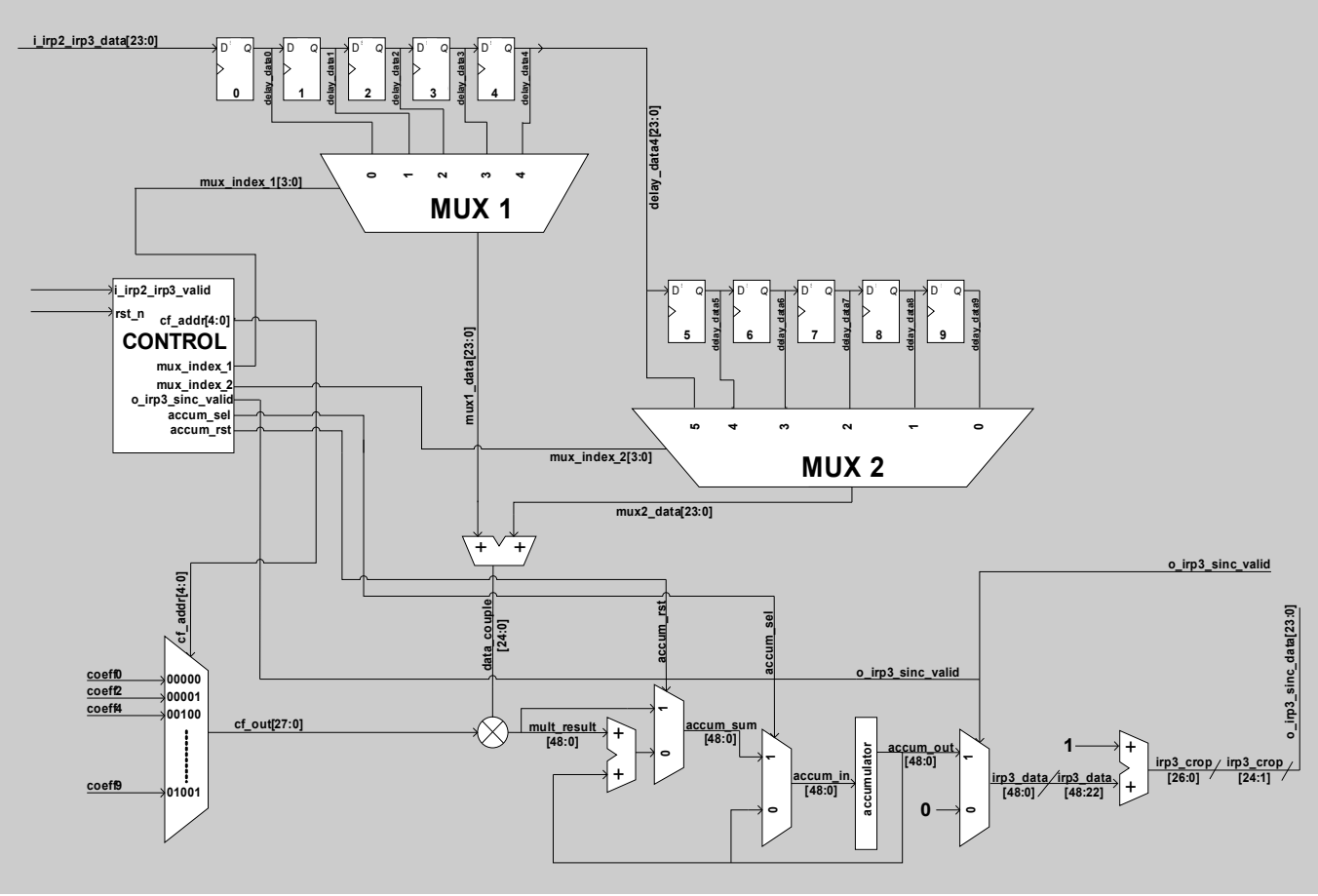

Figure 2.41: IRP3 Interface Diagram

Description

The third block of the interpolator is the IRP3 block which comprise a half band filter of 18th order. The input data are fed to IRP3 block through the *i_irp2_irp3_data* signal. As in IRP2 filter, the IRP3 filter follows exactly the same architecture. The IRP3 filter is split in two polyphase branches. The differential equations describing the filter are given below:

$$y_0(n)=h(0)x(n)+...+h(18)x(n-9) \tag{45}$$

$$y_1(n)=h(1)x(n)+...+h(17)x(n-8) \tag{46}$$

Since the filter has the same architecture as IRP2 resulting in symmetric coefficients along with the even ones to be zero except from the middle, the IRP3 block follows the rules described in the previous sub-block.

It is important to mention that the IRP3 filter has 6 non zero coefficients from which 5 belong to the first filter and the other one is the non zero coefficient of the second filter. Also the FSM of the IRP3 block has 8 states. The first state (state0) is the state in which the filter halts, the next six states make the proper calculations for the first half of the filter and the other two states make the proper calculations for the second half of the filter. Table 2.14 below depicts the FSM state assignment.

state	mux_index1	mux_index2	cf_addr	o_irp3_sinc_valid	accum_sel	accum_rst	nx_state
state0	0	0	-1	'0'	'1'	'0'	state0 if i_valid='0' else state1
state1	0	0	0	'0'	'1'	'1'	state2
state2	1	1	2	'0'	'1'	'0'	state3
state3	2	2	4	'0'	'1'	'0'	state4
state4	3	3	6	'0'	'1'	'0'	state5
state5	4	4	8	'0'	'1'	'0'	state6
state6	0	0	-1	'1' if counter=31 else '0'	'0'	'0'	state7 if counter=31 else state6
state7	4	5	9	'0'	'1'	'1'	state11
state8	0	0	-1	'1' if counter=64 else '0'	'0'	'0'	state0 if counter=64 else state8

Table 2.14: IRP3 FSM state description table

The input data and coefficient relation are presented in the Table 2.15

mux_index1	mux_index2	inputs	cf_addr	i_irp2_irp3_valid	o_irp3_sinc_valid
0	0	x(n)+x(n-9)	h0	**1**	0
1	1	x(n-1)+x(n-8)	h2	0	0
2	2	x(n-2)+x(n-7)	h4	0	0
3	3	x(n-3)+x(n-6)	h6	0	0
4	4	x(n-4)+x(n-5)	h8	0	1
4	5	x(n-4)+x(n-4)	h9	0	1
0	0	x(n)+x(n-9)	h0	**1**	0
1	1	x(n-1)+x(n-8)	h2	0	0

Table 2.15: IRP3 signal and time table

2.4.5 SINC filter (IRP4)

The pin list along with the interface diagram of IRP1 will be shown in Table 2.16 and Figure 2.42 respectively.

Signal	I/O	Description
Clocks and reset		
clk	in	Master clock at 256Fs
rst_n	in	Active low asynchronous reset
CONTROL		
i_irp3_sinc_valid	in	Indicates that a new 24-bit input data is sampled at the input at 8*Fs rate
Input interface		
i_irp3_sinc_data [23:0]	in	24-bit input data from IRP3
ΣΔ MODULATOR interface		
o_sinc_sd_data[23:0]	out	24-bit output interpolated data connected with ΣΔ modulator at 256*Fs rate

Table 2.16: SINC Pin List

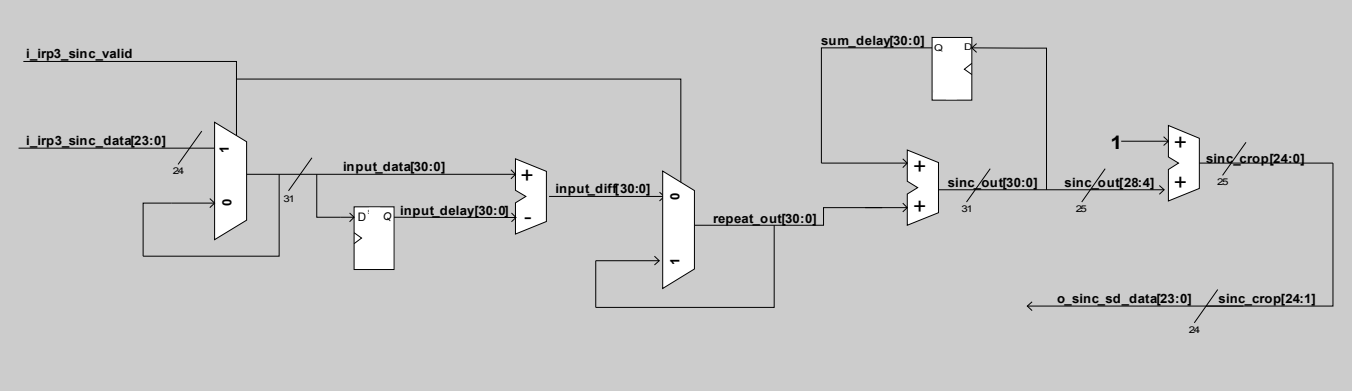

Figure 2.42: SINC Interface Diagram

Description

The fourth and last block of the interpolator is the SINC filter. Conversely to the other filters developed, the SINC filter doesn't involve multiplications. The differential equation describing the SINC filter is depicted below.

$$y(n)=x(m)-x(m-1)+y(n-1), n=1,2,3\ldots, m=\lfloor\frac{n}{32}\rfloor \quad (47)$$

Equation (47) describes that the output of the SINC filter depends on the current input which receives a new value each 32 clock cycles, the previous input value and on the previous output value.

From the Figure 2.42 we can see that the SINC block calculates its output by subtracting the *input_diff* signal with the *input_delay* signal (x(m) – x(m-1) operation) and assigning the result to the *input_diff* signal. If the *i_irp3_sinc* signal is set to '0' then the *repeat_out* signal is set to the *input_data* otherwise it holds its previous value. The *repeat_out* signal is the repetition of the subtracted input signal for 32 times. The final output is the sum of the *repeat_out* signal and the *sum_delay* signal (x(m) – x(m-1) + y(n-1) operation). The result is assigned to the *sinc_out* singal.

Because the SINC filter makes multiple additions and subtractions, the *input_data* as well as the rest of the signals operating in the filter, are sign extended from 24-bit to 31-bit in order for the filter to avoid overflows. The final output is calculated by cropping the *sinc_out* signal and holding the bits 28 down to 4. The cropped signal is assigned to the *sinc_crop* signal which is of width 25. The *sinc_crop* signal is finally added with an ace and the bits 24 down to 1 are kept and assigned to the *o_sinc_sd_data* signal.

The SINC filter transmits data every clock cycle except from the reception of reset, where the output of the SINC filter is set to zero. Therefore the SINC filter does not need a valid signal to indicate the transmition of the output.Table 2.17 describes the input data and signal relation.

o_sinc_sd_data	i_irp3_sinc_data	repeat_out	o_sinc_mod_data	i_irp3_sinc_valid
y(0)	x(0)	x(0)-x(-1)	y(-1)+x(0)-x(-1)	**1**
y(1)	x(0)	x(0)-x(-1)	y(0)+x(0)-x(-1)	0
y(2)	x(0)	x(0)-x(-1)	y(1)+x(0)-x(-1)	0
y(3)	x(0)	x(0)-x(-1)	y(2)+x(0)-x(-1)	0
y(4)	x(0)	x(0)-x(-1)	y(3)+x(0)-x(-1)	0
y(5)	x(0)	x(0)-x(-1)	y(4)+x(0)-x(-1)	0
y(6)	x(0)	x(0)-x(-1)	y(5)+x(0)-x(-1)	0
y(7)	x(0)	x(0)-x(-1)	y(6)+x(0)-x(-1)	0
y(n)	x(m)	x(m)-x(m-1)	y(n-1)+x(m)-x(m-1)	0
Y(32)	x(1)	x(1)-x(0)	y(31)+x(1)-x(0)	**1**
Y(33)	X(1)	x(1)-x(0)	y(32)+x(1)-x(0)	0

Table 2.17: SINC signal and time table

2.5 Verification plan

The MATLAB/SIMULINK model based on fixed point arithmetic is used to perform the verification process of the RTL model. The verification process is based on the comparison of the high level model output (MATLAB/SIMULINK) and RTL (Modelsim) output.

The output of each filter and the final output of the interpolator is stored in a file for both cases. The verification procedure is terminated successfully, if each MATLAB/SIMULINK output file matches exactly the corresponding Modelsim output file. The comparison procedure is performed using the tool KDiff3.

In Figure 2.43 the fixed point model of the interpolator is displayed. The initial floating point model is modified so that each filter can produce an output to the workspace. Each output of the MATLAB model is multiplied with 2^{23} in order to obtain a 24 bit number .

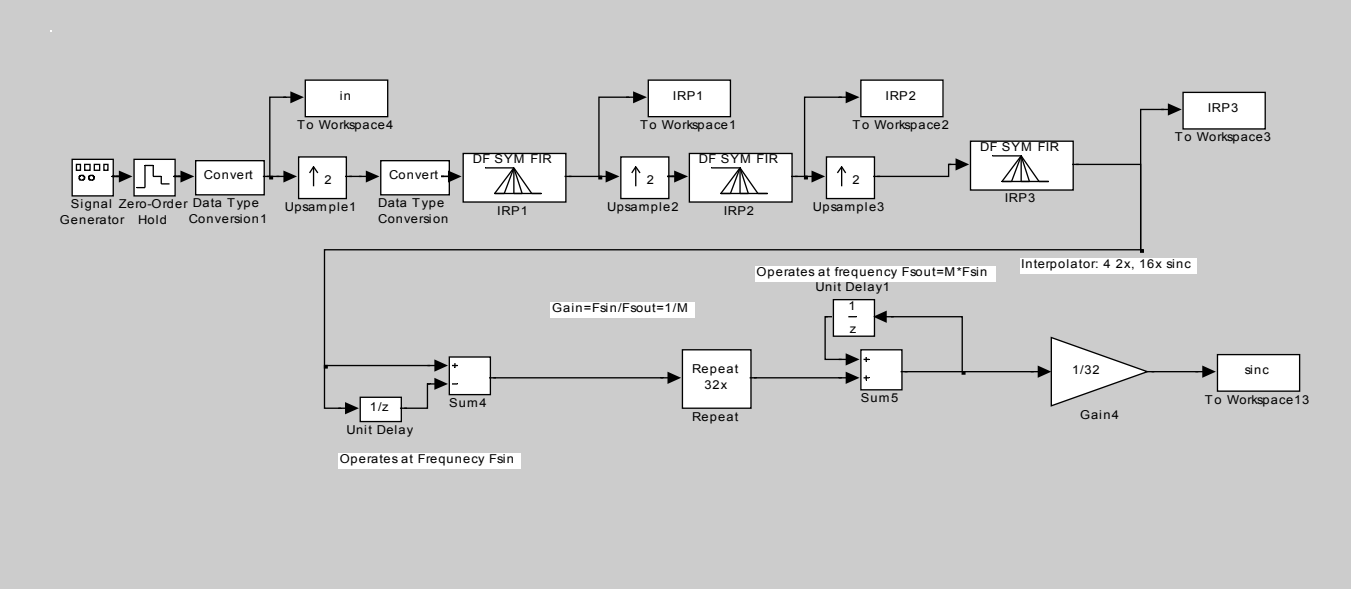

Figure 2.43: Fixed point model of interpolator

2.6 References

[1] Multirate Digital Signal Processing, Ronald E. Crohiere, Lawrence R. Rabiner, 1983, p129-181

[2] Delta-Sigma Data Converters: Theory, Design and Simulation, Steven R. Norsworthy, Richard Schreier, Gabor C. Temes, First edition, 1996, p: 406-446

[3] A Multibit Delta-Sigma Audio DAC with 120dB Dynamic Range, Ichiro Fujimori, Tetsuro Sugimoto, IEEE Journal of Solid-State Circuits, VOL.35 NO 8, August 2000.

3 ΣΔ Modulator

The ΣΔ modulator is the core of the ΣΔ DAC placed between the interpolator and the internal analog DAC. The next paragraphs will describe the theory of ΣΔ modulation, the different issues of the design, the architecture and the implementation of this block.

3.1 ΣΔ modulator theory

In a ΣΔ DAC, a signal sampled at the Nyquist frequency is initially processed by an interpolation stage. The interpolator changes the data rate and suppresses the spectral replicas which occur from the upsampling process. This signal feeds the noise-shaping loop, which shortens the word length to a single bit or to a few bits (case of multibit DAC).

In practice, ΣΔ modulation is a technique for converting an oversampled digital signal into an analogue one by integrating differential signals. In this section the basic operation of ΣΔ modulation is described. The modulator subtracts the quantization noise of the previous input from the current input and quantizes the signal produced. The general case of a ΣΔ modulator is shown in Figure 3.1 where u(n) is the input signal and y(n) is the output signal.

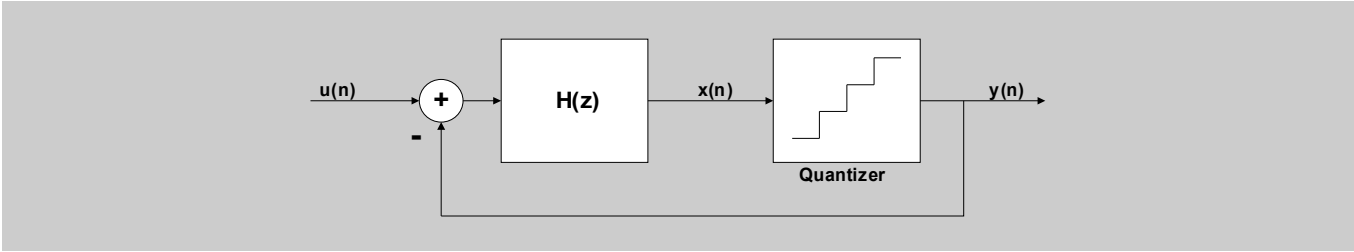

Figure 3.1: ΣΔ topology

The topology of Figure 3.1 can be analyzed by using the linear model shown in Figure 3.2, where the quantization noise e(n) is considered to be an independent input random signal independent from the input.

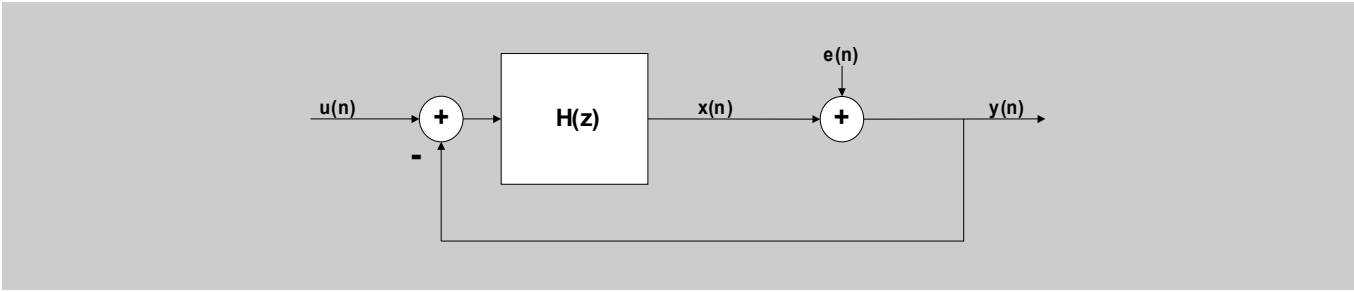

Figure 3.2: Linear model of ΣΔ topology

The model is characterized by the following transfer functions:

$$S_{TF}(z)=\frac{Y(z)}{U(z)}=\frac{H(z)}{1+H(z)} \tag{48}$$

$$N_{TF}(z)=\frac{Y(z)}{E(z)}=\frac{1}{1+H(z)} \tag{49}$$

where $S_{TF}(z)$ is the signal transfer function, $N_{TF}(z)$ is the noise transfer function and E(z) is the quantization noise in the frequency domain. Considering the axiom of superposition and using equations (48) and (49), the output signal can be written as:

$$Y(z)=S_{TF}(z)U(z)+N_{TF}(z)E(z) \tag{50}$$

In order to shape the quantization noise to high frequencies, we have to consider that H(z) has a large magnitude between 0 and f_0, (where f_0 is the input signal frequency) and that the input signal must remain within the maximum levels of the feedback signal y(n), otherwise the large gain in H(z) will cause x(n) to saturate.

3.1.1 First order noise shaping

For the first order ΣΔ modulator we should have a zero at DC for the N_{TF}, in order to high pass filter the quantization noise, since the poles of H(z) are equal to the zeros of $N_{TF}(z)$. Thus, we choose:

$$H(z)=\frac{1}{z-1} \tag{51}$$

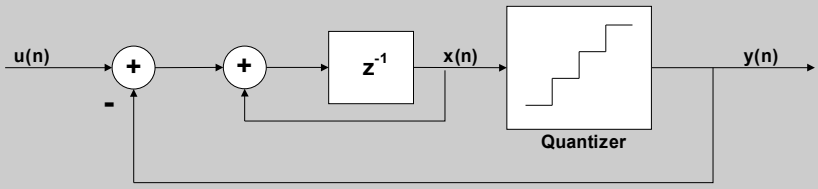

Figure 3.3: First order ΣΔ modulator

The equations describing the modulator in the time domain are given by:

$$X(n)=X(n-1)+U(n-1)-Y(n-1) \tag{52}$$

$$Y(n)=sign(X(n)) \tag{53}$$

where Y(n) is the quantized output signal, U(n) is the input signal and Y(n) – X(n) is the quantization noise, namely the difference between the input signal and the output signal.

At the frequency domain, using equation (51), the signal transfer function is:

$$S_{TF}(z)=\frac{Y(z)}{U(z)}=\frac{H(z)}{1+H(z)}=\frac{\frac{1}{z-1}}{1+\frac{1}{z-1}}=z^{-1} \tag{54}$$

and the noise transfer function is:

$$N_{TF}(z)=\frac{Y(z)}{E(z)}=\frac{1}{1+H(z)}=\frac{1}{1+\frac{1}{z-1}}=(1-z^{-1}) \tag{55}$$

To calculate the SNR, we have to estimate the signal power and the quantization noise power. If we let $z=e^{i\omega T}=e^{j2\pi f/f_s}$, where F_s is the sampling frequency we have:

$$N_{TF}(f)=1-e^{-j2\pi f/f_s}=\frac{e^{j\pi f/f_s}-e^{-j\pi f/f_s}}{2\mathrm{j}}\times 2\mathrm{j}\times e^{-j\pi f/f_s}=\sin(\frac{\pi f}{f_s})\times 2\mathrm{j}\times e^{-j\pi f/f_s} \tag{56}$$

Taking the magnitude of both sides:

$$|N_{TF}(f)|=2\sin(\pi f/f_s) \tag{57}$$

The quantization noise over the frequency band from 0 to f_0 is given by:

$$P_e=\int_{-f_0}^{f_0} S_e^2(f)|N_{TF}(f)|^2\,df=\int_{-f_0}^{f_0}\left(\frac{\Delta^2}{12}\right)\frac{1}{f_s}\left[2\sin\left(\frac{\pi f}{f_s}\right)\right]^2 df \tag{58}$$

where $S_e(f)=\frac{\Delta^2}{12}\frac{1}{f_s}$ is the spectral density of the quantization noise and Δ the difference between two adjacent quantization levels. Since OSR>>1, then $f_0<<f_s$ and $\sin(\pi f/f_s) \approx \pi f/f_s$, we can write equation (58):

$$P_e=\frac{\Delta^2\pi^2}{36}\left(\frac{1}{OSR}\right)^3 \tag{59}$$

The signal power is given by the equation:

$$P_s=\frac{\Delta^2 2^N}{8} \tag{60}$$

because the maximum peak value of the input signal without clipping is $2^N(\Delta/2)$ where 2^N are the quantizer levels. Combining equations (59) and (60) the SNR becomes:

$$SNR_{max}=10\log\left(\frac{P_s}{P_e}\right)=6.02\mathrm{N}+1.76-5.17+30\log(OSR) \tag{61}$$

3.1.2 3rd order ΣΔ D/A modulator

The case of our study will be a third order ΣΔ modulator with error feedback structure. This model is shown in Figure 3.4:

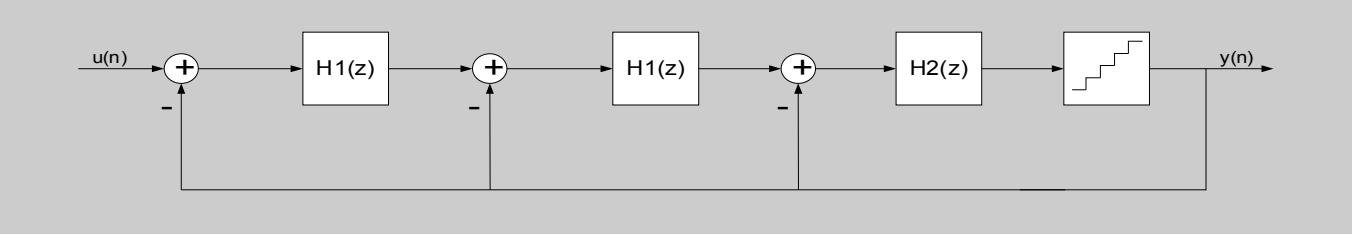

Figure 3.4: Third order ΣΔ modulator

where the integrator blocks (H1(z)) and the delayed integrator block (H2(z)) are shown in Figure 3.5.

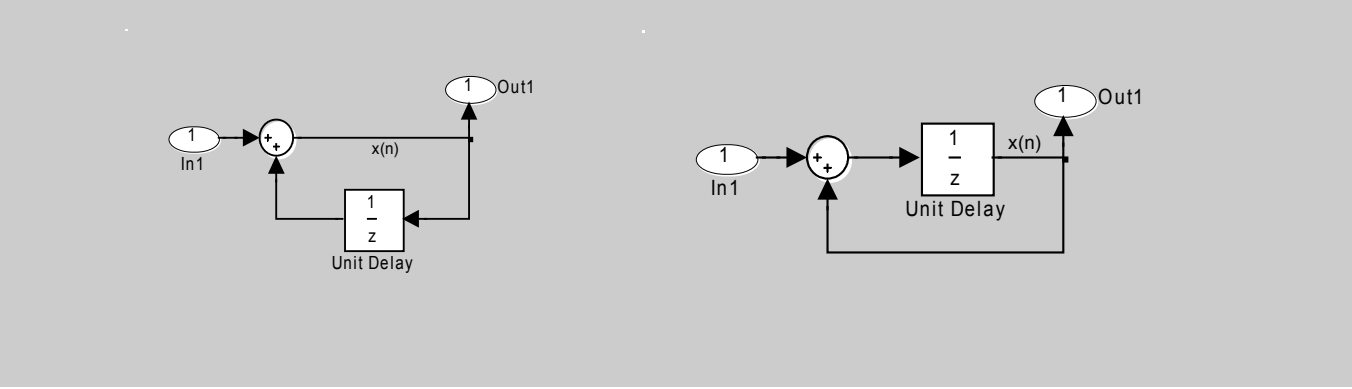

Figure 3.5: Integrator and delayed integrator

The MATLAB/SIMULINK model of the third order modulator is presented in Figure 3.6, containing as sub-blocks the integrator and the delayed integrator as described previously.

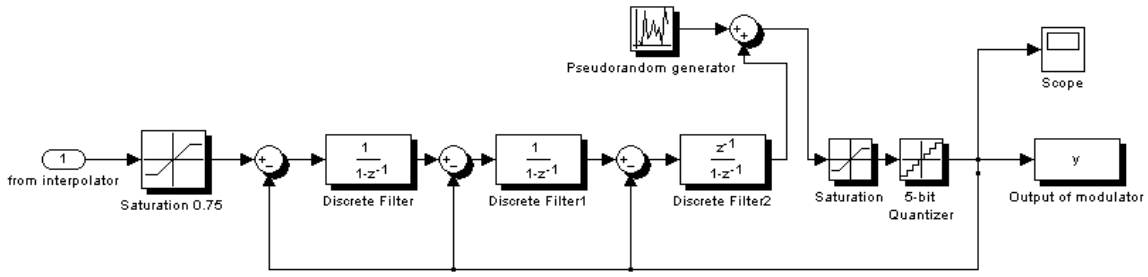

Figure 3.6: Simulink model of a third order ΣΔ modulator

In the third order modulator, the transfer functions are described by the following equations[1]:

$$S_{TF}(z)=\frac{Y(z)}{U(z)}=z^{-1} \tag{62}$$

$$N_{TF}(z)=\frac{Y(z)}{E(z)}=(1-z^{-1})^3 \tag{63}$$

The SNR of a third order modulator is given by the following equations, using the transfer functions mentioned above and following the same analysis as for the 1st order modulator:

$$|N_{TF}(z)|=[2\sin(\frac{\pi f}{f_s})]^3 \tag{64}$$

$$P_e=\int_{-f_0}^{f_0} S_e^2(f)|N_{TF}(f)|^2 df=\int_{-f_0}^{f_0}\frac{\Delta^2}{12}\frac{1}{f_s}\left[2\sin(\frac{\pi f}{f_s})\right]^6 df=\frac{32\Delta^2\pi^6}{21}\left(\frac{1}{OSR}\right)^7 \tag{65}$$

$$P_s=\frac{\Delta^2 2^{2N}}{8} \tag{66}$$

The maximum theoretical SNR is calculated to be:

$$SNR_{max}=10\log(\frac{P_s}{P_e})=6.02N-76+70\log(OSR) \tag{67}$$

The advantage of the noise shaping and the oversampling is the increase of the SNR. Taking N=5 for (67) the SNR is reduced to:

$$SNR=-45.9+70\log(OSR) \tag{68}$$

We consider that the input signal of WIDTH bits takes both positive and negative values. This signal is driven through two integrators and one delayed integrator. The output signal is then quantized to 5 bits. The feedback we need in order to make the subtraction is WIDTH-1 bits, so we shift WIDTH-6 bits the output when assigning it to the feedback.

3.1.3 The problem of stability

Higher order modulators suffer from instability problems because of the overload of the quantizer. In high order modulators, the input range must be a few dB below the full scale range of the feedback DAC.

1 In order to maintain the specified signal transfer function, we place as showed in Figure 3.6, two integrators ($1/(1-z^{-1})$) to avoid having a transfer function like z^{-3}.

It is obvious that when the input reaches the limit of the quantizer, without overloading, the addition of the quantization error, may cause range overload. This will result a multiple quantizer overload and because the error feedback will be saturated, the modulator will be saturated too for the rest of its inputs, especially if the next input is similar to the previous one. For any input u(n) of the modulator with an M step quantizer, if the following condition is true, the modulator will not experience overload.

$$\max_{n} |u(n)| \leq M + 2 - \|h\|_1 \tag{69}$$

where $\|h\|_1 = \sum_{n=0}^{\infty} |h(n)|$ and h(n) is the inverse z-transform of the noise transfer function H(z). For a third order modulator having a 32 level quantizer, the maximum value of u(n) should be 78% of the quantizer's full scale value. In Figure 3.7, a 3rd order ΣΔ modulator is oscillating since the quantizer is overloaded when an input signal of amplitude 0.9 (instead of a theoretical maximum of 0.78) is inserted.

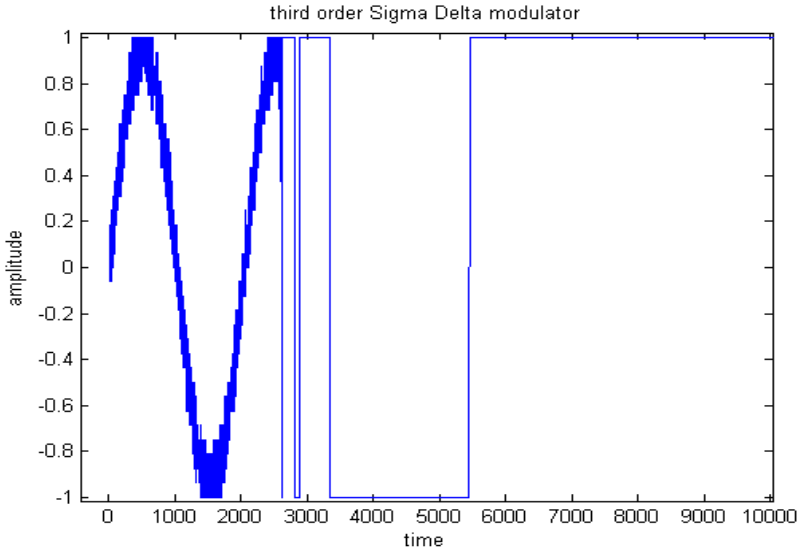

Figure 3.7: Overloaded 3rd order ΣΔ modulator

3.1.4 Idle Tones and dithering

Idle tones are produced, when the modulator's input is a DC signal or a very low frequency signal and periodic patterns may be present. When this is the case, assuming that the output has a period of n cycles, there will be an f_s/n signal placed over it. At this case a tone is produced at f_s/n and the low pass filter will not suppress the tone, because this signal will remain inside the band of interest. In our case, where the input is not a DC signal, idle tones can appear when there is a slowly varying input for a certain period which is the same as having a DC signal and can produce periodic output. In order to avoid this effect, the use of dithering is recommended.

Dithering is the act of importing some random or pseudo random signal into the modulator before the quantization stage to avoid idle tone generation. The random signal injected will not alter the modulator's output, because it will be noise shaped in the same manner as the quantization noise. Its amplitude will be smaller than the modulator's amplitude.

The generation of pseudo-random signals is based on the use of a Linear Feedback Shift Register (LFSR). LFSR produces a pseudo random signal which is repeated every 2^N-1 where N is the number of registers been used. LFSR will implement a 35 bit polynomial from which the 19 MSBs will be fed to the output. By implementing a 35 bit polynomial, we get larger appearance sequence of the random signal than having a 19 bit polynomial. Figure 3.10 shows the amplitude of dither compared to the input signal:

Dither signal will therefore not affect the spectral output of the modulator since the dither signal is also noise shaped. If the dither has large magnitude it can overload the quantizer resulting in instability.

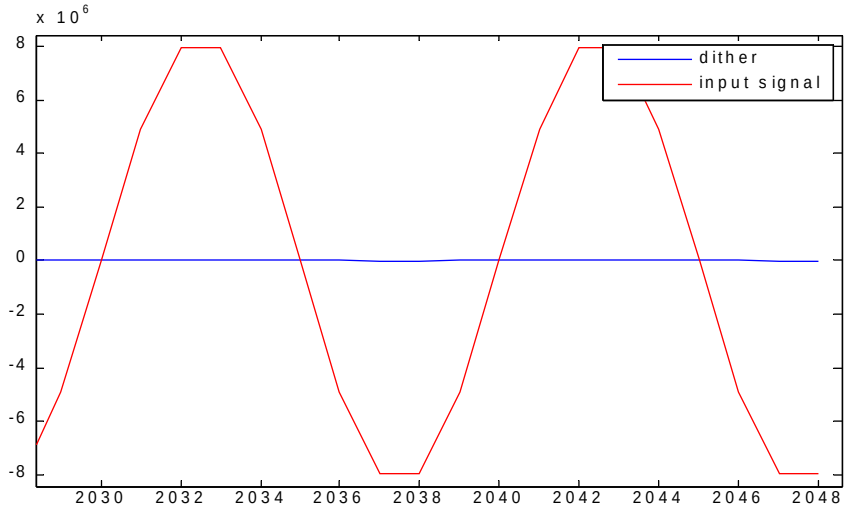

Figure 3.8: Input-dither signals amplitude relation

3.2 MATLAB Simulation Results

To better understand the performance difference when raising the order of a modulator,an example is illustrated in the following paragraph presenting a first order ΣΔ modulator and a third order.

In a SIMULINK model, a signal generator is used to produce a sinusoidal signal of amplitude 0.7 and of frequency Fin=750 Hz. The signal is then inserted into a zero order hold block, to virtually upsample the signal up to OSR*Fs with Fs=6000 Hz and OSR=256. After the integration the signal is driven into a saturation block which keeps the input level of the quantizer saturated between -1 and 1. The saturation is necessary due to the large gain of the transfer function. The quantizer has 5 bits (32 levels). Figure 3.9 Shows the performance results:

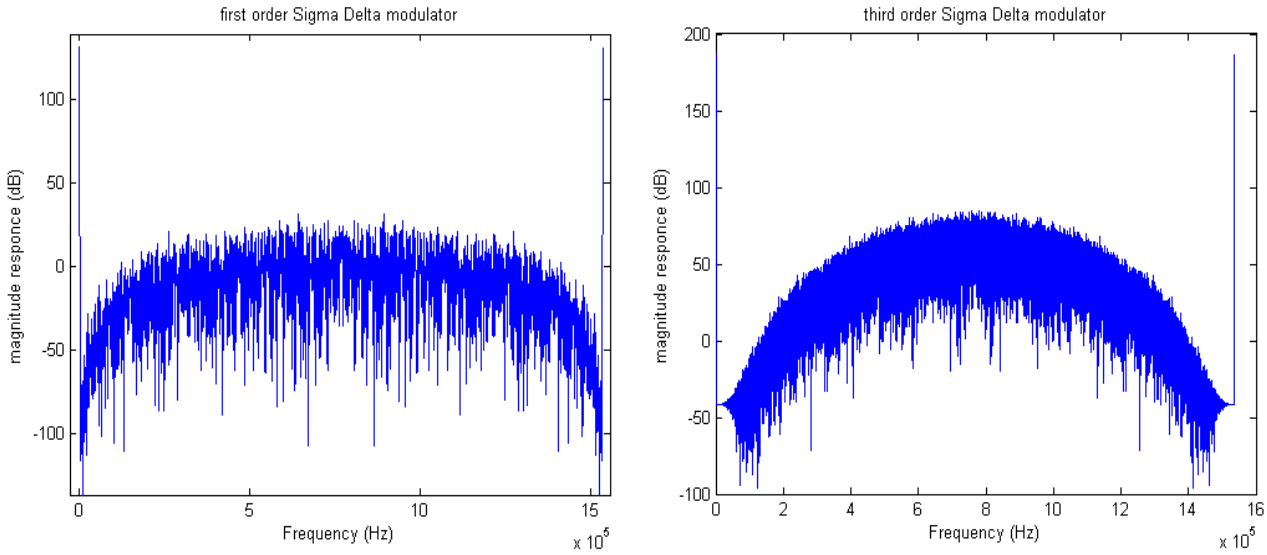

Figure 3.9: Magnitude response of a first and third order ΣΔ modulators

Figure 3.9 illustrates the improvement of the SNR when the order of the modulator is increased. If we take a closer look at Figure 3.9 we will see that the quantization noise power of the third order modulator is very small in our area of interest (up to 1KHz) in contrast with the first order. Besides, equation (68) verifies the expected SNR performance with the use of a third order modulator and a 5-bit quantizer.

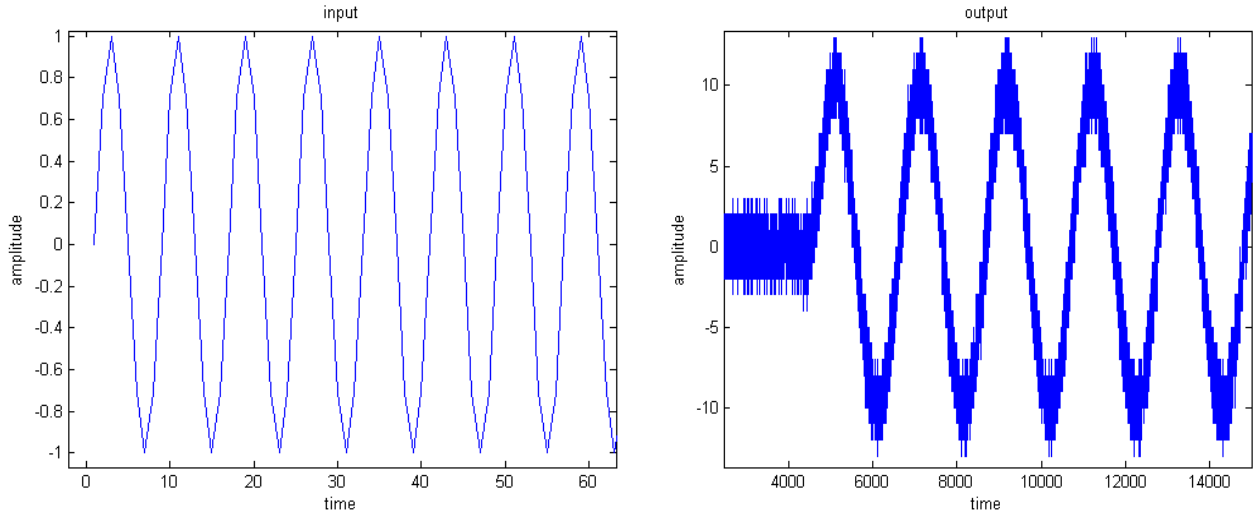

Figure 3.10: input and quantized output of a sinusoidal signal

3.3 H/W Implementation

This chapter will present the implementation considerations of the ΣΔ modulator block. RTL architecture will be presented along with I/O interfaces, block diagrams and design details.

3.3.1 Procedural diagram

The ΣΔ modulator is composed of two basic sub-blocks as shown in Figure 3.11. The first sub-block is the integrator and the second sub-block is the delayed integrator. The quantizer block actually truncates the signal and holds the 5 MSBs therefore it will be not considered as a separate block from the top level design. These 5 bits are shifted and sign-extended. Before the output of the modulator enters the quantizer, the output of the dither block is added. If i_dither_enable is set to '1' then dithering is enabled, otherwise the dither block output is set to zero and the modulator operates as normal.

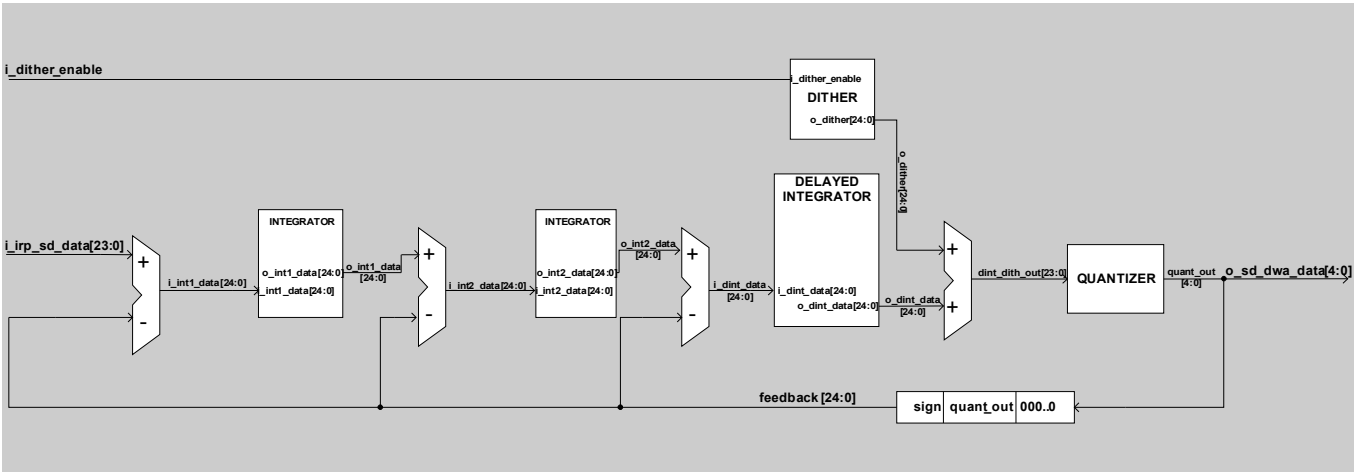

Figure 3.11: ΣΔ MODULATOR Procedural Diagram

In Figure 3.12 the I/O block diagram of the modulator is presented.

The detailed pin list of the ΣΔ modulator block is shown in Table 3.1.

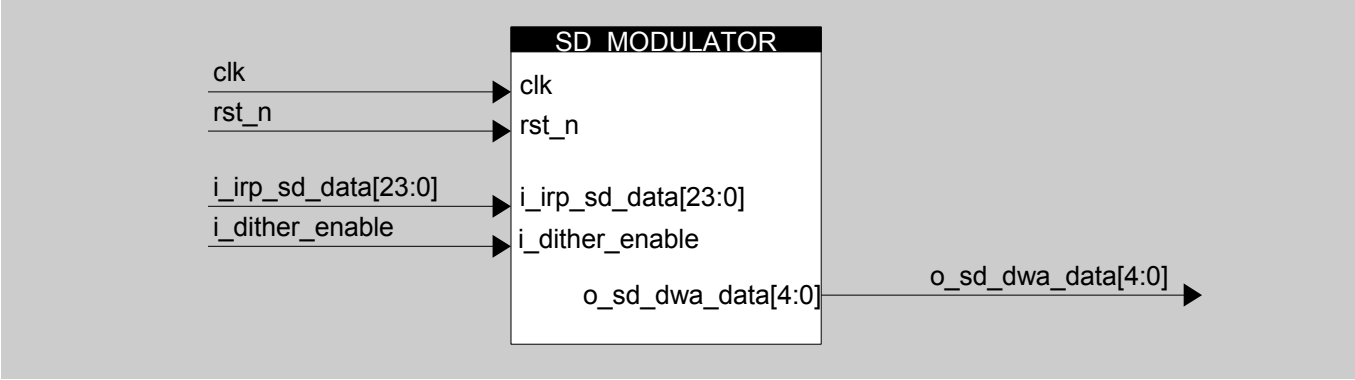

Figure 3.12: ΣΔ MODULATOR Block Diagram

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock at 256Fs
rst_n	in	Active low asynchronous reset
CONTROL		
i_dither_enable	in	External, Synchronized from upper level
Interpolator stage interface		
i_irp_sd_data [23:0]	in	Input data from interpolator
Internal DAC interface		
o_sd_dwa_data [4:0]	out	Output data to DWA block

Table 3.1: ΣΔ MODULATOR Block Pin List

3.3.2 Overflow Detection

Before continuing to the description of the implementation of the ΣΔ modulator, a critical issue concerning overflows has to be presented. For signal handling, a two's complement representation is considered. As far as additions and subtractions are concerned, overflow detection should be incorporated. The additional logic used to detect an arithmetic overflow is to perform a XOR operation between the last two carry bits of the adder. If the result of the XOR gate is '0', no overflow occurred, otherwise if the result it is '1' an arithmetic overflow occurred. When an overflow occurs, the adder outputs a result, so that the modulator continues to operate normally.

The approach is to detect if the overflow is either positive or negative. If the overflow is positive then the last two carry will be “01”, if the overflow is negative there will be “10”. For the first case the result will be the maximum positive number which in our case is “0111...1” and for the second case “1000...0”. Thus, we can assure that the output of the adder is the closest one to the number expected.

Also, the stability of the modulator has to be taken into consideration. Therefore a saturation occures before the input signal enters the modulator. The maximum value of the input signal must be of 75% of the total input range according to the stability theory and equation (69)

3.3.3 Integrator sub-block

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock at 256Fs
rst_n	in	Active low reset synchronized from upper level
Input interface		
i_int_data [24:0]	in	Input data
Output interface		
o_int_data[24:0]	out	Output data

Table 3.2: Integrator sub-block Pin List

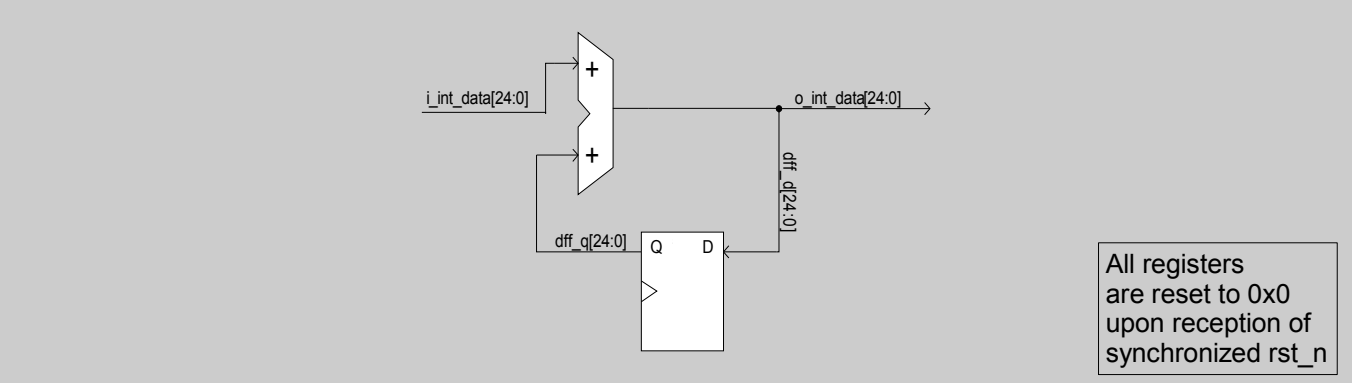

Figure 3.13: Integrator sub-block Interface Diagram

Description

The integrator sub-block is composed by a D flip-flop, an adder and a multiplexer. The input data is summed with the previous input delayed data and are driven through the multiplexer. The timing diagram of the integrator sub-block is given below in Figure 3.13

Adder functionality

The adder sub-block of the integrator has to take into account an arithmetic overflow in the two's complement representation. In order to allow overflow detection, the adder makes an XOR operation between the last two carry bits. If the XOR result is 0 (1 XOR 1, 0 XOR 0) there is no overflow, otherwise if the XOR result is 1 (1 XOR 0), overflow is detected.

When an overflow occurs, the adder has to decide whether it was a positive or a negative overflow. If the summed numbers exceed the maximum positive value of the adder (01111...1) then the last two carry bits are “01” and when the added numbers exceed the maximum negative value (10000...0) the last two carry bits are “10”. In case of a positive overflow, the maximum allowed positive value is assigned to the result whereas in the case of a negative overflow, the maximum negative allowed value is assigned to the result.

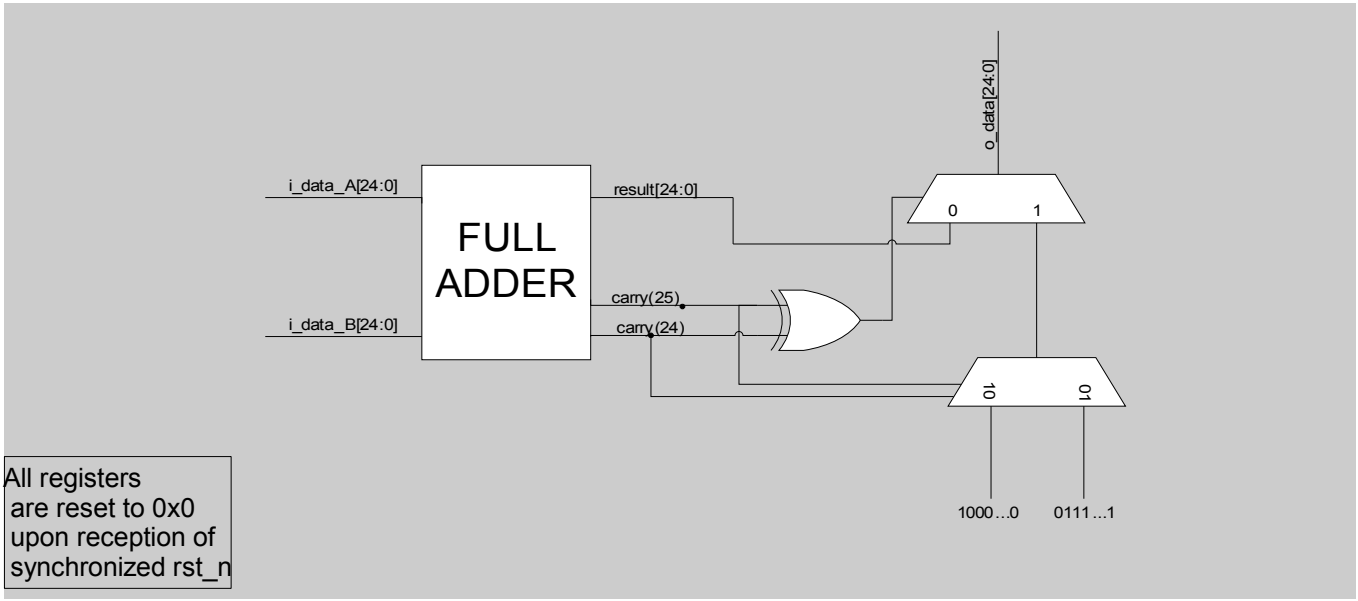

Figure 3.14: Adder sub-block Interface Diagram

3.3.4 Delayed Integrator sub-block

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock
rst_n	in	Active low reset synchronized from upper level
Input interface		
i_dint_data [24:0]	in	Input data
Output interface		
o_dint_data[24:0]	out	Output data

Table 3.3: Delayed Integrator sub-block Pin List

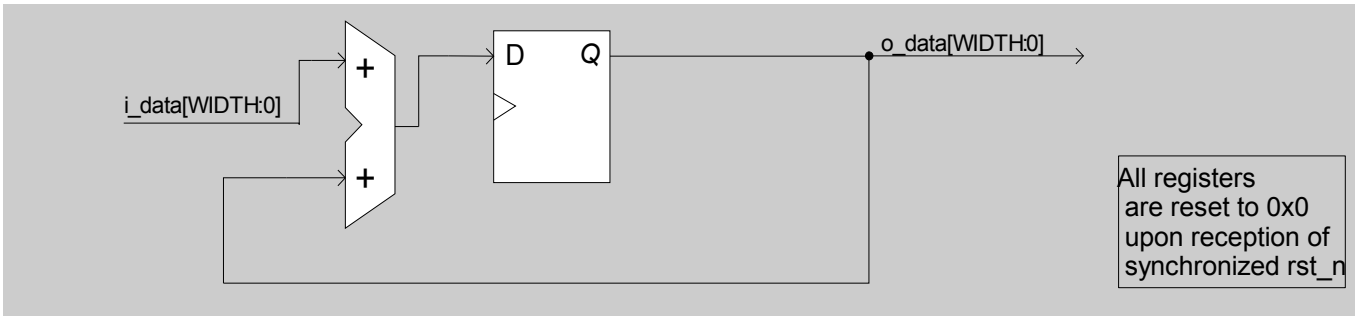

Figure 3.15: Delayed Integrator sub-block Interface Diagram

Description

The Delayed Integrator sub-block, is composed by a D flip flop, a multiplexer and an adder as shown in Figure 3.15. The basic functionality of this sub-block, is to sum the input data with the output data and delay them through the D flip flop. The same architecture described previously concerning the adder overflow is also applied to the adder of the delayed integrator.

3.3.5 Dither sub-block

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock
rst_n	in	Active low reset synchronized from upper level
CONTROL		
i_dither_enable	In	External, Synchronized from upper level
Output interface		
o_dither[24:0]	out	Output data

Table 3.4: Dither sub-block Pin List

The following block implements the Fibonacci polynomial $x^{35}+x^{34}+x^{28}+x^{27}+1$

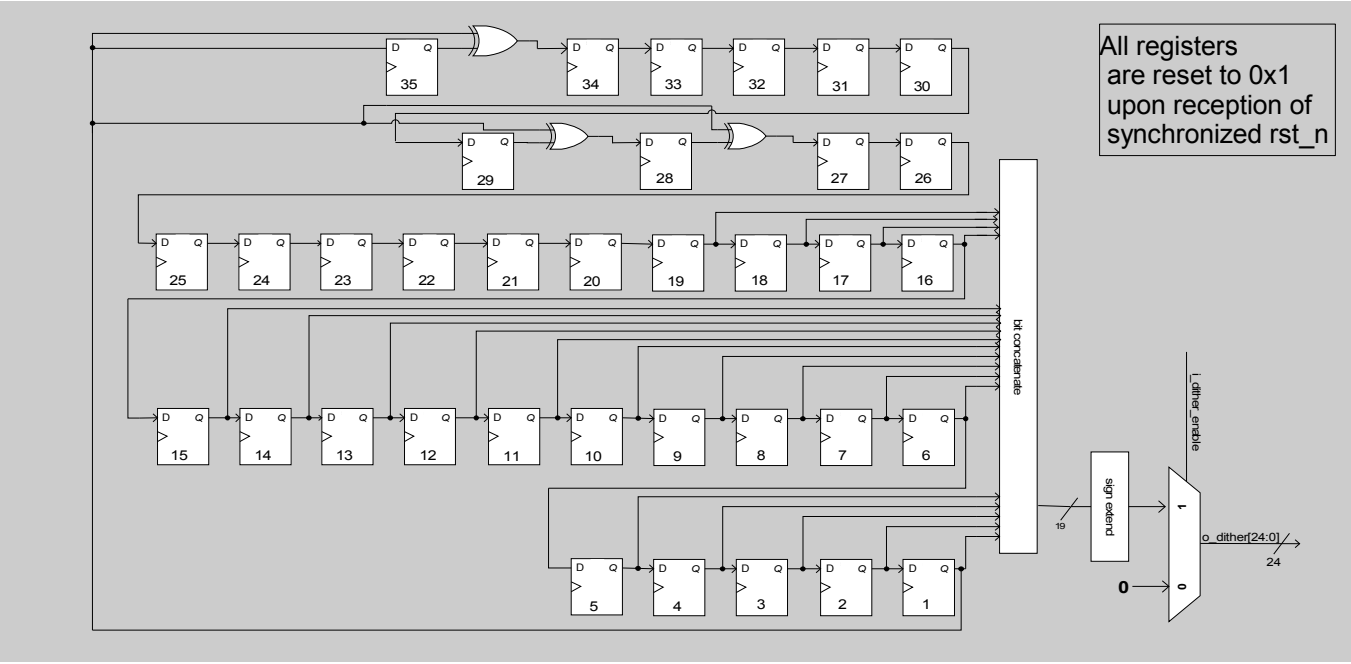

Figure 3.16: Dither sub-block Interface Diagram

Description

The Dither sub-block is composed by 35 registers (D flip-flop). The output of each register is connected to the input of the next register. However, registers 34,28 and 27 get their output through an XOR gate connected with the input of the first register. Thus, a pseudo random number generator is created whose 35 bit pattern is repeated after 2^{35}-1 cycles. From the 35 bits of the output, the 19 LSB's are selected. This 19-bit signal is sign extended from 19 bits to 24 and sent to the input of the modulator. If the i_dither_enable signal is set to '0' the dither sub-block does not produce any output and the o_dither signal is set to zero. Otherwise, dither is enabled and the output generates random signals. Table 3.5 shows an example of the dither block output.

i_dither_enable	o_dither
0	00000000000000000000000111
0	00000000000000000000000011
0	00000000000000000000000001
1	00000000000000000000000000
0	00000000000000000000000000
0	11111110000000000000000000
0	11111111000000000000000000
0	11111111100000000000000000
0	11111111110000000000000000
0	11111111111000000000000000
0	11111111111100000000000000
0	11111111111110000000000000
0	11111111111111000000000000

Table 3.5: Dither block output example

3.4 Verification plan

The implementation of the ΣΔ modulator block is followed by the verification of the results produced. The verification procedure is implemented, as in the interpolator block, with the use of the fixed point SIMULINK model provided.

The procedure requires for both the RTL and the MATLAB models, to receive the same input. After the simulation ends, the output files of the Modelsim model and the SIMULINK model are compared with the KDiff3 tool. If the files match, the RTL follows the MATLAB model of the modulator.

The fixed point SIMULINK model of the ΣΔ modulator block is depicted in Figure 3.17. The simulation is performed without the use of the dither block because it is difficult to be simulated in MATLAB and give the exact same results with the Modelsim.

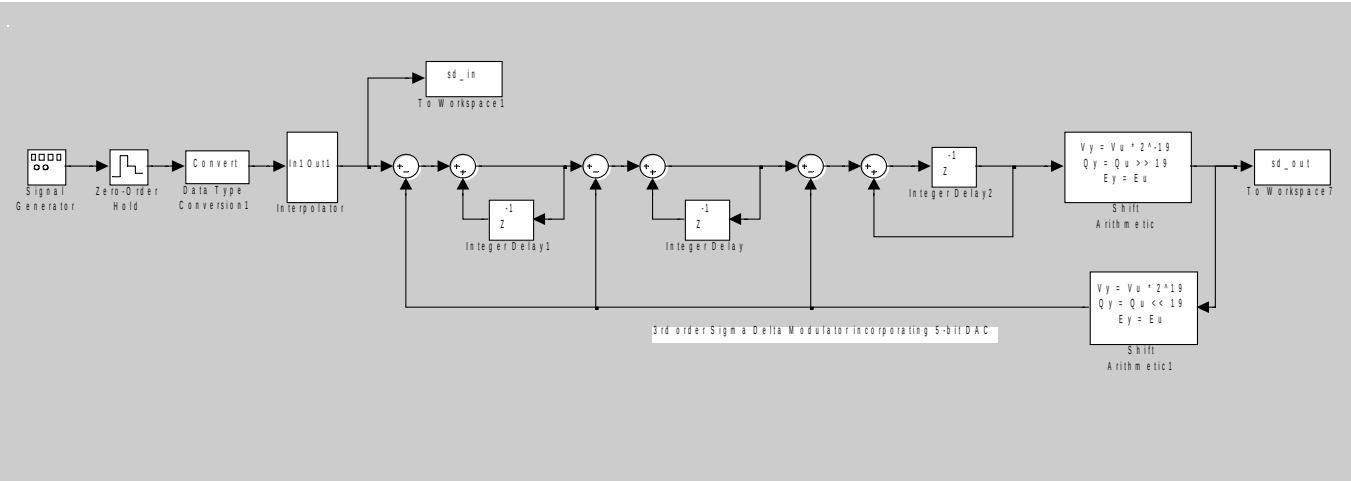

Figure 3.17: Fixed point model of ΣΔ modulator

3.5 References

[1] Delta-Sigma Data Converters: Theory, Design and Simulation, Steven R. Norsworthy, Richard Schreier, Gabor C. Temes, First edition, 1996, p: 309-316

[2] *Data Converters*, Franko Maloberti, First edition, 2007, p: 1-73, 253-298

[3] *Analog Integrated Circuit Design*, David Johns, Ken Martin, First edition, 1996, p: 531-551

[4] A Multibit Delta-Sigma Audio DAC with 120dB Dynamic Range, Ichiro Fujimori, Tetsuro Sugimoto, IEEE Journal of Solid-State Circuits, VOL.35 NO 8, August 2000.

[5] A 14-bit, 10-Msamples/s DAC Converter Using Multibit ΣΔ modulation, Katayoun Falakshahi, Chih-Kong Ken Yang, Bruce A. Wooley, IEEE Journal of Solid-State Circuits, VOL.34, NO 5 May 1999.

4 Data Weighted Averaging

The use of a multibit ΣΔ modulator implies the use of an internal multibit DAC. This component whether implemented with resistors or capacitors exhibits non-linear characteristics because of mismatch. This chapter describes the theory and analysis of different dynamic element matching techniques that are used to cope with the problem. The different algorithms are explained and the Data Weighted Averaging Algorithm that has been finally chosen is presented along with theoretical and simulation results.

4.1 DWA theory

4.1.1 Internal DAC topology

A digital to analog converter outputs an analog signal A that is related with the digital input D through equation (70). An appropriate value for A is chosen depending on the digital input. In practice this a D is a single number without dimensions, a sets the range of A.

$$A = aD \tag{70}$$

Considering that a can be a current (similar with voltage or charge) quantity, I_{REF} for example, the analog output can be expressed as:

$$A = I_{REF} D \tag{71}$$

The digital input fed to the converter can be either in binary, thermometer or other suitable format as shown in Table 4.1.

Decimal	0	1	2	3
Binary code	00	01	10	11
Thermometer code	0000	0001	0011	0111

Table 4.1: formats of DAC conversion

In the binary format an m-bit binary number $D_{m-1} D_{m-2} \ldots D_0$ can be represented in decimal as $D_{m-1} 2^{m-1} + D_{m-2} 2^{m-2} + \ldots + D_0 2^0$. In the thermometer code the decimal value indicates the amount of consecutive ones.

These formats are useful when current-steering DACs are examined. The major distinction is made between binary and thermometer current steering DAC. In order to choose the appropriate topology, we will present two of the most basic metrics concerning the static linearity of a converter. The Differential Nonlinearity (DNL) and the Integral Nonlinearity (INL) are the metrics that will be discussed. DNL is the maximum deviation in the output step size from the ideal value normalized to one LSB. On the other hand, INL is the maximum deviation of the input-output characteristic from a straight line passing through its end points. These two metrics are useful to measure the monotonicity of the converter. In general the monotonicity is guaranteed when INL is between ±0.5 LSB. In Figure 4.1 there are shown the INL and DNL metrics in the input-output characteristic of a signal.

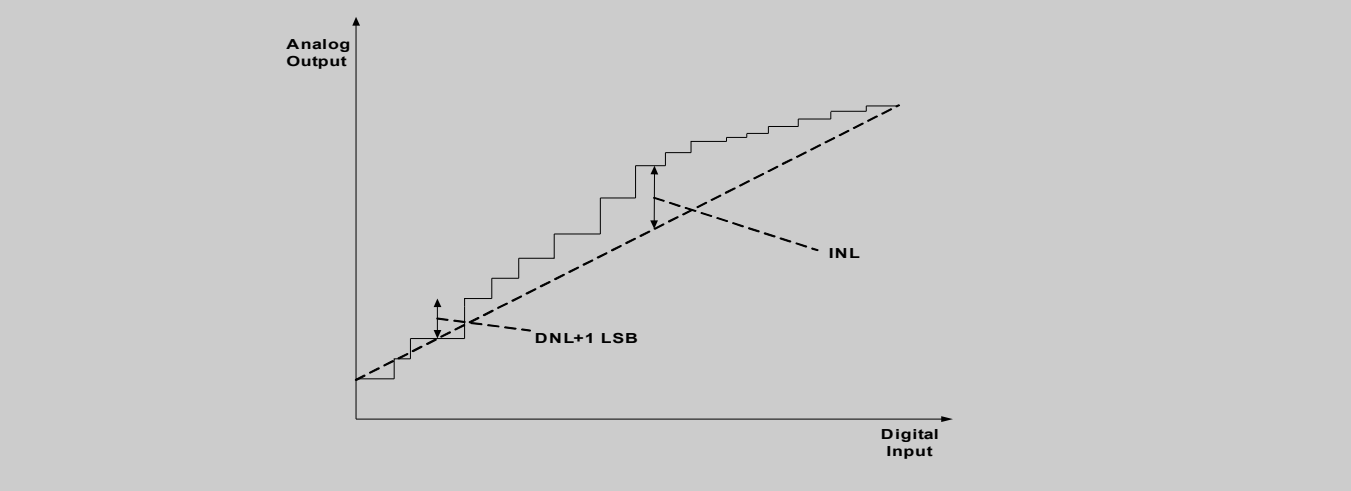

Figure 4.1: Input-output diagram

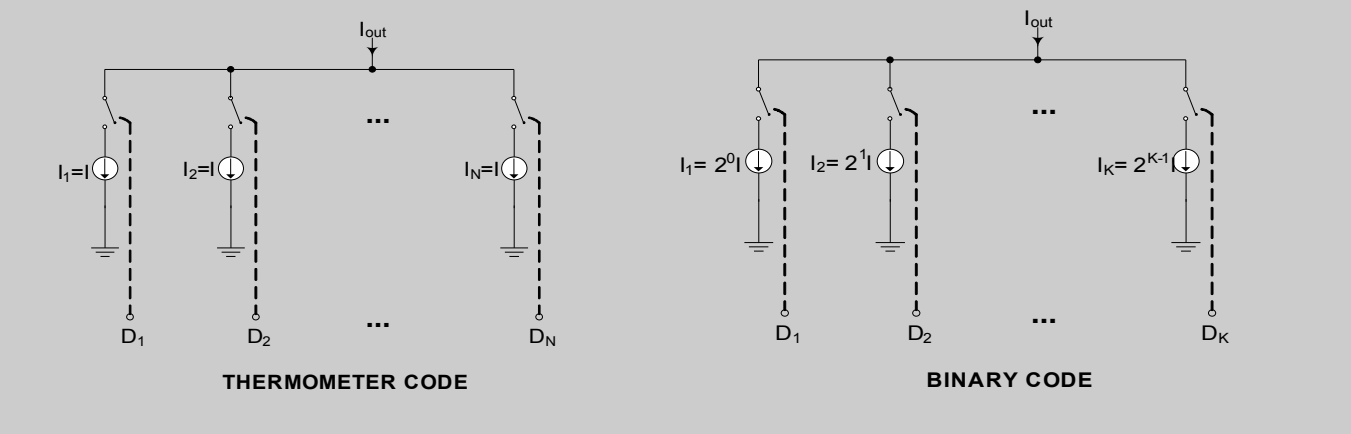

Figure 4.2: DAC topologies

The topologies that can be used are:

Binary topology

The most simple architecture for digital to analog conversion is based on the binary topology shown in Figure 4.2. It uses K current sources and each source steers different current. Each current source is controlled by Di that corresponds to the i-source. The output current is the summation of all the currents depending on the value of control signal Di. The output current is given by:

$$I_{out} = D_1 2^0 \cdot I + D_2 2^1 \cdot I + D_3 2^2 \cdot I + \ldots + D_K 2^{K-1} \cdot I \quad (72)$$

where Di may receive the values 0 or 1.

This topology is simple and generic since it has many variations (resistor ladder, R-2R et.c).

On the other hand, when using a binary code topology the DAC suffers from nonlinearity. It is possible that for a given transition (of the type 01111..1 → 1000...0) all the current sources are switched on or off simultaneously and this may cause unwanted glitches. This strongly affects the DNL performance.

Thermometer topology

Thermometer topology, shown in Figure 4.2 is a similar approach differing to the amount of current that each current source steers. Similar to the binary topology, in the thermometer architecture, the thermometer code drives N current sources, where N is the representation of the N-bit binary signal. Each current source drives equal amount of current I, and the output current of the DAC is the summation of these currents.

The thermometer code topology overcomes the problem of differential non linearity because the

analog output increases or decreases by one current source each time. A DAC using thermometer code must have a total of 2^{N-1} current sources in order to represent all the possible codes. The price paid for this type of topology is the decode logic added on the digital part of the DAC.

4.1.2 Dynamic Element Matching Algorithms

In a ΣΔ DAC where multibit quantization is used, non-linearity of the internal DAC can severely degrade the overall DAC performance.

These errors are usually caused by the element mismatch such as mismatch on capacitors, resistors or transistors due to random variations or gradients. These random errors can be eliminated by applying different algorithms or circuit topologies. In recent design the use of a dynamic element matching algorithm is preferred. Some of them are presented in the following sections.

Butterfly randomization

One of the simplest methods of element matching is the randomization of the element that will be selected. By this method, the elements that will be selected are independent by the time nor the input signal. The basic principle of this approach is that the mismatch error at one time will not be the same with the mismatch error at any other time. Therefore the mismatch error will be converted into a white noise.

Random element matching can be performed by using a butterfly network by coupling the inputs to the outputs, having at least a number of butterfly stages equal to the number of the bits of the converter. The input of the network is the thermometer code of the signal. By opening or closing the switch of each stage, the bit follows a path which ends at the current source that the bit will be represented. The switches are set by a random number of M bits where M is the number of stages in the network This type of element matching is not preferred for large thermometer codes because of the complexity needed for the network. Also when the number of stages is big, a more complex randomizer will be needed (for example LFSR) which adds more complexity. A three stage butterfly network is depicted in Figure 4.3 with 8 elements and three bit control.

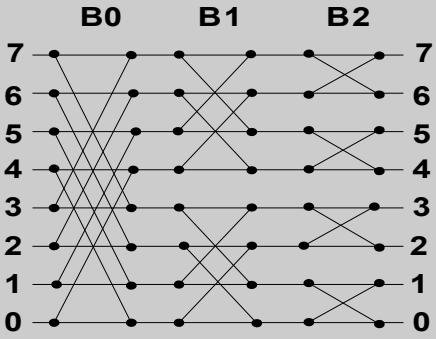

Figure 4.3: 3 stage Butterfly network

It can be proved that the Butterfly randomization algorithm has a theoretical performance given by equation (73).

$$SNR = \frac{3\mathrm{M}}{OSR \cdot \sigma_E^2} \tag{73}$$

Where M is the number of elements used, OSR is the oversampling ratio of the converter (it is set to 1 if the sampling is performed in Nyquist rate) and σ_E is the variance of the mismatch error.

Individual Level Averaging

Individual level averaging (ILA) method is aiming at using each element with equal probability for each input code. This algorithm is using a table of indexes for each input code I_K. For each input code K, the element I_K, I_K +1,... I_K +K-1 is used. If the index exceeds the number of the elements, the indexing begins from the beginning of the elements.

There are two methods of selecting the elements according to each indexes value, the rotation and the addition.

The rotation approach increases the index of the code K, I_K by one each time the code occurs and the addition method increases the index by K. The arrays on the Table 4.2 below present the index values and the element selection according to the input values for the rotation method and Table 4.3 for the addition method.

Time – Index	input	Indexes							Element						
		I_1	I_2	I_3	I_4	I_5	I_6	I_7	1	2	3	4	5	6	7
1	5	1	1	1	1	1	1	1	●	●	●	●	●		
2	6	1	1	1	1	2	1	1	●	●	●	●	●	●	
3	3	1	1	1	1	2	2	1	●	●	●				
4	5	1	1	2	1	2	2	1		●	●	●	●		
5	2	1	1	2	1	3	2	1	●	●					
6	3	1	2	2	1	3	2	1		●	●	●			
7	6	1	2	3	1	3	2	1		●	●	●	●	●	●
8	5	1	2	3	1	3	3	1			●	●	●	●	●
9	5	1	2	3	1	4	3	1	●			●	●	●	●

Table 4.2: ILA rotation method

Time – Index	input	Indexes							Element						
		I_1	I_2	I_3	I_4	I_5	I_6	I_7	1	2	3	4	5	6	7
1	5	1	1	1	1	1	1	1	●	●	●	●	●		
2	6	1	1	1	1	6	1	1	●	●	●	●	●	●	
3	3	1	1	1	1	6	7	1	●	●	●				
4	5	1	1	4	1	6	7	1	●	●	●			●	●
5	2	1	1	4	1	4	7	1	●	●					
6	3	1	3	4	1	4	7	1				●	●	●	
7	6	1	3	7	1	4	7	1	●	●	●	●	●		●
8	5	1	3	7	1	4	6	1	●			●	●	●	●
9	5	1	3	7	1	1	3	1	●	●	●	●	●		

Table 4.3: ILA addition method

The final outcome of the ILA method is that after a certain amount of time, all the sources are used equal times.

Data Weighted Averaging

The third approach of the dynamic element matching algorithms is the Data Weighted Averaging (DWA). The basic idea of the DWA algorithm is that it uses only one index which is updated with the addition of the new input code to the context of the index register. The index indicates the position of the first bit of the input data. Therefore as can be seen in Table 4.4 the thermometer code stream occupants the bits of the output beginning from the next bit of the previous output last bit.

The advantage of the DWA algorithm against the ILA is that it is faster thanks to the usage of only one single index instead of K. Table 4.4 shows the operation of the DWA algorithm.

Time – Index	input	index	Element						
			1	2	3	4	5	6	7
1	5	1	●	●	●	●	●		
2	6	6	●	●	●	●		●	●
3	3	5					●	●	●
4	5	1	●	●	●	●	●		
5	2	6						●	●
6	3	1	●	●	●				
7	6	4	●	●		●	●	●	●
8	5	3			●	●	●	●	●
9	5	1	●	●	●	●	●		

Table 4.4: DWA algorithm

It can be shown that a first order noise shaping of the mismatch error is performed using the DWA algorithm. We assume that the value of each element is $X_i = \bar{X} + \delta X_i$, the summation of the mismatch error δX_i and the mean value of the elements. The mean value of the mismatch error is assumed to be 0.

$$\sum_{1}^{M} \delta X_i = 0 \tag{74}$$

Using the mismatch error that corresponds to each current source as a random variable, we can define (75) and (76).

$$\Delta_i(k) = \sum_{i}^{i+k-1} \delta X_k \qquad for\, i+k-1 < M \tag{75}$$

$$\Delta_i(k) = \sum_{i}^{M} \delta X_k + \sum_{1}^{i+k-1-M} \delta X_k \qquad for\, i+k-1 > M \tag{76}$$

The equations above describe the total error produced by the elements for converting an input number k while the index points at i.

We can observe that the mismatch error tends to 0 when all the current sources have been used. We use the term cycle to refer to this. For sake of clarity the following example is presented.

Consider a DAC using 8 elements for the analog conversion. Also suppose an input sequence of the numbers 3,4,3. Figure 4.4 displays the element usage according to the DWA algorithm.

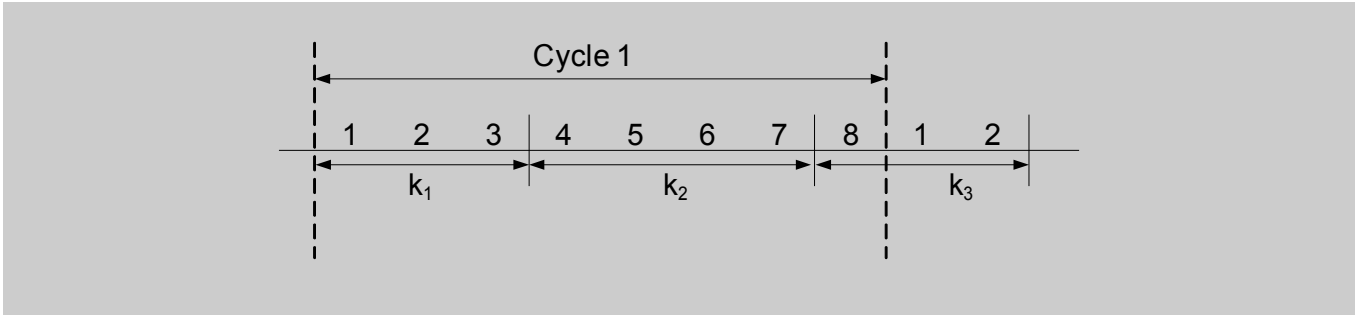

Figure 4.4: DWA example using 8 elements

The noise injected at the first input is $\Delta_1(k_1)=\delta X_1+\delta X_2+\delta X_3$, at the second input is $\Delta_2(k_2)=\delta X_4+\delta X_5+\delta X_6+\delta X_7$. By splitting the mismatch noise of the third input into $\Delta_3'(k_3)=\delta X_8$ and $\Delta_3''(k_3)=\delta X_1+\delta X_2$ we have for the first cycle a total mismatch error of

$$\Delta_1(k_1)+\Delta_2(k_2)+\Delta'_3(k_3)=0 \tag{77}$$

Expressing the mismatch error of the second input as a function of the other two we have:

$$\Delta_2(k_2)=-[\Delta_1(k_1)+\Delta'_3(k_3)] \tag{78}$$

Since, k3 is the code corresponding at time n, k2 corresponds at time (n-1) and code k1 corresponds at time (n-2) we can write in the z-domain we can convert the total mismatch error

$$-z^{-1}\Delta_1(k_1)(1-z^{-1})+\Delta_3'(k_3)(1-z^{-1}) \tag{79}$$

Equation (79) shows the first order noise shaping that takes place during DWA algorithm. This can be applied to the rest of the inputs by separating the mismatch error when needed in order for the total mismatch error of each cycle to become zero.

The DWA method is simulated in the MATLAB-SIMULIKNK environment for the ΣΔ DAC giving the following results in the frequency domain.

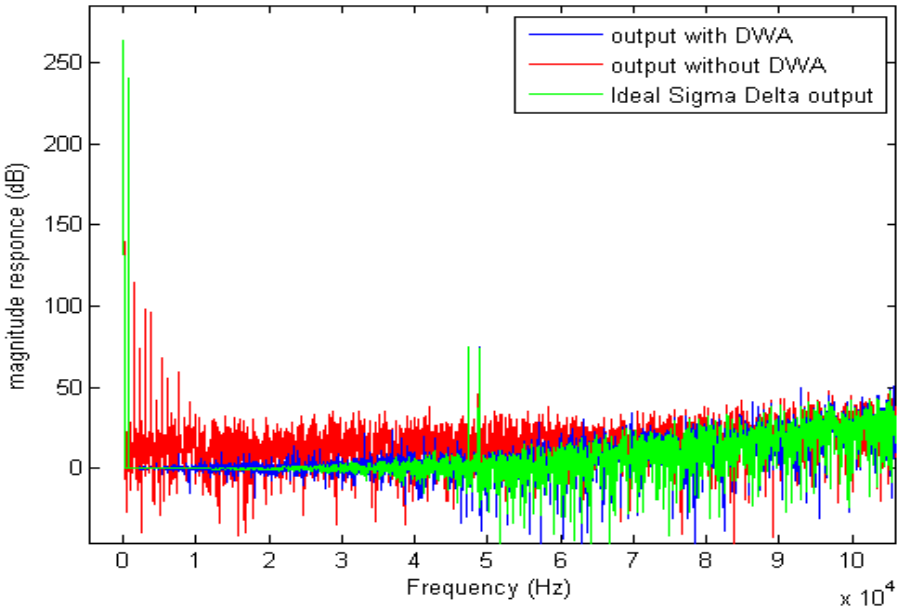

Figure 4.5: DWA method contribution in magnitude respoce

In Figure 4.5 the output of the model is displayed. In the SIMULINK model a mismatch error is added at the output of the ΣΔ modulator resulting the magnitude response displayed with the red line. The magnitude response of the ΣΔ modulator without the mismatch error is displayed with the green line

whereas the output of the mismatch error shaped signal using the DWA method is displayed with the blue line. It is obvious that the DWA algorithm eliminates the errors added by the element mismatch.

The SIMULINK model is displayed in Figure 4.6 and Figure 4.7. The model uses the dwa function in which a mismatch error is added in each element. Also in the dwa function the DWA algorithm is performed which uses an index pointing the starting element which will represent the input in thermometer code as described previously.

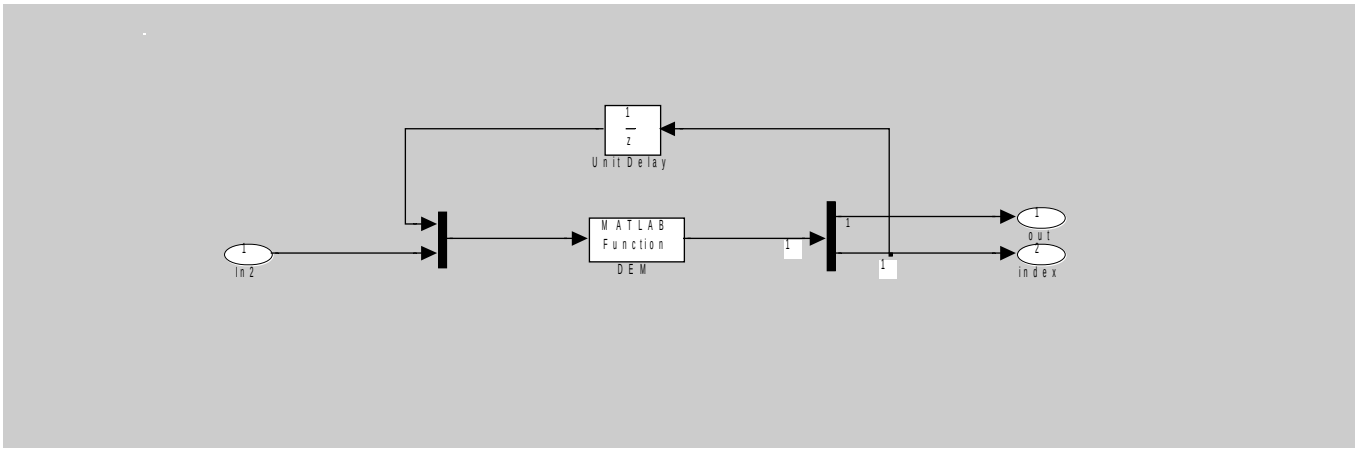

Figure 4.6: DWA model

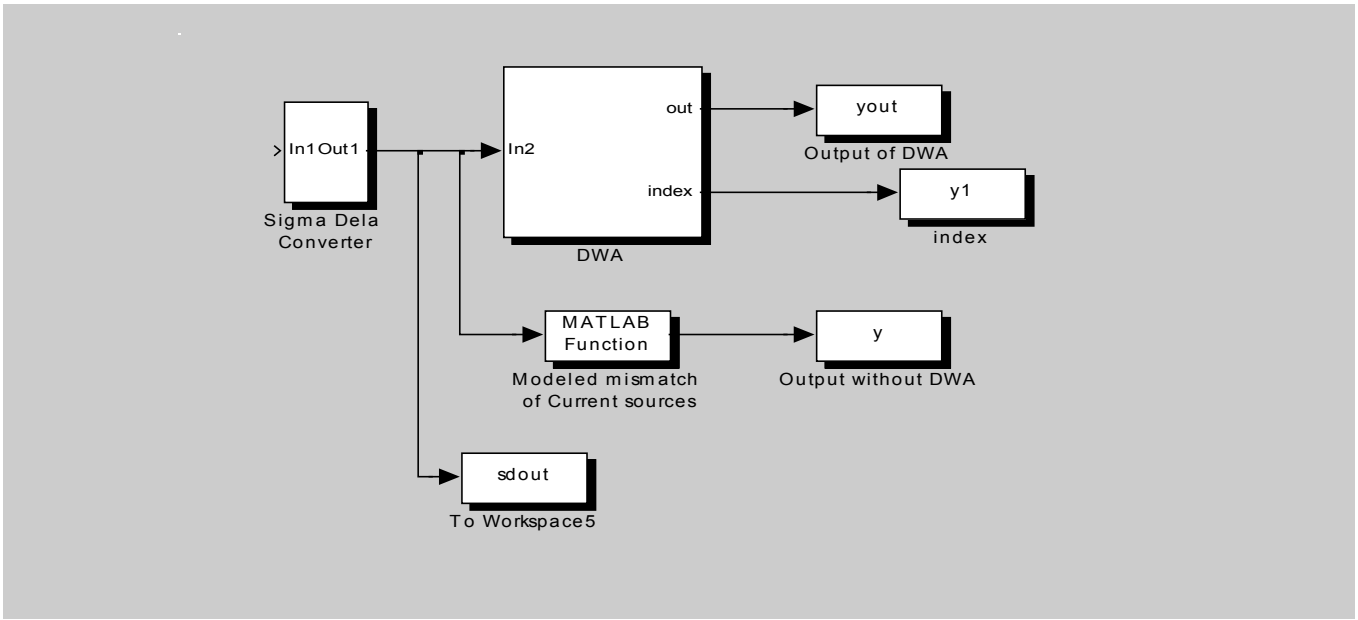

Figure 4.7: Topology of model with and without using DWA algorithm

4.2 Architecture

The section that follows will present the basic concept principles of the reasons that lead us to the specific architecture.

DAC Topology

The topology that will be used in the converter will be the one described in the previous sections and is composed by a thermometer code translator and a block that will perform the dynamic element matching. As already described, this architecture is better because the current sources that will be used will be equivalent which leads to lower element mismatch errors.

The Dynamic Element Matching algorithm that will be used is the DWA algorithm. Because the output of the ΣΔ modulator is a 5 bit signal it will be translated to a 32 bit signal after the thermometer code block.

Figure 4.8 displays the topology of the DWA block

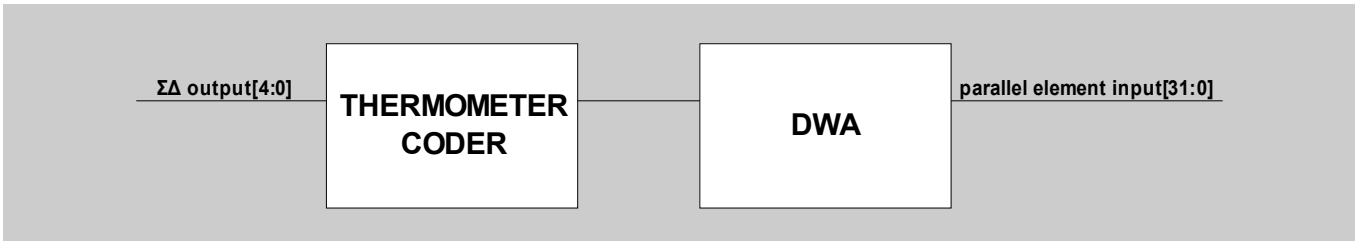

Figure 4.8: Topology of DWA block

The output of the DWA block is the input of each parallel element that will perform the analog conversion.

4.3 H/W Implementation

In this chapter there will be presented the basic principles followed in the implementation of the DWA block along with I/O description and desgn considerations.

4.3.1 Procedural diagram

Figure 4.9 shows the procedural diagram of the DWA block. The DWA algorithm is implemented without the use of internal subbllocks because of its small size.

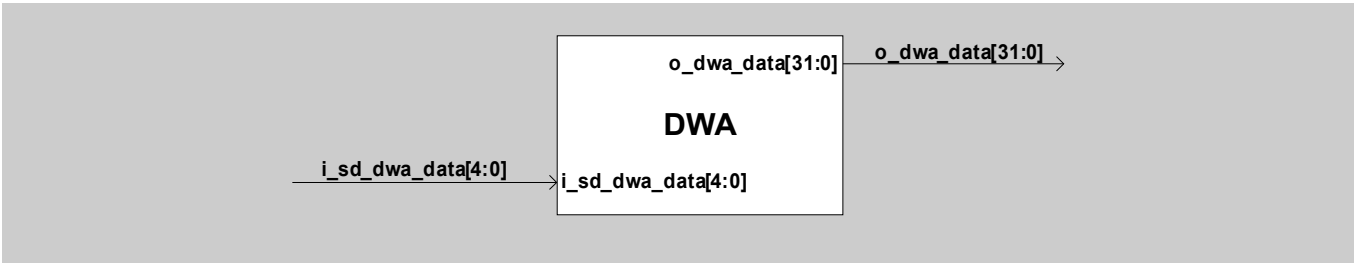

Figure 4.9: DWA Block Procedural Diagram

Figure 4.10 displays the DWA block interface diagram.

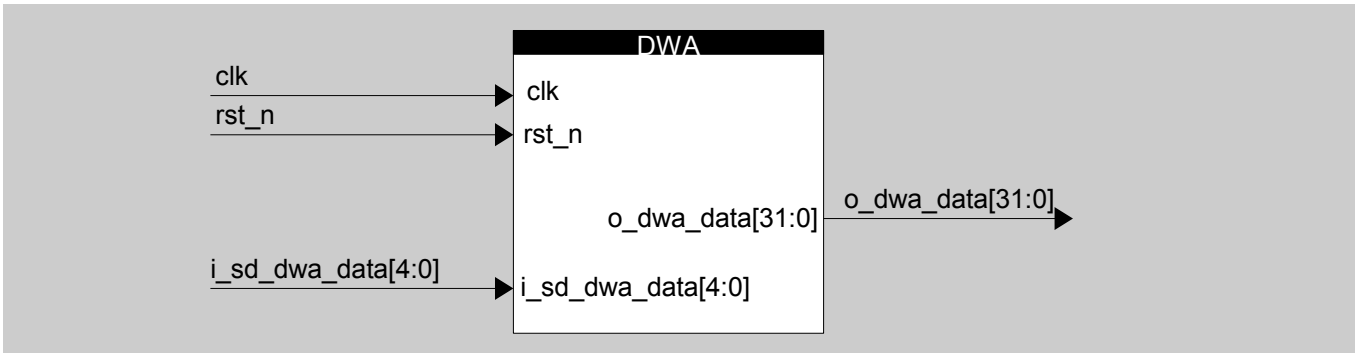

Figure 4.10: DWA Block Diagram

Table 4.5 lists the DWA block pin list.

Signal	I/O	Description
Clock & reset		
clk	in	Master Clock
rst_n	in	Active low reset synchronized from upper level
ΣΔ modulator interface		
i_sd_dwa_data[4:0]	in	5 bit input data from modulator
Analog Converter Interface		
o_dwa_data[31:0]	out	32 bit signal driven to the current sources of the analog converter.

Table 4.5: DWA Pin List

Description

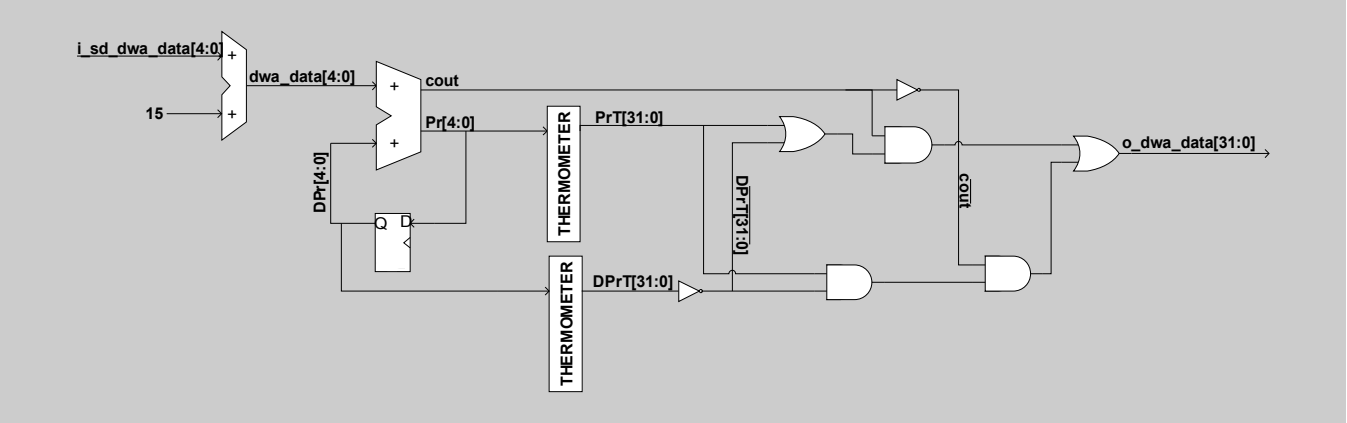

Figure 4.11: DWA Interface Diagram

The DWA block as explained in the theory part, translates an input sequence of 5 bits into a 32-bit thermometer code sequence indicating the position of the representation elements.

In order to implement the DWA block, we first have to modulate the input data to an unsigned signal because the thermometer coding does not support signed representation. Therefore we add 15 at each input and the DAC output range becomes 0 up to 32.

At this point it is important to mention that when an *rst_n* signal is received, the *dwa_data* signal becomes 15 since the zero value is represented as 15 in the new output range. The other elements of the DWA block are not affected by the reception of a reset because it will affect the output sequence.

The *dwa_data* signal is summed with the delayed pointer signal, *DPr*, and assigned to the pointer signal *Pr*. To receive the *DPr* we connect the *Pr* with a D flip-flop.

The summation is performed by a full adder which outputs the sum of the *dwa_data* and *DPr* as well as the last carry of the operation in the *cout* signal.

The *Pr* and *DPr* signals are then translated in thermometer code and assigned to the *DprT* and *PrT* respectively . After the pointer translation a logic equation is performed to calculate the final output of the DWA block. The equation is described above.

$$o\_dwa\_data = (PrT + \overline{DPrT}) \cdot cout + (PrT \cdot DPrT) \cdot \overline{cout} \tag{80}$$

4.4 Verification plan

As a result of the fact that the DWA block does not alter the output, instead it removes the mismatch error involved in the analog conversion, the verification process can be held by the following simple procedure:

A testbench is created which stores the DWA output into a file. The output of the DWA is stored in binary string format. A MATLAB script file loads the output file and transforms it to the original input of the DWA block (a quantized signal ranging from -15 to 15). Then the KDiff3 application is used to compare the MATLAB transformed output file with the original MATLAB output file of the ΣΔ modulator. If the files match the DWA block performs the proper calculations.

4.5 References

[1] Delta-Sigma Data Converters: Theory, Design and Simulation, Steven R. Norsworthy, Richard Schreier, Gabor C. Temes, First edition, 1996, p: 247-265

[2] *Data Converters*, Franko Maloberti, First edition, 2007, p: 374-391

[3] A Multibit Delta-Sigma Audio DAC with 120dB Dynamic Range, Ichiro Fujimori, Tetsuro Sugimoto, IEEE Journal of Solid-State Circuits, VOL.35 NO 8, August 2000.

[4] A 14-bit, 10-Msamples/s D/A Converter Using Multibit ΣΔ modulation, Katayoun Falakshahi, Chih-Kong Ken Yang, Bruce A. Wooley, IEEE Journal of Solid-State Circuits, VOL.34, NO 5 May 1999.

[5] Principles of Data Conversion Systems, Behzad Razavi, IEEE Press p. 45-63.

[6] High-Order Multibit Modulators and Pseudo Data-Weighted-Averaging in Low-Oversampling ΔΣ ADCs for Broad-Band Applications, Anas A. Hamoui, Kenneth W. Martin ,IEEE Transactions on Circuits and Systems-I: Regular Papers, vol 51 No 1, January 2004.

[7] Switching Sequence Optimization for Gradient Error Compensation in Thermometer-Decoded DAC Arrays.

[8] A Low Complexity Data Weighted Averaging (DWA) Algorithm Implementation, Ramon Lopez-Holloway, Miguel Garcia, Instituto National de Astrofisica Optica y Electronica, Luis Enrique Erro No.1, Puebla, Mexico.

5 Top level integration and Synthesis

5.1 Top Level Integration and results

The separate components of the DAC are connected inside a top level block. The sub-blocks composing the DAC are the Integrator, the ΣΔ modulator and the DWA block. Their functionality is described in the previous chapters. The interface diagram of the top level is shown in Figure 5.1:

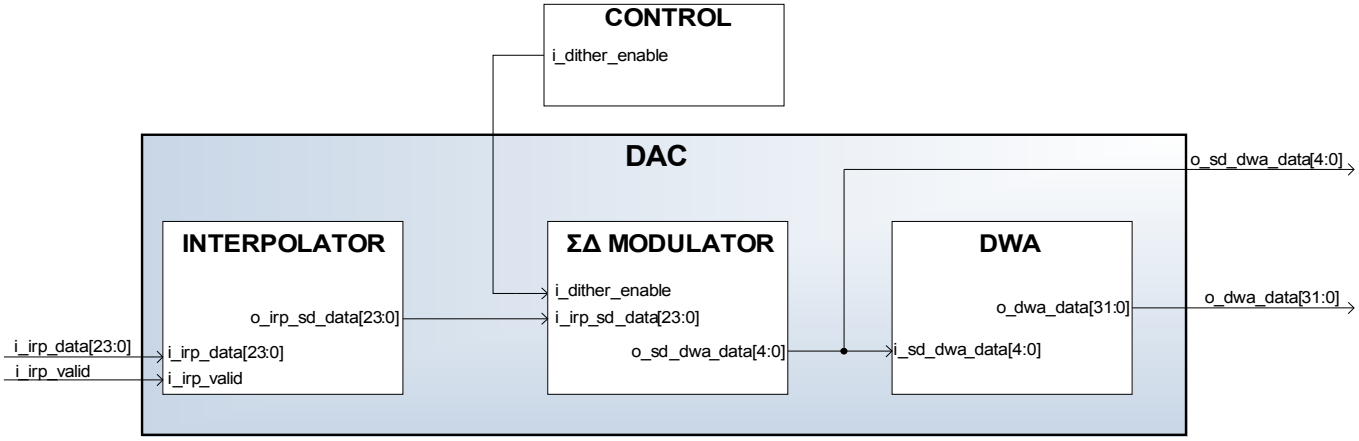

Figure 5.1: Top level DAC Interface Diagram

The outputs of the DAC are the 5-bit quantized output of the modulator and the 32-bit thermometer code output of the DWA block. The top level block includes in its output the modulators output in terms of easier simulation and verification.

The output of the DAC is given in Figure 5.2 below, verifying the expected performance. The input signal is a sinusoidal signal at 750Hz with an initial sampling rate of 6kHz.

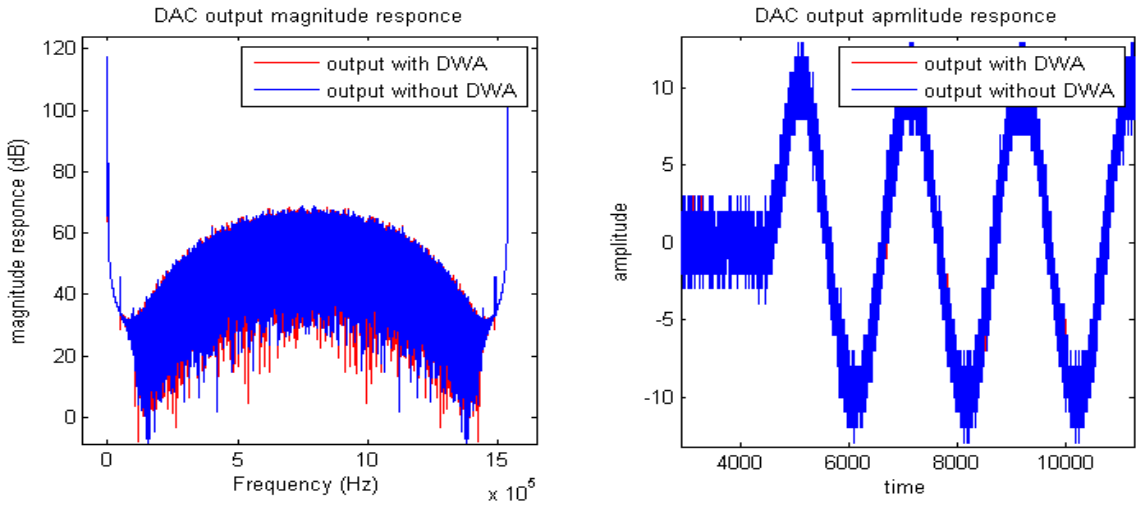

Figure 5.2: DAC amplitude and magnitude responce

As we can see, the noise of the signal is shaped at very high frequencies resulting in high SNR in the band of interest.

5.1.1 Verification Plan

The verification of the top level is made by comparing the RTL results with the MATLAB results. A test bench is created (dac_tb.vhd) which receives its input from a file already produced by a MATLAB script. The data are processed by the RTL and the test bench outputs two files: the ΣΔ quantized output of 5-bit length and the DAC output of 32-bit length using thermometer code arithmetic. These two files are then processed by a MATLAB script and compared with the KDiff3 application with the SIMULINK fixed point model outputs.

5.2 Synthesis results

The synthesis is performed in the Altera Quartus II environment. The synthesized file is the top level block of the converter. For the synthesis purpose, a clock oscillator is inserted in the top level design. The results of the synthesis can be summarized in the following metric table:

Results	Metrics
Total logic elements	2,846/18,752
Total combinational functions	2,357/18,752
Dedicated logic registers	1,702/18,752
Total pins	65/315
Total memory bits	0/239,616
Embedded Multiplier 9-bit element	21/52
Total PLLs	0/4

Table 5.1: Synthesis results summary

6 Conclusion

The design and implementation of the ΣΔ DAC, was a great challenge due to the large number of considerations that had to be taken. Several steps where made until the completion of this thesis. Some of them where the study of the theory involved in the ΣΔ architecture, the design of theoretical models in SIMULINK/MATLAB, the implementation in VHDL of the theoretical model, the verification of the RTL functionality and its synthesis.

The output of the RTL coincides with the output of the SIMULINK resulting the expected performance. The implementation of the converter was made according to the initial specifications. Although this is a very good approach of the converter, several modifications could be made. Some of them are described in the following paragraphs.

The interpolator is designed so the output signal would have a passband range of 1KHz. The filters inside the interpolator could be scaled by changing the sampling rate of the initial signal. Also, the coefficients of the filters in the current implementation are hard wired resulting in bigger area inside the IC. A memory or a register file could be used instead, to store the filters coefficients.

As far as the ΣΔ modulator is concerned, other implementations could be performed by changing the architecture of the modulator (MASH, different approach of the error feedback loop) and keeping the same characteristics.

Finally the internal DAC topology could be different by choosing the binary code or other approaches. As for the thermometer code topology, the Dynamic Element Matching algorithm is an option with many different implementations to choose.

Finally the synthesis results are satisfactory according to the percentage of the used sources of the FPGA.

This implementation could also be extended furthermore. The most significant extension that could be made, is the connection of the FPGA with an analog part. The analog part could be composed by a set of current sources and an analog filter.

7 Glossary

D

DAC: Digital to Analog Converter

DEM: Dynamic Element Matching

DWA: Data Weighted Averaging

DNL: Differential Nonlinearity

F

FIR: Finite Impulse Response

FPGA: Field Programmable Gate Array

FSM: Finite State Machine

H

HB: Half Band

I

IIR: Infinite Impulse Response

ILA: Individual Level Averaging

INL: Integral Nonlinearity

L

LPF: Low Pass Filter

LSB: Less Significant Bit

LFSR: Linear Feedback Shift Register

M

MASH: Multi-stage noise-shaping

MSB: Most Significant Bit

O

OSR: Oversampling Ratio

R

RTL: Register Transfer Language

S

ΣΔ: ΣΔ

SINC: Sinus Cardinal

SNR: Signal to Noise Ratio

8 Appendix A

In this appendix, the content of each source code file is mentioned

File Name	Description
Interpolator	
interpolator.vhd	Top level interpolator file
irp1.vhd	First FIR equiripple filter (IRP1)
irp1_control.vhd	FSM of IRP1
irp1_accumulator.vhd	Accumulator of IRP1
irp2.vhd	First Half Band filter (IRP2)
irp2_control.vhd	FSM of IRP2
irp2_accumulator.vhd	Accumulator of IRP2
irp3.vhd	Second Half Band filter (IRP3)
irp3_control.vhd	FSM of IRP3
irp3_accumulator.vhd	Accumulator of IRP3
sinc.vhd	SINC filter
dff_sum_delay.vhd	Delay element of SINC filter
dff_irp.vhd	D flip-flop of Interpolator
irp_tb.vhd	Test bench of Interpolator
ΣΔ Modulator	
sd_mod.vhd	Top level ΣΔ modulator file
integrator.vhd	Integrator block file
delayed_integrator.vhd	Delayed integrator block file
substractor.vhd	Substractor block file
dither.vhd	Dither block
adder.vhd	Adder block
dff_sd.vhd	D flip-flop of ΣΔ modulator
sdmod_tb	ΣΔ modulator test bench
DWA	
dwa.vhd	DWA top level file
dff_dwa.vhd	D flip-flop of DWA block
adder_dwa.vhd	Adder of DWA block
dwa_tb.vhd	DWA block test bench
ΣΔ DAC	
dac.vhd	Top level ΣΔ DAC file
dac_tb.vhd	Top level test bench (outputs both quantized output of ΣΔ modulator and thermometer code output of DWA block)

9 Appendix B

The design and implementation of the ΣΔ DAC is performed by using the following applications:

- Windows XP Professional SP2 Operating System
- MATLAB/SIMULINK 7.1
- Modelsim SE 6.0
- Altera Quartus II sp1 Web Edition
- KDiff3
- OpenOffice.org Writer 2.4.0

10 Appendix C

The coefficients of the filters are given in the tables above:

IRP1			
coeff0	1173	coeff12	-1094025
coeff1	5824	coeff13	877839
coeff2	11125	coeff14	3328842
coeff3	611	coeff15	3205748
coeff4	-43281	coeff16	-1405278
coeff5	-92752	coeff17	-7832448
coeff6	-48522	coeff18	-8889393
coeff7	166304	coeff19	1847042
coeff8	411370	coeff20	23269264
coeff9	293593	coeff21	45105411
coeff10	-438312	coeff22	54350989
coeff11	-1298726		

IRP2			
coeff0	581	coeff10	617383
coeff2	5320	coeff12	-1465185
coeff4	-26186	coeff14	5226797
coeff6	91371	coeff15	8388608
coeff8	-254616		

IRP3			
coeff0	6357	coeff6	-1172842
coeff2	-63943	coeff8	5101847
coeff4	322885	coeff9	8388608