The Dissertation Committee for Waqas Akram certifies that this is the approved version of the following dissertation:

**Tunable Mismatch Shaping**
for
**Bandpass Delta-Sigma Data Converters**

Committee:

Earl E. Swartzlander, Jr., Supervisor

Mircea D. Driga

Michael E. Orshansky

Vivek Telang

Nur A. Touba
Tunable Mismatch Shaping
for
Bandpass Delta-Sigma Data Converters

by

Waqas Akram, B.S.E.E.; B.A.; M.S.E.

DISSERTATION
Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN
May 2011
Dedicated to the memory of my loving parents, Zakia and Muhammad Akram. My mother was a mathematician and my father an engineer. Together, they ignited my passion to explore and encouraged my tenacity to succeed, but above all else, they nurtured me with patience and showed me how to persevere.
Acknowledgments

My deepest gratitude to my adviser, Dr. Earl E. Swartzlander, Jr., for his limitless patience and the thoughtful guidance he has generously provided, as a mentor and as a friend, throughout the course of my graduate studies.

My sincere thanks to all the members of my dissertation committee for their support and kindness in sharing their precious time.

I remain truly indebted to Melanie Gulick for always being willing to help; no matter how busy the moment, or how vexing the issue, her cheerfulness never failed to brighten the day.

Most importantly, none of this would be possible without the encouragement and support of my family. Their love and patience has sustained me throughout my life and brought me here today.
Oversampled digital-to-analog converters typically employ an array of unit elements to drive out the analog signal. Manufacturing defects can create errors due to mismatch between the unit elements, leading to a sharp reduction in the effective dynamic range through the converter. Mismatch noise shaping is an established technique for alleviating these effects, but usually anchors the signal band to a fixed frequency location. In order to extend these advantages to tunable applications, this work explores a series of techniques that allow the suppression band of the mismatch noise shaping function to have an adjustable center frequency. The proposed techniques are implemented in hardware and evaluated according to mismatch shaping performance, latency and hardware complexity.
# Table of Contents

Acknowledgments v  
Abstract vi  
List of Tables x  
List of Figures xi  

## Chapter 1. Introduction  
1  

## Chapter 2. Background and Related Work  
4  
2.1 Overview of ΔΣ Modulators  
2.1.1 Noise Shaping  
2.1.2 Lowpass and Bandpass Loop Filters  
2.1.3 ΔΣ Modulator as ADC and DAC  
2.1.4 Single-bit and Multi-bit Quantization  
2.2 Mismatch Noise Shaping  
2.2.1 Dynamic Element Matching  
2.2.2 Averaging Techniques  
2.2.3 Vector-based Mismatch Shaping  
2.2.4 Tree-based Mismatch Shaping  
2.2.5 N-Path Mismatch Shaping  
2.2.6 Quadrature Mismatch Shaping  

## Chapter 3. Tunable Vector Mismatch Shaping  
19  
3.1 Overview of the Vector Shaper  
3.2 Tunable Implementation  
3.2.1 Tunable Loop Filter  
3.2.2 Vector Quantizer
3.3 Performance and Complexity .................................. 30
3.4 Summary ......................................................... 38

Chapter 4. Tunable Tree Mismatch Shaping 40
4.1 Overview of the Tree Structure ............................... 41
4.2 Tunable Implementation ......................................... 46
  4.2.1 Retiming the Loop Filters ............................... 48
  4.2.2 Designing the Tunable Noise-Transfer Function ....... 52
  4.2.3 CRFB Architectures for 1st- and 2nd-order Shaping ... 53
    4.2.3.1 Tunable 1st-order Shaping ....................... 54
    4.2.3.2 Tunable 2nd-order Shaping ....................... 57
  4.2.4 Polar Architecture for 1st-order Shaping ............... 60
4.3 Comparison of Tunable Tree Architectures .................. 64
4.4 Comparison with Tunable Vector Structure .................. 71
4.5 Summary ......................................................... 77

Chapter 5. Tunable $N$-Path Mismatch Shaping 79
5.1 Introduction .................................................... 80
5.2 Tunable $N$-path Technique ................................... 82
  5.2.1 $N$-path Filter Principle ................................. 82
  5.2.2 Modified $N$-path for Tunable Operation ............... 84
  5.2.3 Prototype Diversity ...................................... 87
  5.2.4 Proposed Tunable Structure ............................. 89
5.3 Prototype Implementation ..................................... 94
  5.3.1 DWA Prototype ............................................ 96
  5.3.2 Vector Prototype ......................................... 100
  5.3.3 Tree Prototype ........................................... 104
5.4 Comparison of Tunable $N$-path Architectures ............. 109
  5.4.1 Shaping Performance .................................... 109
  5.4.2 Hardware Complexity .................................... 114
5.5 Comparison with Tunable Tree Structure ..................... 116
  5.5.1 Shaping Performance .................................... 116
  5.5.2 Hardware Complexity .................................... 121
5.6 Summary ......................................................... 125

viii
## Chapter 6. Tunable Quadrature Mismatch Shaping

6.1 Introduction .................................................. 128  
6.2 Overview of Mismatch Shaping ................................. 131  
  6.2.1 Quadrature Path Mismatch .............................. 133  
  6.2.2 Quadrature Mismatch Shaping ......................... 134  
  6.2.3 Tunable Mismatch Shaping ........................... 135  
6.3 Proposed Tunable Technique ................................ 136  
6.4 Hardware Implementation .................................. 139  
  6.4.1 Tunable Complex Loop Filter .......................... 140  
  6.4.2 Path Quantizer ........................................ 142  
  6.4.3 Complex Limiter ....................................... 143  
  6.4.4 Simulation of Tunable Operation ...................... 147  
6.5 Comparison of Shaping Performance ......................... 150  
6.6 Comparison of Hardware Complexity ......................... 155  
6.7 Summary .................................................... 158  

## Chapter 7. Conclusion ........................................ 160  
7.1 Tunable Mismatch Shaping for Real Signals ............... 161  
7.2 Tunable Mismatch Shaping for Complex Signals .......... 162  

## Appendix ....................................................... 164  

## Appendix 1. Mismatch Shaping Performance .................. 165  
  1.1 Comparison of Tunable Real Mismatch Shapers .......... 165  
  1.2 Comparison of Tunable Quadrature Mismatch Shapers .... 173  

## Bibliography .................................................. 181  

## Vita ............................................................ 191
List of Tables

3.1 Coefficient and datapath widths for loop filter ............... 28
3.2 Tunable bandpass $\Delta \Sigma$ modulator specifications .......... 30
3.3 Hardware synthesis conditions .................................. 36

4.1 Coefficient and datapath widths for CRFB2 structure ...... 57
4.2 Coefficient and datapath widths for CRFB4 structure ...... 60
4.3 Coefficient and datapath widths for polar structure ....... 63
4.4 Tunable bandpass $\Delta \Sigma$ modulator specifications .......... 64
4.5 Hardware synthesis conditions .................................. 69

5.1 Tunable bandpass $\Delta \Sigma$ modulator specifications .......... 109
5.2 Hardware synthesis conditions .................................. 114

6.1 Coefficient and datapath widths for quadrature structure ... 141
6.2 Limiter table for the lowest layer of the tree ................ 145
6.3 Limiter table for all upper layers of the tree ............... 146
6.4 Quadrature BP $\Delta \Sigma$ modulator specifications .......... 150
6.5 Hardware synthesis conditions .................................. 156

7.1 Mismatch shaper decision matrix .............................. 162

1.1 Index of performance figures for real-valued shapers ........ 165
1.2 Index of performance figures for complex-valued shapers ... 173
List of Figures

2.1 Structure of ∆Σ modulator ................................. 6
2.2 ∆Σ modulator spectra ..................................... 8
2.3 Structures of ∆Σ data converters .......................... 9
2.4 Single and multi-bit Quantizers ............................ 10
2.5 Structure of unit-element DAC ............................. 11
2.6 Mismatch shaping ∆Σ data converters ................... 12
2.7 Conceptual implementation of $N$-path transform ........ 17

3.1 Unit-element DAC with element selection logic .......... 20
3.2 Structure of the vector mismatch shaper .................. 22
3.3 The re-arranged vector mismatch shaper ................. 25
3.4 Tunable loop filter using 2nd-order structure ............ 27
3.5 Tuning coefficients used in the 2nd-order structure ...... 28
3.6 Structure of the vector quantizer .......................... 29
3.7 Sorting network using an 8-input merging network ....... 31
3.8 Structure of a single comparator .......................... 32
3.9 SNR with 3% mismatch for a 2-element vector shaper .... 33
3.10 SNR with 3% mismatch for a 4-element vector shaper .... 33
3.11 SNR with 3% mismatch for a 8-element vector shaper .... 34
3.12 SNR with 3% mismatch for a 16-element vector shaper ... 34
3.13 SNR with 3% mismatch for a 32-element vector shaper ... 35
3.14 SNR with 3% mismatch for a 64-element vector shaper ... 35
3.15 SNR with 3% mismatch for a 128-element vector shaper .. 36
3.16 Critical path delay of vector shaper ...................... 37
3.17 Hardware complexity of vector shaper .................. 38

4.1 Unit-element DAC with element selection logic .......... 42
4.2 Tree-structured element selection logic ................... 43
4.3 Switching block for tree structure ........................................ 44
4.4 Switching sequence generator .............................................. 45
4.5 Types of quantizers used within the sequence generator .......... 46
4.6 Tunable tree structure ..................................................... 47
4.7 Retimed versions of the sequence generator ......................... 49
4.8 Critical path using a single retiming method ......................... 50
4.9 Critical path using more than two retiming methods .............. 51
4.10 Critical path using only two retiming methods .................... 51
4.11 Tunable loop filter using 2nd-order CRFB structure ............... 55
4.12 Tuning coefficients used in the 2nd-order CRFB structure ....... 56
4.13 Tunable loop filter using 4th-order CRFB structure ............... 58
4.14 Tuning coefficients used in the 4th-order CRFB structure ....... 59
4.15 Tunable loop filter using 2nd-order Polar structure .............. 61
4.16 Tuning coefficients used in the 2nd-order polar structure ...... 63
4.17 SNR with 3% mismatch for 2-element tree shapers ............... 65
4.18 SNR with 3% mismatch for 4-element tree shapers ............... 66
4.19 SNR with 3% mismatch for 8-element tree shapers ............... 66
4.20 SNR with 3% mismatch for 16-element tree shapers ............. 67
4.21 SNR with 3% mismatch for 32-element tree shapers ............. 67
4.22 SNR with 3% mismatch for 64-element tree shapers ............. 68
4.23 SNR with 3% mismatch for 128-element tree shapers .......... 68
4.24 Comparison of tree critical path delays ......................... 70
4.25 Comparison of tree hardware complexity ....................... 70
4.26 SNR with 3% mismatch for 2-element vector & tree shapers ... 72
4.27 SNR with 3% mismatch for 4-element vector & tree shapers ... 72
4.28 SNR with 3% mismatch for 8-element vector & tree shapers ... 73
4.29 SNR with 3% mismatch for 16-element vector & tree shapers .. 73
4.30 SNR with 3% mismatch for 32-element vector & tree shapers .. 74
4.31 SNR with 3% mismatch for 64-element vector & tree shapers .. 74
4.32 SNR with 3% mismatch for 128-element vector & tree shapers 75
4.33 Comparison of tree and vector critical path delays ............ 76
4.34 Comparison of tree and vector hardware complexity ........... 76
5.1 Commutator model of the $N$-path transform $z \rightarrow z^N$ . . . . . . . 83
5.2 $N$-path pole-zero locations for (a) $N = 4$ and (b) $N = 8$ . . . 86
5.3 Mismatch error-shaping when (a) $N = 4$ and (b) $N = 8$ . . . . 86
5.4 3-path pole-zero plots for (a) lowpass and (b) highpass . . . . . 88
5.5 Mismatch shaping response at various tuning frequencies . . . . 90
5.6 Structure of tunable $N$-path mismatch shaper . . . . . . . . . . 91
5.7 $N$-path tuning pairs . . . . . . . . . . . . . . . . . . . . . . . . . . 95
5.8 High-level view of DWA prototype shaper . . . . . . . . . . . . 96
5.9 Shifting binary to thermometer converter . . . . . . . . . . . . 97
5.10 Circular barrel shifter . . . . . . . . . . . . . . . . . . . . . . . . . . 97
5.11 highpass (a) response and (b) rotation example . . . . . . . . . 99
5.12 lowpass (a) response and (b) rotation example . . . . . . . . . 99
5.13 1st-order hybrid pointer update structure . . . . . . . . . . . . 100
5.14 Structure of the vector mismatch shaper . . . . . . . . . . . . 101
5.15 $N$-path vector shaper prototype . . . . . . . . . . . . . . . . . . 102
5.16 Loop filter for $N$-path vector prototype . . . . . . . . . . . . 103
5.17 Tree-structured element selection logic . . . . . . . . . . . . . 104
5.18 Tunable $N$-path tree structure . . . . . . . . . . . . . . . . . . . . 106
5.19 Structure of the tunable $N$-path sequence generator . . . . . . 107
5.20 Sequence generator detail for $N$-path tree prototype . . . . . 108
5.21 SNR with 3% mismatch for 2-element $N$-path shapers . . . . . 110
5.22 SNR with 3% mismatch for 4-element $N$-path shapers . . . . . 111
5.23 SNR with 3% mismatch for 8-element $N$-path shapers . . . . . 111
5.24 SNR with 3% mismatch for 16-element $N$-path shapers . . . . 112
5.25 SNR with 3% mismatch for 32-element $N$-path shapers . . . . 112
5.26 SNR with 3% mismatch for 64-element $N$-path shapers . . . . 113
5.27 SNR with 3% mismatch for 128-element $N$-path shapers . . . 113
5.28 Comparison of $N$-path critical path delays . . . . . . . . . . . 115
5.29 Comparison of $N$-path hardware complexity . . . . . . . . . . . 115
5.30 SNR with 3% mismatch for 2-element $N$-path & tree shapers . 117
5.31 SNR with 3% mismatch for 4-element $N$-path & tree shapers . 118
5.32 SNR with 3% mismatch for 8-element $N$-path & tree shapers . 118
5.33 SNR with 3% mismatch for 16-element N-path & tree shapers 119
5.34 SNR with 3% mismatch for 32-element N-path & tree shapers 119
5.35 SNR with 3% mismatch for 64-element N-path & tree shapers 120
5.36 SNR with 3% mismatch for 128-element N-path & tree shapers 120
5.37 Comparison of N-path and tree critical path delays ............ 121
5.38 Comparison of N-path and tree hardware complexity ........... 122
5.39 Comparison of combinational logic area ......................... 124
5.40 Comparison of non-combinational logic area ..................... 124

6.1 Low-IF bandpass ΔΣ receiver architecture ........................ 129
6.2 Low-IF bandpass ΔΣ transmitter architecture .................... 130
6.3 Unit-element DAC with element selection logic .................. 132
6.4 QBP-ΔΣ ADC with complex DAC ................................. 134
6.5 Tree-structured element selection logic ........................... 136
6.6 Switching block for complex data ............................... 138
6.7 Complex switching sequence generator ............................ 139
6.8 Tunable 1st-order complex loop filter ............................ 140
6.9 Coefficient values plotted over the tuning range ................ 142
6.10 Full spectrum with 3% mismatch error ........................... 148
6.11 Signal band detail with 3% mismatch error ...................... 149
6.12 SNR with 3% mismatch for 2-element quadrature shapers ....... 152
6.13 SNR with 3% mismatch for 4-element quadrature shapers ...... 152
6.14 SNR with 3% mismatch for 8-element quadrature shapers ...... 153
6.15 SNR with 3% mismatch for 16-element quadrature shapers ..... 153
6.16 SNR with 3% mismatch for 32-element quadrature shapers .... 154
6.17 SNR with 3% mismatch for 64-element quadrature shapers ..... 154
6.18 SNR with 3% mismatch for 128-element quadrature shapers ... 155
6.19 Comparison of complex tree critical path delays ............... 157
6.20 Comparison of complex tree hardware complexity .............. 157

1.1 SNR with 1% mismatch for 2-element N-path & tree shapers . 166
1.2 SNR with 2% mismatch for 2-element N-path & tree shapers . 166
1.3 SNR with 1% mismatch for 4-element N-path & tree shapers . 167
1.4 SNR with 2% mismatch for 4-element N-path & tree shapers . 167
1.5 SNR with 1% mismatch for 8-element N-path & tree shapers . 168
1.6 SNR with 2% mismatch for 8-element N-path & tree shapers . 168
1.7 SNR with 1% mismatch for 16-element N-path & tree shapers 169
1.8 SNR with 2% mismatch for 16-element N-path & tree shapers 169
1.9 SNR with 1% mismatch for 32-element N-path & tree shapers 170
1.10 SNR with 2% mismatch for 32-element N-path & tree shapers 170
1.11 SNR with 1% mismatch for 64-element N-path & tree shapers 171
1.12 SNR with 2% mismatch for 64-element N-path & tree shapers 171
1.13 SNR with 1% mismatch for 128-element N-path & tree shapers 172
1.14 SNR with 2% mismatch for 128-element N-path & tree shapers 172
1.15 SNR with 1% mismatch for 2-element quadrature shapers . . . 174
1.16 SNR with 2% mismatch for 2-element quadrature shapers . . . 174
1.17 SNR with 1% mismatch for 4-element quadrature shapers . . . 175
1.18 SNR with 2% mismatch for 4-element quadrature shapers . . . 175
1.19 SNR with 1% mismatch for 8-element quadrature shapers . . . 176
1.20 SNR with 2% mismatch for 8-element quadrature shapers . . . 176
1.21 SNR with 1% mismatch for 16-element quadrature shapers . . . 177
1.22 SNR with 2% mismatch for 16-element quadrature shapers . . . 177
1.23 SNR with 1% mismatch for 32-element quadrature shapers . . . 178
1.24 SNR with 2% mismatch for 32-element quadrature shapers . . . 178
1.25 SNR with 1% mismatch for 64-element quadrature shapers . . . 179
1.26 SNR with 2% mismatch for 64-element quadrature shapers . . . 179
1.27 SNR with 1% mismatch for 128-element quadrature shapers . . . 180
1.28 SNR with 2% mismatch for 128-element quadrature shapers . . . 180
Chapter 1

Introduction

Delta-Sigma (ΔΣ) modulation is a common technique for data conversion in applications where high linearity is required over a narrow bandwidth. Both Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) can be designed with this technique. In a ΔΣ ADC, oversampling is used to convert a band-limited continuous-amplitude signal to a coarsely-quantized signal at a sampling rate much higher than the Nyquist rate. In a ΔΣ DAC, oversampling is used to convert a finely quantized signal at the Nyquist rate to a coarsely-quantized signal at a much higher sampling rate. These types of data converters preserve a high dynamic range within the narrow signal band by employing a noise-shaping loop filter, which strives to push most of the quantization noise out of the signal band [1][2]. A bandpass ΔΣ data converter has a noise-shaping loop filter that operates with a bandpass signal band [3][4][5].

As process technology allows ΔΣ data converters to operate at increasingly higher sample rates, more of the analog functionality is shifting into the digital domain [6][7][8]. An important feature for communications systems is the ability to tune the intermediate-frequency (IF) to arbitrary fre-
quencies in the Nyquist range [9]. Using a bandpass $\Delta \Sigma$ modulator with a fixed signal-band location would require a wider bandwidth that encompasses all anticipated IF settings [10][11][12]. However, using a tunable bandpass $\Delta \Sigma$ modulator would increase the allowable over-sampling ratio (OSR), or reduce the required sample rate for the same OSR, and inevitably improve the performance-to-power ratio of the design. The use of a tunable noise-shaping loop filter allows the bandpass signal-band to be centered at arbitrary frequencies in the Nyquist range.

The linearity of the internal coarse quantizer within a $\Delta \Sigma$ modulator sets the linearity of the overall data converter [1][2]. A single-bit quantizer is inherently linear, but places limitations on the achievable signal-to-noise ratio (SNR). Multibit quantizers allow higher achievable SNR, but semiconductor manufacturing defects can lead to severe non-linearities through the quantizer, and thereby limit the overall converter SNR. A common type of coarse DAC used within a $\Delta \Sigma$ data converter employs an array of unit elements, such as capacitors or current sources, to produce the analog output. In such a DAC, manufacturing defects manifest themselves as deviations of unit-element levels from their ideal values, known as mismatch error. The spectral noise due to mismatch errors, known as mismatch noise, can sharply reduce the achievable SNR. In a unit-element DAC, mismatch noise-shaping is an established technique for spectrally shaping the mismatch noise power away from the signal band [13][14][15]. This is done by dynamically re-arranging the order in which the converter’s unit-elements are utilized. Prior works have anchored
the signal band to a fixed frequency location. This work proposes a series of mismatch shaping techniques that allow the center frequency of the mismatch suppression band to be centered at arbitrary frequencies within the Nyquist band [16].

Chapter 2 provides background information on \(\Delta \Sigma\) data converters, along with a quick review of existing mismatch shaping techniques. Chapter 3 presents the vector-based mismatch shaper, and proposes an extension to this technique in order to allow tunable operation. Chapter 4 presents the tree-structured mismatch shaper, and proposes several architectures that enable tunable operation. Chapter 5 proposes the tunable \(N\)-path technique, and describes several architectures for implementing this technique. Chapter 6 proposes an extension to the tunable tree-structured mismatch shaper from Chapter 4 that allows the processing of complex-valued signals and extends the benefits of tunable mismatch shaping to quadrature bandpass \(\Delta \Sigma\) data converters [17]. The work is finally concluded in Chapter 7.
Chapter 2

Background and Related Work

The concept of software-defined radio is now a reality as radio front-ends become more adept at managing multiple wireless standards. Different wireless standards can have widely divergent modulation schemes and signal bandwidths, operating at various intermediate frequencies (IF). As a result, direct-digital synthesis of the IF signal is becoming increasingly attractive [6][7][8]. The use of bandpass ΔΣ modulators for data conversion has become ubiquitous in these applications due to their high linearity over a narrow signal bandwidth, as compared to their Nyquist-rate counterparts [3][4][12].

Within a ΔΣ modulator loop, the performance of the quantizer places an upper bound on the achievable signal linearity through the entire converter. Single-bit quantization is an inherently linear process, whereas multi-bit quantization can suffer from severe non-linearities due to manufacturing defects. However, multi-bit quantization increases the achievable dynamic range through the data converter. As a result, many attempts have been made at alleviating or eliminating the effects of mismatch error within the signal band [13][14][15]. These techniques include trimming, calibration, digital correction, and mismatch shaping [2].
This chapter describes existing efforts to linearize a unit-element DAC using mismatch noise shaping. Section 2.1 first provides a brief description of data converters employing ΔΣ modulation, including bandpass ΔΣ converters using multi-bit quantization. Section 2.2 introduces mismatch shaping techniques for mitigating the effects of DAC element mismatch, including lowpass and bandpass shaping.

2.1 Overview of ΔΣ Modulators

Delta-Sigma data converters are widely used in applications where high-linearity is required over narrow bandwidth [1][2]. A higher signal-to-noise ratio (SNR) is achieved at the expense of a much higher sampling rate and increased out-of-band quantization noise at the output. This noise-shaped characteristic directly follow from the noise-transfer function (NTF) of the ΔΣ modulator.

2.1.1 Noise Shaping

Figure 2.1(a) shows the basic structure of a ΔΣ modulator. The input is \( U(z) \), the output is \( V(z) \), and the loop filter transfer function is denoted by \( L(z) \). Due to the inherently nonlinear operation of the quantizer, the model must first be linearized in order to analyze it using classical linear techniques. A linearized \( z \)-domain model is shown in Figure 2.1(b). The modulator output \( V(z) \) consists of a signal component \( U(z) \) and an error component \( E_q(z) \), and
is given by

\[ V(z) = STF(z)U(z) + NTF(z)E_q(z) \]  

(2.1)

where \( STF(z) \) denotes the \textit{signal transfer function} and \( NTF(z) \) denotes the \textit{noise transfer function}. These are each given by:

\[ STF(z) = \frac{L(z)}{1 + L(z)} \]  

(2.2)

\[ NTF(z) = \frac{1}{1 + L(z)} \]  

(2.3)

These expressions show that a loop filter \( L(z) \) with a high gain in the signal band and low gain elsewhere would greatly suppress the quantization noise within the signal band, while leaving the signal virtually unchanged. The quantization noise would now mostly lie outside the signal band and can
easily be filtered using the appropriate lowpass or bandpass filter. Due to this noise-shaping process, it is beneficial to maximize the sampling rate of the data converter to allow the loop filter more freedom to be aggressive. The ratio of the converter sampling rate to the actual signal bandwidth is quantified by the *oversampling ratio* (OSR) [1].

### 2.1.2 Lowpass and Bandpass Loop Filters

In the linearized model depicted in Figure 2.1(b), the quantizer is shown as a source of additive error. This quantizer is generally very coarse, and generates a large quantization error. However, the high OSR of the converter affords a relatively large bandwidth in which the quantization noise power can be shaped. As described in Section 2.1.1, noise shaping is performed by the loop filter $L(z)$, and the width of the loop filter pass-band determines the allowable signal bandwidth [1]. For example, Figure 2.2(a) shows the response from a $\Delta\Sigma$ modulator using a lowpass loop filter $L(z)$ producing a highpass noise-transfer function $NTF(z)$.

The noise-shaping behavior of a $\Delta\Sigma$ modulator can be generalized to handle a signal band located anywhere within the Nyquist band, and a bandpass $\Delta\Sigma$ data converter finds common use in many wireless systems. For example, a loop filter $L(z)$ with a bandpass response will create a $STF$ with a bandpass response and a $NTF$ with a bandstop response. Figure 2.2(b) shows the response of a bandpass $\Delta\Sigma$ modulator with the signal band centered at $F_S/4$, where $F_S$ is the sampling rate.
2.1.3 ΔΣ Modulator as ADC and DAC

A ΔΣ modulator can be used either as an analog-to-digital converter (ADC) or a digital-to-analog converter (DAC), as shown in Figure 2.3. In a ΔΣ ADC, the quantizer is itself an ADC, but with a far lower precision than the overall ADC. A correspondingly coarse DAC is required within the loop in order to provide feedback, as shown in Figure 2.3(a). In a ΔΣ DAC, the entire loop remains in the digital domain, including the coarse quantizer. However, the quantized signal still needs to be converted to an analog signal and thus requires a coarse DAC at the output, as shown in Figure 2.3(b).

2.1.4 Single-bit and Multi-bit Quantization

In both a ΔΣ ADC and a ΔΣ DAC, the linearity of the embedded coarse DAC limits the overall linearity of the ΔΣ converter. Manufacturing
Defects lead to deviations of individual quantizer levels from their ideal values. In a single-bit quantizer, there are only two quantization levels, as shown in Figure 2.4(a). In this case, the DAC output will always be linear, irrespective of gain and offset errors. In a multi-bit quantizer, there are multiple quantization levels, as shown in Figure 2.4(b). Mismatch in the quantizer levels leads to poor linearity in the DAC output, irrespective of gain and offset errors [1].

There are many benefits from using multi-bit quantization over single-bit quantization within a $\Delta\Sigma$ modulator loop. These benefits include better modulator loop stability and greater flexibility in selecting the aggressiveness of the loop filter, which allows for lower in-band quantization noise. In a $\Delta\Sigma$ ADC, benefits can also include relaxed requirements on the slew-rate and
Figure 2.4: Single and multi-bit Quantizers

linearity of the analog input stages of the loop filter [1]. These advantages all translate to a higher achievable SNR at a given OSR.

As a result of these advantages, there have been many attempts to reduce the in-band nonlinearity caused by mismatch errors when using a multi-bit quantizer. These include trimming, calibration, digital correction, and mismatch shaping [1][2]. Trimming techniques are generally very expensive and can suffer from long-term drift of circuit parameters. Calibration and digital correction schemes require some form of error measurement and acquisition techniques before these can be corrected or canceled in the digital domain, eventually leading to complex and expensive systems. However, mismatch shaping offers the most promising approach for alleviating the effects of mismatch errors. The next section provides brief descriptions of existing mismatch shaping techniques.
2.2 Mismatch Noise Shaping

In ΔΣ data converters, the internal DACs are typically constructed using unit-sized circuit elements, such as capacitors and current sources. Each DAC element is driven by a single control bit which represents a single least-significant bit (LSB) of the digital data. An example of an 8-bit unit-element DAC is shown in Figure 2.5, where the unit-element control bits are generated from element-selection logic. In a conventional unit-element DAC, the element selection logic simply converts the binary data to a thermometer-encoded vector, where each bit is dedicated to controlling a single DAC unit-element.

In a unit-element DAC, mismatch noise-shaping is an established technique for spectrally shaping the mismatch noise power away from the signal band [13][14][15]. This is done by modifying the element selection logic to dynamically re-arrange the order in which the unit-elements are utilized. When
Figure 2.6: Mismatch shaping $\Delta \Sigma$ data converters

this is done in a certain way, the noise power created by mismatch errors can be suppressed within the signal band. These mismatch shaping techniques are used for the embedded DACs in both $\Delta \Sigma$ ADC and $\Delta \Sigma$ DAC, as shown in Figure 2.6.

2.2.1 Dynamic Element Matching

Dynamic Element Matching (DEM) is a simple technique for reducing tonal and signal-dependent behavior in the mismatch noise spectrum [18][19]. In a DAC employing DEM, the element usage pattern is randomized so as to break up signal-dependent tones. This whitens the mismatch noise power and spreads it over the entire spectrum. However, DEM has the undesirable
effect of raising the noise floor, and thereby reducing the achievable SNR in the band of interest. Implementation is relatively straight-forward with the use of a thermometer encoder, a pseudo-random number generator and a connection switch-box [1].

2.2.2 Averaging Techniques

Averaging techniques apply different combinations of DAC input codes so as to average out the mismatch error over some number of input words. These techniques differ from randomization in that the mismatch noise does not become white, but instead takes a distinct spectral shape. In their basic form, these techniques can only be applied to baseband modulators, as the mismatch noise can only be shaped toward higher frequencies [2].

Clocked Averaging

Clocked Averaging (CLA) [20] strives to maintain a long-term average of the mismatch error from all DAC elements. CLA applies the basic thermometer code to the DAC, but increments the start-index every sample. The start-index is the specific bit location where the thermometer code begins. The mismatch noise power is thus spread out over all harmonics of the averaging period, which is $F_S/N$, where $N$ is the number of DAC elements and $F_S$ is the sampling frequency. Clocked averaging does not significantly improve the SNR in the band of interest, but is relatively simple to implement in hardware.
**Individual Level Averaging**

Individual Level Averaging (ILA) [20][21] strives to maintain long-term averages separately for each DAC input code. ILA maintains separate start indices for each DAC input code. The start index for each input code is updated only when that code is encountered. As in CLA, the start index is used to select the location of the first bit of the thermometer code. This results in a mismatch noise shape similar to 1st-order noise shaping, but the implementation complexity increases linearly with the number of DAC output levels.

**Data-Weighted Averaging**

Also known as *element rotation*, Data-Weighted Averaging (DWA) takes an approach similar to the above techniques involving averaging, but instead tries to cycle through all the DAC elements as rapidly as possible [22][23][24]. The start-index of the thermometer code is modulo-incremented according to the magnitude of the DAC input code. This rapid cycle through the elements allows the mismatch error to quickly approach the average mismatch, safely moving most of the mismatch noise power to higher frequencies and out of the band of interest. The implementation complexity is similar to CLA, but the shaping performance is an improvement over ILA. The DWA algorithm can suffer from tonal behavior and many techniques have been reported for preventing this behaviour [25][26][27][28].
2.2.3 Vector-based Mismatch Shaping

The averaging techniques described in 2.2.2 are able to achieve at best a 1st-order shaping. A technique for achieving higher-order mismatch shaping is described in [13][29]. This architecture consists of a \( \Delta \Sigma \) modulator operating on the array of unit-element controls and requires an array of independent loop filters, along with a vector quantizer to create the DAC element controls. The vector quantizer simultaneously operates on the entire array of loop filter outputs, and uses the DAC data to create the correct quantizer output. As with a \( \Delta \Sigma \) modulator, the vector-based mismatch shaper creates a noise-shaped mismatch noise spectrum according to the characteristics of the loop filter used in the array.

This technique allows the location of the mismatch suppression band to be centered at an arbitrary frequency within the Nyquist band by using an array of tunable loop filters. The center frequency of the NTF can then be chosen to match that of the \( \Delta \Sigma \) modulator in order to ensure that mismatch shaping is performed in the correct frequency band. This approach suffers from concerns about stability as well as a high complexity of hardware required to implement the vector quantizer. The vector-based mismatch shaping technique is described in more detail in Chapter 3, where it is extended to operate at tunable center-frequencies.
2.2.4 Tree-based Mismatch Shaping

The tree-based shaper is a more efficient approach to achieving a higher-order mismatch shaping response [15][30][31]. This structure consists of a tree of switching blocks which acts as a router for directing each bit of the DAC input thermometer code to each DAC element. The tree is composed of multiple layers of switching blocks, each of which acts as a router for unit-element control bits. The primary goal of the routing tree is to determine the final location of the bits that control a DAC element.

Each switching block takes as an input a $\Delta\Sigma$-modulated input sequence that is spectrally shaped according to the desired mismatch shaping NTF. As a result, it is possible to achieve higher orders of mismatch noise shaping while centering the signal band in the passband [32]. This approach can have a high implementation complexity, although it is more efficient than the vector-based technique. However, efficient implementations of the tree structure have been reported for baseband mismatch shaping DACs [33][34][35][36]. A major advantage of the tree-based mismatch shaper is the ability to remain relatively immune from spurious tones by using dither [37].

The tree-based mismatch shaper is described in detail in Chapter 4, where it is extended to operate at tunable center frequencies.

2.2.5 $N$-Path Mismatch Shaping

A novel technique for applying a baseband mismatch shaping algorithm to bandpass modulators is described in [38] and [39]. This involves using the
Figure 2.7: Conceptual implementation of $N$-path transform

$N$-path filter principle to replicate the response of a given baseband mismatch shaping algorithm to a bandpass location. Figure 2.7 shows a conceptual commutator model of the $N$-path transform $z \rightarrow z^N$ applied to a mismatch shaping function $H(z)$. In [38], a 4-path transform $z \rightarrow z^4$ is applied to a 1st-order highpass mismatch shaping response $H(z) = 1 - z^{-1}$, created using the DWA algorithm. This produces a mismatch shaping response with suppression zeros at $z = 1$, $z = \pm j$ and $z = -1$. The replication of zeros at $z = \pm j$ allows this structure to be used within a bandpass $\Delta \Sigma$ modulator with the signal band centered at $F_s/4$.

An improvement to this approach is described in [40], where a 2-path transform $z \rightarrow z^2$ is applied to a 1st-order lowpass mismatch shaping response $H(z) = 1 + z^{-1}$, created using a modified version of the DWA algorithm. This approach also produces a mismatch shaping response with zeros at $z = \pm j$, but requires only 2 paths instead of 4. This in turn increases the mismatch shaping bandwidth and lowers the implementation complexity. The $N$-path technique is described in detail in Chapter 5, where the approach is extended to operate at tunable center-frequencies.
2.2.6 Quadrature Mismatch Shaping

Quadrature bandpass ΔΣ data converters are widely used in low-IF transceiver applications [41][42]. In such modulators, each of the in-phase (I) and quadrature (Q) components of the quantized complex signal require a coarse DAC. In order to extend the benefits of multi-bit quantization to quadrature signals, a quadrature mismatch shaper can be used [43][44].

Existing mismatch-shaping techniques for complex-valued DACs require the signal band be located at a fixed frequency. Chapter 6 describes an architecture for tunable quadrature bandpass mismatch shaping which allows the center frequency of the quadrature mismatch transfer function to be adjustable over the entire Nyquist range. The approach is based on the tree-structured mismatch shaper from [44].
Chapter 3

Tunable Vector Mismatch Shaping

The vector mismatch shapers described in [13] and [14] allow some flexibility in the choice of the spectral shape of the noise-transfer function. Previously reported work has placed the signal-band at a fixed frequency location, mostly at DC. In this chapter, the vector mismatch shaper is modified to use tunable loop filters within the vector ΔΣ modulator. The tunable filters allow the location of the mismatch suppression band to be tunable over the entire Nyquist range of the DAC. The design has been implemented using VHDL and synthesized to logic gates for several DAC sizes in order to compare the hardware complexity. Synthesis results show a rapid increase in hardware complexity with DAC size, making the vector shaper an impractical choice for in larger-sized DACs.

Section 3.1 provides an overview of the vector-structured mismatch shaper, as previously reported in the literature. Section 3.2 describes a detailed implementation of the proposed tunable vector architecture. Section 3.3 charts the hardware complexity for various DAC sizes, and presents the mismatch shaping performance. The chapter is summarized and concluded in Section 3.4.
3.1 Overview of the Vector Shaper

The digital to analog converter (DAC) used within a multibit ΔΣ data converter is very often constructed using an array of unit-size elements. Each DAC element is used to convert exactly one least-significant bit of the input data to an analog value. Figure 3.1 shows the structure of a unit-element DAC.

The digital input $u[n]$ to the DAC is quantized to one of $(M + 1)$ integers in the range $[0, M]$. The element-selection logic (ESL) is used to convert $u[n]$ to a vector $\vec{v}[n]$ of $M$ single-bit controls such that:

$$u[n] = \sum_{k=1}^{M} v_k[n], \quad v_k[n] \in \{0, 1\} \quad (3.1)$$

Each control bit $v_k[n]$ drives a single unit DAC element. The DAC output is formed by summing together the outputs of all the DAC elements:

$$d[n] = \sum_{k=1}^{M} d_k[n], \quad d_k[n] = \Delta \cdot v_k[n] \quad (3.2)$$

where $\Delta$ denotes the nominal step-size of a single element.
The output of the ESL block is a vector that contains $u[n]$ positions where the bit value is equal to 1, and $(M - u[n])$ positions where the bit value is equal to 0. The simplest form of element selection is a binary-to-thermometer converter, which converts the DAC input data $u[n]$ to a thermometer code of length $M$. However, if only a simple thermometer code is used for element selection, component mismatch can lead to significant performance degradation. Mismatch errors arise out of manufacturing defects that create non-ideal values for the DAC elements.

The effects of mismatch can be mitigated by making the observation that there are multiple ways in which the input $u[n]$ can be used to select DAC elements to form the output, $\vec{v}[n]$. These additional degrees of freedom can be used to vary the sequence of element selection patterns in such a way as to spectrally shape the mismatch noise away from the signal-band [1]. This is known as mismatch noise shaping, and is achieved without making any changes to the actual number of DAC elements selected for activation.

The vector mismatch shaper [13] is one such form of mismatch shaping ESL, whose basic structure is shown in Figure 3.2. This can either be viewed as an array of $\Delta \Sigma$ modulators, each of which operates on a single DAC element control, or as a vectorized $\Delta \Sigma$ modulator operating on a vector of DAC element controls. The vector loop filter consists of an array of loop filters, each of which operates independently from the others. The vector quantizer operates on the entire vector of loop filter outputs $\vec{y}[n]$, and uses the DAC input data $u[n]$ to create the correct quantizer output, $\vec{v}[n]$. 

21
As shown in Figure 3.2 and described in [13], the loop filter first computes the error $\vec{e}$ of the quantization operation: $\vec{e}[n] = \vec{v}[n] - \vec{y}[n]$. This error vector is then fed to an array of $M$ filters, each of which has a transfer function $NTF(z) - 1$. Before being sent to the vector quantizer, the filter output vector is shifted by a scalar $s[n]$. The vector quantizer first determines the index locations of the $u[n]$ highest elements of $\vec{y}[n]$. These index locations are then identified as the $u[n]$ elements out of $M$ to be selected in the quantized output vector $\vec{v}[n]$. The $M$ DAC unit-elements are then driven by the output vector $\vec{v}[n]$ to create the analog DAC output, as shown in Figure 3.1.

The mismatch shaping operation can be analyzed as follows. Let the mismatch error in the $k^{th}$ DAC element be denoted by $\epsilon_k$, where $k \in [1, M]$. These mismatch errors can be assumed to be zero-mean [2], which makes their sum equal to zero. The scalar $s[n]$ can be viewed as a length $M$ vector:

$$\vec{s} = s[n] \cdot [1, 1, \ldots, 1]$$  \hspace{1cm} (3.3)
The output of the loop filter is given by:

\[
\vec{y} = (NTF - 1) \cdot \vec{v} + \vec{s}
\]  
(3.4)

The output of the vector quantizer is given by:

\[
\vec{v} = NTF \cdot \vec{v} + \vec{s}
\]  
(3.5)

The mismatch error \( v_k \epsilon_k \) from each DAC element creates a total mismatch error of \( \vec{v} \cdot \vec{\epsilon} \) at the DAC output, and this is given by:

\[
\vec{v} \cdot \vec{\epsilon} = (NTF \cdot \vec{v} + \vec{s}) \cdot \vec{\epsilon}
\]  
(3.6)

\[
= NTF \cdot \vec{v} \cdot \vec{\epsilon} + s \cdot [1, 1, \ldots, 1] \cdot \vec{\epsilon}
\]  
(3.7)

The second term in Equation 3.7 is eliminated because the mismatch errors are assumed to be zero-mean, and hence the elements of the mismatch error vector \( \vec{\epsilon} \) sum to zero. The mismatch error at the DAC output is therefore given by:

\[
\vec{v} \cdot \vec{\epsilon} = NTF \cdot \vec{v} \cdot \vec{\epsilon}
\]  
(3.8)

As described in [13], Equation 3.8 shows that the mismatch errors can indeed be suppressed by an appropriately chosen filter \( NTF \), as long as \( \vec{v} \vec{e} \) remains bounded. This can be done by choosing the value of the scalar \( s[n] \) as the negated minimum value of the loop filter output, as shown in Figure 3.2. Due to the non-linear and complicated nature of the vector quantizer, loop stability needs to be confirmed by extensive simulation [2][13].
3.2 Tunable Implementation

In this section, the vector shaper described in Section 3.1 is extended to allow mismatch shaping at arbitrary locations of the signal band. According to Equation 3.8, the mismatch noise can be spectrally shaped by the loop filter shown in Figure 3.2, and the order and frequency location of the noise shaping function are determined by $NTF(z)$. Therefore, **tunable** mismatch shaping can be achieved by using an array of tunable loop filters and simultaneously controlling their tuning frequencies. The structure of the tunable loop filter is described in Section 3.2.1.

The mismatch shaper requires a vector quantizer to create the DAC element selection vector $\mathbf{v}[n]$ according to the element usage requirements $u[n]$ and the vector output $\mathbf{y}[n]$ of the loop filter array. The vector quantizer simply selects the highest $u[n]$ values from the $\mathbf{y}[n]$ vector, and enables the corresponding index positions in $\mathbf{v}[n]$. In order to select the highest $u[n]$ values of the loop output $\mathbf{y}[n]$, the vector $\mathbf{y}[n]$ must first be sorted.

As shown in Figure 3.2, the loop filter output vector is shifted by a scalar value $s[n]$, given by:

$$s[n] = min(\mathbf{y}[n]), \quad \mathbf{y}[n] = \langle y_1[n], y_2[n], \ldots, y_M[n] \rangle$$  \hspace{1cm} (3.9)

This minimum value is produced incidentally as a consequence of the sorting operation within the vector quantizer. As a result, it is more efficient to rearrange the vector shaper from Figure 3.2 to instead look like that shown in Figure 3.3. The scalar $s[n]$ is now produced by the vector quantizer and fed
back to the loop filter for subtraction within the same step as the creation of the $\vec{e}[n]$ vector. It is of no consequence that the version of $\vec{y}[n]$ which now enters the vector quantizer is the raw loop filter output, because the sorted order remains unchanged. The structure of the vector quantizer is described in Section 3.2.2.

### 3.2.1 Tunable Loop Filter

Figure 3.4 shows the structure of the loop filter array, with detail of a single copy of a $2^{nd}$-order tunable filter. This filter effectively performs $1^{st}$-order mismatch shaping at each tuning setting. A $2^{nd}$-order filter is required for $1^{st}$-order mismatch shaping in order to place complex-conjugate NTF zeros around the unit circle. There is a single tunable coefficient, which is a direct function of the tuning frequency: $c = -2\cos\omega$, where $\omega = 2\pi F_c$. This effectively places the zero locations at $z = e^{j\omega}$ and $z = e^{-j\omega}$.
Ignoring the scalar shift $s[n]$, the loop filter transfer function $L(z)$ can be expressed as:

$$L(z) = \frac{-z^{-1}2\cos\omega + z^{-2}}{1 - z^{-1}2\cos\omega + z^{-2}}$$

(3.10)

The noise transfer function is given by:

$$NTF(z) = \frac{1}{1 - L(z)}$$

(3.11)

$$NTF(z) = \frac{(z - e^{j\omega})(z - e^{-j\omega})}{z^2}$$

(3.12)

Figure 3.4(a) shows an early retimed structure and Figure 3.4(b) shows a late retimed structure. Either of the two equivalent structures can be used, depending on when the DAC input data is available within the clock cycle. The early-retimed structure is used when the DAC input data is available early in the clock cycle, and vice-versa. Also shown in Figure 3.4 is the addition of dither (the equivalent of 0.5LSB) to reduce spurious tones. Pseudo-random dither is generated using a linear-feedback shift register (LFSR) [45].

Figure 3.5 shows the values of the coefficient $c$ plotted over the entire tuning range. The coefficients can easily be stored in a lookup-table or read-only memory (ROM). As the coefficient is simply a scaled cosine function of the tuning frequency, a direct digital frequency synthesizer (DDFS) can be used to generate the coefficient as needed [46][47]. This would provide greater flexibility during selection of the tuning frequency. The datapath and coefficient widths are listed in Table 3.1.
Figure 3.4: Tunable loop filter using 2\textsuperscript{nd}-order structure
Table 3.1: Coefficient and datapath widths for loop filter

<table>
<thead>
<tr>
<th>Coefficient width CW (bits)</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path width PW (bits)</td>
<td>11</td>
</tr>
</tbody>
</table>

3.2.2 Vector Quantizer

As described above, the vector quantizer uses the original positions of the $u[n]$ highest values in the loop filter output vector $\vec{y}[n]$ to determine the indices of the DAC elements to enable in $\vec{v}[n]$. In order to find the $u[n]$ highest values, the vector $\vec{y}[n]$ must first be sorted. The final sorted values themselves are unimportant, as only the indices of their sorted positions are needed. These indices are then used to identify the DAC elements to be selected in $\vec{v}[n]$.

This process is repeated every sample, so the sample index $n$ is omitted in the following description. To keep track of element positions before sorting, an index $k$ is first assigned to each loop filter output $y_k$ to form the element pair $(y_k, i_k)$, where $i = k$ and $k \in [1, M]$. This vector sequence of pairs is
then sorted using the value of $y_k$ to produce a sorted vector sequence of pairs $(s y_k, s i_k), k \in [1, M]$, as shown in Figure 3.6. Assuming the sorting process produces a sequence in descending order, the $u$ highest elements in $s y_k$ are now indexed by $k = 1, 2, \ldots, u$. Now, the indices that correspond to $s y_k$ ($k \in [1, u]$) are given by $s i_k$ ($k \in [1, u]$). Therefore, the indices of the DAC elements to be selected are also $s i_k, k \in [1, u]$. The DAC output can be defined as:

$$v_k[n] = \begin{cases} 1 & \text{if } k = s i_1, s i_2, \ldots, s i_u \\ 0 & \text{if } k = s i_{u+1}, s i_{u+2}, \ldots, s i_M \end{cases} \quad (3.13)$$

Sorting is performed using a parallel sorting network [48], which is a comparison network in which multiple comparisons can be performed in parallel. In particular, the modular construction described in [49] is the exact structure employed in this work for vector quantizers of all DAC sizes. This structure consists of a recursive cascade of merging networks, as shown in Figure 3.7(a). Using a parallel network of comparators, each merger unit takes a pair of sorted sequences and merges them into a single sorted sequence. Figure 3.7(b) shows the merging network detail using a network of two-input comparators. The symbol and structure of a single comparator is shown in

![Figure 3.6: Structure of the vector quantizer](image)
Figure 3.8(a). The symbol corresponds to that used in the merging network shown in Figure 3.7(b). The input values $x_m$ and $y_n$ are compared and then conditionally swapped along with the indices $m$ and $n$. The logic structure of each conditional swapper is shown in Figure 3.8(b).

The depth of this type of $M$-input sorting network is given by $D(M) = D(M/2) + \log M$, where $M = 2^k$ and $k \geq 1$. This means that the sorting network can sort $M$ numbers in $O(\log^2 N)$ time. Despite this vast improvement over serial sorting algorithms, the critical path through the vector quantizer grows quickly as the DAC size increases. The next section presents the shaping performance and hardware complexity of the vector shaper.

### 3.3 Performance and Complexity

The tunable vector shaper has been implemented in VHDL and simulated using data generated by a tunable bandpass $\Delta \Sigma$ modulator. The specifications of the modulator are shown in Table 3.2. The DAC mismatch errors are modeled as uniformly distributed random mismatch, with the standard deviation expressed as a percentage of a single unit element value.

<table>
<thead>
<tr>
<th>Modulator order</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSR</td>
<td>64</td>
</tr>
<tr>
<td>Quantizer levels</td>
<td>variable</td>
</tr>
<tr>
<td>Tuning settings</td>
<td>64</td>
</tr>
<tr>
<td>Normalized signal bandwidth</td>
<td>0.0078</td>
</tr>
</tbody>
</table>

Table 3.2: Tunable bandpass $\Delta \Sigma$ modulator specifications
Figure 3.7: Sorting network using an 8-input merging network
Shaping Performance

Figures 3.9–3.15 show the simulated SNR for vector shapers with DAC sizes of 2, 4, 8, 16, 32, 64 and 128 unit elements, respectively. In each figure, the SNR with no mismatch is that of a DAC without any mismatch errors, where the performance is limited by the tunable bandpass ΔΣ modulator specified in Table 3.2. The unshaped SNR is the response when mismatch is added, but no shaping is performed. In this case, the element selection is performed using a simple thermometer code. The shaped SNR is the response when mismatch is added and mismatch shaping is performed using the vector shaper. Each data-point corresponds to the SNR average computed for a thousand different randomly-generated mismatch DAC configurations, at each tuning frequency setting. In all figures, the standard deviation of the applied mismatch is 3% of the unit element step-size. The cases of 1% and 2% mismatch are similar.
Figure 3.9: SNR with 3% mismatch for a 2-element vector shaper

Figure 3.10: SNR with 3% mismatch for a 4-element vector shaper
Figure 3.11: SNR with 3% mismatch for a 8-element vector shaper

Figure 3.12: SNR with 3% mismatch for a 16-element vector shaper
Figure 3.13: SNR with 3% mismatch for a 32-element vector shaper

Figure 3.14: SNR with 3% mismatch for a 64-element vector shaper
Figure 3.15: SNR with 3% mismatch for a 128-element vector shaper

**Hardware Complexity**

The 1\textsuperscript{st}-order tunable vector mismatch shaper has been implemented using VHDL and tested for DAC sizes of 2, 4, 8, 16, 32, 64 and 128 unit-elements. The designs for each DAC precision have been synthesized using Synopsys Design Compiler \cite{50} under the conditions specified in Table 3.3. These conditions represent the slowest operational point of the logic gates in this manufacturing process library.

<table>
<thead>
<tr>
<th>Process</th>
<th>0.18 µm CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device corners</td>
<td>slow-NMOS / slow-PMOS</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.62 V</td>
</tr>
<tr>
<td>Temperature</td>
<td>125°C</td>
</tr>
</tbody>
</table>
Figures 3.16 and 3.17 show the synthesis results for each DAC size. Figure 3.16 shows the critical path delay in nanoseconds (ns) plotted against DAC size. Figure 3.17 shows the cumulative cell area corresponding to the points in Figure 3.16. The area is calculated in square-microns ($\mu m^2$) and plotted on a logarithmic scale. The cell area does not include routing area, as silicon layouts were not created for the designs. The overhead for routing and layout can be expected in the range of 10-15%. This overhead depends on the area utilization during the place-and-route process. When 3 or more layers of metal are available, 85% utilization can be achieved with careful control of the auto-router.
3.4 Summary

Unit-element DACs are commonly used in over-sampled data converters to convert a coarsely-quantized signal to analog form. Manufacturing defects can cause mismatch between the unit-sized elements, thereby creating large drops in converter dynamic range. Mismatch noise shaping is an established technique that spectrally shapes the mismatch noise away from the signal band. Existing mismatch shaping techniques have placed the signal band at fixed center frequencies, usually $DC$ or $F_S/4$, where $F_S$ is the sampling rate. This chapter presents a technique for extending the benefits of mismatch shaping to tunable applications.
The vector mismatch shaper is a well-known technique for achieving higher-order mismatch noise shaping, and has been used for shaping mismatch noise away from signal bands at DC [13] and \(F_s/4\) [43]. In this chapter, the vector mismatch shaper has been extended to allow the signal band to operate at an arbitrary center frequency. This is achieved by using a bank of tunable 2\(^{nd}\)-order loop filters, each of which is designed to create a 1\(^{st}\)-order noise-shaped suppression band located at an arbitrary frequency in the tuning range. The tunable loop filters can be slightly modified to work with either early- or late-arriving DAC input data. The vector quantizer used within the vector shaper is implemented using a parallel sorting network.

The tunable vector shaper has been implemented in VHDL for various DAC sizes. The designs have been synthesized using a 0.18\(\mu\)m CMOS technology, running at a nominal voltage of 1.8\(V\). Logic synthesis results show that both the critical path delay and the cell area increase exponentially with increasing DAC size. The designs have been simulated with data generated from a tunable 6\(^{th}\)-order \(\Delta\Sigma\) modulator. An over-sampling ratio of 64 allows coverage of the entire Nyquist band using only 64 tuning settings. Simulation results show that the vector mismatch shaper consistently improves performance for all center frequencies across the entire tuning range.
Chapter 4

Tunable Tree Mismatch Shaping

The tree-structured mismatch shaper is an effective approach to achieving higher-order mismatch shaping responses [15][30][31]. This structure allows some flexibility in the choice of the spectral shape of the noise-transfer function. Most previously reported work has placed the signal-band at a fixed frequency location, mostly at DC. The exception to this is a fully-analog implementation of the tree for creating a variable frequency location for the signal-band, tunable only over a small range of frequencies [9].

In this chapter, the tree-structured approach to mismatch-shaping is modified and extended in order to allow the signal band location to be tunable over the entire Nyquist range of the DAC. The design has been implemented using VHDL and synthesized to logic gates for several DAC sizes in order to compare hardware complexity. Synthesis results show a reduction in hardware complexity as compared to the vector mismatch shaper. This improvement becomes quite dramatic for larger DAC sizes.

Section 4.1 provides an overview of the tree-structured mismatch shaper, as previously reported in the literature. Section 4.2 presents three implementations of the proposed tunable architectures. Section 4.3 compares the
mismatch shaping performance and complexity of the various tree-structured architectures, while Section 4.4 compares the tree-structured architecture to the vector-structured architecture. The chapter is summarized and concluded in Section 4.5.

4.1 Overview of the Tree Structure

Figure 4.1 shows the structure of a unit-element DAC used in multibit ΔΣ data converters. The element-selection logic (ESL) is used to convert the DAC input $x[n]$ into a vector of $M$ single-bit controls to the unit-element array:

$$x[n] = \sum_{k=1}^{M} x_k[n], \quad x_k[n] \in \{0, 1\} \quad (4.1)$$

The DAC output is formed by summing together the outputs of all the DAC unit-elements (where $\Delta$ denotes the nominal step-size of a single element):

$$y[n] = \sum_{k=1}^{M} y_k[n], \quad y_k[n] = \Delta \cdot x_k[n] \quad (4.2)$$

Component mismatch due to manufacturing defects leads to non-ideal values for the DAC elements. As a result, each DAC element exhibits mismatch errors:

$$y_k[n] = \begin{cases} \Delta + \epsilon_{hk} & \text{if } x_k[n] = 1 \\ \epsilon_{lk} & \text{if } x_k[n] = 0 \end{cases} \quad (4.3)$$

where $\epsilon_{hk}$ and $\epsilon_{lk}$ denote the static mismatch error when the DAC element is enabled and disabled, respectively. The overall DAC output is given by:

$$y[n] = \alpha x[n] + \beta + \epsilon[n], \quad (4.4)$$
Figure 4.1: Unit-element DAC with element selection logic

where $\alpha$, $\beta$ and $\epsilon$ are the gain error, offset error and aggregate mismatch error, respectively, and all three quantities depend exclusively on the element mismatch errors [15].

During element selection, there are multiple ways in which the input $x[n]$ can be used to select DAC elements to form the output $y[n]$. These additional degrees of freedom can be used to vary the sequence of element selection patterns in such a way as to spectrally shape the mismatch noise away from the signal-band [1]. This is known as mismatch noise shaping, and is achieved without changing the actual number of selected DAC elements from the input DAC data.

The mismatch shaper used in this work is based on the tree-structured approach described in [15] and adapted to allow the signal band to be centered at an arbitrary frequency within the Nyquist range. The tree-structured mismatch noise shaper performs element selection through the use of a tree of switching blocks, which acts as a router for directing each DAC input unit
value to a specific DAC unit element. The primary goal of the tree is to determine the final location of each bit that controls a given DAC unit element.

Figure 4.2 shows an example of an 8-element DAC using the tree structure to perform element selection. Each of the blocks labeled $S_k$, $r$ is a switching block that routes the input data in two possible directions. There are $\log_2 M$ layers of switching blocks. Each switching block in the final layer drives out a pair of single-bit values to control a pair of DAC elements. As the input travels through the tree, portions of the input data word are spread across the branches of the tree until they arrive at the inputs of the unit-DACs, where only a single bit determines whether or not the DAC element is selected for activation.
Each switching block operates according to:

\[
x_{k-1,2r-1}[n] = \frac{1}{2}(x_{k,r}[n] + s_{k,r}[n]) \quad (4.5)
\]
\[
x_{k-1,2r}[n] = \frac{1}{2}(x_{k,r}[n] - s_{k,r}[n]) \quad (4.6)
\]

where \( s_{k,r}[n] \) is a switching sequence generated within the switching block. The value of the switching sequence \( s_{k,r}[n] \) determines the portion of the input data \( x_{k,r}[n] \) that is routed through each of the two outputs of the switching block [15]. Figure 4.3 shows the structure of a switching block, including the switching sequence generator, that implements the operations described by Equations 5.13 and 5.14.

Since the actual data must remain unchanged from the input of the ESL block to the output, each switching block must ensure that it generates a switching sequence that forces the output data to satisfy this condition. This restriction is known as the number conservation rule. Ensuring that all switching blocks satisfy the following equations is sufficient to guarantee that
the number conservation rule is observed [15]:

\[
s_{k,r}[n] = \begin{cases} 
  \text{EVEN} & \text{if } x_{k,r}[n] \text{ is EVEN} \\
  \text{ODD} & \text{if } x_{k,r}[n] \text{ is ODD}
\end{cases} \quad (4.7)
\]

\[|s_{k,r}[n]| \leq \min\{x_{k,r}[n], 2^k - x_{k,r}[n]\} \quad (4.8)\]

The DAC mismatch error sequence \(\epsilon[n]\) can be expressed as:

\[
\epsilon[n] = \sum_k \sum_r \Delta_{k,r}s_{k,r}[n] \quad (4.9)
\]

where \(\Delta_{k,r}\) is the nominal value of the unit-DAC step size. If the switching sequence \(s_{k,r}[n]\) is generated as an \(L^{th}\)-order noise-shaped sequence, uncorrelated with the switching sequences in the other switching blocks, this will result in an \(L^{th}\)-order noise-shaped DAC mismatch error sequence [15].

Figure 4.4 shows the structure of the sequence generator. Noise shaping is achieved by employing a zero-input \(\Delta\Sigma\) modulator, with an \(L^{th}\)-order loop filter designed to achieve the desired noise-transfer function for the mismatch noise. As described in [15], the number conservation rule is enforced by the quantizer and limiter blocks. Equation 4.7 is satisfied by using the least-significant bit (LSB) of the input \(x_{k,r}[n]\) to the switching block to choose
between either a mid-rise or mid-tread quantizer, which are both shown in Figure 4.5. Subsequently, the limiter block restricts the output of the sequence generator from exceeding the range described by Equation 4.8.

4.2 Tunable Implementation

As described in the previous section, the order and frequency location of the noise shaping function are determined by the ∆Σ loop filter shown in Figure 4.4. It follows that in order to gain control over the center frequency of the mismatch transfer function, there needs to be a way to control the center frequency of the noise-shaped switching sequence itself. Therefore, tunable mismatch shaping can be achieved by *simultaneously controlling the loop filters in all the switching blocks*, as shown in Figure 4.6. This section presents the implementation details of different tunable loop filter architectures for use specifically within the sequence generator.
Section 4.2.1 presents an architectural approach to reducing the critical path delay through the tunable tree, by applying register retiming. Section 4.2.2 describes the design of the noise-transfer function (NTF) of each sequence generator. Section 4.2.3 presents two architectures for the tunable sequence generator with optimized NTFs and realized as a cascade of resonators with distributed feed-back (CRFB). These architectures implement tunable 1st- and 2nd-order mismatch shapers using tunable 2nd- and 4th-order CRFB structures, respectively. Section 4.2.4 presents an alternative realization of the tunable sequence generator using a tunable 2nd-order resonator in polar form, without optimizing the noise-transfer function.
4.2.1 Retiming the Loop Filters

One problem that arises in the implementation of the tree structure for larger DAC sizes is an exceedingly long critical path delay through the datapath, especially when using an identical structure for all sequence generators in the tree. As shown in Figure 4.6, the input data to the ESL block flows through each layer of the tree, passing through the adder and subtracter in each switching block, until finally reaching the DAC unit-elements. This critical path is further exacerbated by the sequence generator if the loop filter output is not ready when the input data arrives at that layer, as shown in Figure 4.4.

If the latency through the ESL block is not an issue, then the critical path can be reduced by the addition of pipeline registers at suitable points within the tree structure. However, this is not a viable option for applications which require the ESL block to have a low latency. An example of such an application would be a high-speed ∆Σ ADC in which any added loop delay threatens the stability of the ∆Σ modulator. It is therefore an advantage to reduce the critical path by careful design of the loop filter.

Retiming the loop filter data-paths is a viable approach to reducing the critical path delays [51]. If the sequence generator loop is retimed by placing a loop filter delay element (or register) directly at the output of the loop filter, as shown in Figure 4.7(a), the filter outputs can be made available at the start of the clock cycle. The filter output would now be ready to enter the quantizer and limiter blocks as soon as the input data reaches that layer.
However, if all the sequence generators are designed in this way, as shown in Figure 4.8, the critical path would effectively remain the same because the sequence outputs in the last layer of sequence generators would still need to ripple back through the loop filters before reaching the registers that capture the loop filter outputs. This problem can be alleviated by retiming the sequence generator loop filters in such a way that a loop filter delay register appears near the input of the loop filter, as shown in Figure 4.7(b).

These can be used in the latter stages of the tree structure. Figure 4.9 shows the ideal solution, in which variously retimed loop filter architectures are employed so that the bottle-necks are gradually relieved as the data flows through the tree. Unfortunately, the cost of this approach can grow exponentially if retiming registers are placed within large datapath functional blocks, such as multiply-accumulate (MAC) units [51].

In this work, only two retimed versions of the loop filter are used: the early and the late retimed structures, as shown in Figure 4.10. The early retimed structure ensures that the loop filter output is ready near the beginning of the clock cycle, and is used in the earlier stages of the tree. The late retimed
Figure 4.8: Critical path using a single retiming method

structure ensures that once the input data arrives at the quantizer/limiter, there is a minimum amount of logic until the next register. This structure is used in the later stages of the tree.

The determination of the exact tree stage location at which to place the boundary between early and late retimed structures has been made through empirical methods, using a mixture of logic synthesis and simulation. The various architectures have been experimentally synthesized with the boundary placed at new locations until the shortest critical path is achieved. Since the number of tree layers is only $\log_2 M$, where $M$ is the number of DAC elements, the number of possible boundary points grows very slowly as the DAC size is increased.
Figure 4.9: Critical path using more than two retiming methods

Figure 4.10: Critical path using only two retiming methods
4.2.2 Designing the Tunable Noise-Transfer Function

As described in Section 4.1, the spectral shape of the switching sequence determines the spectral shape of the DAC mismatch noise. As shown in Figure 4.4, the sequence generator creates a noise-shaped sequence using the same structure as a $\Delta\Sigma$ modulator (but without an input signal). Therefore, the sequence generator loop filter can be designed in much the same way as that of a conventional $\Delta\Sigma$ modulator.

When designing the loop filter of a conventional $\Delta\Sigma$ modulator, the noise-transfer function (NTF) must first be created [1][2]. In the case of a tunable mismatch shaper, the desired NTF must be created at each tuning setting. The first step is to place pairs of complex conjugate NTF zeros at those points on the unit circle that correspond to the tuning frequency. These mark the frequencies at which there will be maximum attenuation of the mismatch noise. The second step is to determine the best locations for the NTF poles. A major issue that plagues a $\Delta\Sigma$ modulator is the threat of instability due to the severe non-linearity caused by quantizer overload [1]. In the case of the sequence generator loop, there is the added complication of data-dependent selection of either a mid-rise or a mid-tread quantizer, as well as the data-dependent restriction placed by the limiter. In addition, there must be at least one delay in the loop in order for the loop filter to be realizable [2].

In order to reduce the potential for creating an unstable system, it is useful to adhere as closely as possible to the Lee criterion [2][52][53], which states that a binary $\Delta\Sigma$ modulator with a noise transfer function $NTF(z)$ is
likely to be stable if \( \forall \omega : \max |NTF(e^{j\omega})| < 1.5 \). This effectively translates to restricting the maximum NTF gain to less than 1.5 over all frequencies.

In the following sections, two different ways of designing the tunable NTF are explored. Section 4.2.3 describes the first approach, which uses optimization to arrive at the ideal locations of the NTF poles and zeros for each tuning setting. This is applied to both 1st- and 2nd-order shaping. Section 4.2.4 describes the second approach, which uses a form of the loop filter that simplifies the implementation complexity for 1st-order shaping.

### 4.2.3 CRFB Architectures for 1st- and 2nd-order Shaping

The loop filters presented in this section have been designed using the Delta-Sigma Toolbox [54], a collection of useful programs for simulating \( \Delta \Sigma \) modulators. The NTF is optimized at each tuning setting by using the center frequency of the signal band, \( F_c \), and the maximum NTF gain, \( \max |NTF(e^{j\omega})| \). For 1st-order noise shaping, a 2nd-order tunable filter is optimized with \( \max |NTF(e^{j\omega})| = 1.4 \). For 2nd-order noise shaping, a 4th-order tunable filter is optimized with \( \max |NTF(e^{j\omega})| = 1.2 \). This process has been repeated for all DAC sizes. In each case, the loop filter has been realized using a cascade of resonators with distributed feed-back (CRFB) structure [2]. The coefficients are quantized to a precision that balances the implementation complexity with the performance of the mismatch noise suppression. In the following two sections, implementation details are provided for these CRFB architectures.
4.2.3.1 Tunable 1st-order Shaping

In order to perform 1st-order noise shaping, a 2nd-order tunable loop filter is required in the sequence generator. The filter order is twice the order of shaping because there needs to be a complex-conjugate counterpart for all band-pass zeros and poles if the loop filter is to produce real-valued signals. Figure 4.11(a) shows the early retimed and Figure 4.11(b) shows late retimed versions of the 2nd-order tunable CRFB sequence generator. All results of MAC operations are truncated before storing in registers, and the equivalent to 0.5 LSB of dither is added prior to each truncation point. Dither is generated using a linear-feedback shift register (LFSR) [45].

The datapath and coefficient widths for each DAC size are listed in Table 4.1. The widths are denoted by symbols that correspond to those shown in Figure 4.11. The datapath widths change according to the layer in which the sequence generator is operating, and are progressively reduced as the lower layers of the tree are encountered.

Figure 4.12 shows the values of the three coefficients, \((a_1, a_2)\) and \((g_1)\), plotted over the entire tuning range. The coefficients can easily be stored in a lookup-table or ROM. As the coefficients vary relatively smoothly in each tuning half, the size of the ROM can be further reduced by using polynomial interpolation to compute the coefficients as needed [46]. This would also provide flexibility for selecting the center frequency of the signal band.
Figure 4.11: Tunable loop filter using 2\textsuperscript{nd}-order CRFB structure
Figure 4.12: Tuning coefficients used in the 2nd-order CRFB structure
Table 4.1: Coefficient and datapath widths for CRFB2 structure

<table>
<thead>
<tr>
<th>DAC elements</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total layers of tree</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>A1 width A1W (bits)</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2 width A2W (bits)</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G1 width G1W (bits)</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path width PW (bits)</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>Sequence width SW (bits)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

4.2.3.2 Tunable 2nd-order Shaping

In order to perform 2nd-order bandpass noise shaping, a 4th-order tunable loop filter is required in the sequence generator. Figure 4.13(a) shows the early retimed version and Figure 4.13(b) shows the late retimed version of the 4th-order tunable CRFB sequence generator. As in the 2nd-order case, all results of MAC operations are truncated before storing in registers, and the equivalent to 0.5 LSB of dither is added prior to each truncation point. Pseudo-random dither is generated by a LFSR.

The datapath and coefficient widths for each DAC size are listed in Table 4.2. The bit-width symbols correspond to those shown in Figure 4.13. The realized coefficients, \((a_1, a_2, a_3, a_4)\) and \((g_1, g_2)\), are plotted over the entire tuning range in Figure 4.14. The coefficients again exhibit smooth variation over each half of the tuning range, and the size of the lookup table ROM can be reduced by using polynomial interpolation to compute the coefficients as needed [46].
Figure 4.13: Tunable loop filter using 4\textsuperscript{th}-order CRFB structure
Figure 4.14: Tuning coefficients used in the 4th-order CRFB structure
Table 4.2: Coefficient and datapath widths for CRFB4 structure

<table>
<thead>
<tr>
<th>DAC elements</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 width</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>A2 width</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G1 width</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3 width</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4 width</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2 width</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path width</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence width</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

4.2.4 Polar Architecture for 1st-order Shaping

The CRFB architectures described above are suitable when optimized placement of NTF poles and zeros is required. In the case of 1st-order shaping, an alternative approach to designing the 2nd-order tunable filter can also yield good results while achieving a lower hardware complexity. Figure 4.15 shows the structure of the sequence generator using this approach. Figure 4.15(a) shows the early retimed structure and Figure 4.15(b) shows the late retimed structure. The equivalent of 0.5LSB of dither is added prior to truncation in order to reduce the occurrence of spurious tones. Pseudo-random dither is generated using a LFSR. There is only a single tunable coefficient, which is a direct function of the tuning frequency: \( c = -2\cos\omega \), where \( \omega = 2\pi F_c \). This places the zero locations at \( z = e^{j\omega} \) and \( z = e^{-j\omega} \). The factor \( (1 - r) \) shown along the feedback path alters the locus of the NTF poles but not the zeros.
Figure 4.15: Tunable loop filter using 2nd-order Polar structure
The loop filter transfer function $L(z)$ is:

$$L(z) = \frac{-z^{-1}(1-r)2\cos\omega + z^{-2}(1-r)}{1 - z^{-1}2\cos\omega + z^{-2}}$$  \hspace{1cm} (4.10)

The noise transfer function is given by:

$$NTF(z) = \frac{1}{1 - L(z)}$$  \hspace{1cm} (4.11)

$$NTF(z) = \frac{z^2 - z(2\cos\omega) + 1}{z^2 - z(2r\cos\omega) + r}$$  \hspace{1cm} (4.12)

The pole locations are constrained by opposing design goals. Placing the poles too close to the zeros narrows the spectral shape of the NTF around the tuning frequency, thereby reducing the bandwidth over which mismatch noise is sufficiently suppressed. Placing the poles too far away from the zeros can increase the value of $\max|NTF(e^{j\omega})|$ beyond safety. As discussed in Section 4.2.2, the value of $\max|NTF(e^{j\omega})|$ must be kept low in order to reduce the risk of modulator instability. The proposed work uses $r = 0.5$, and this simplifies the hardware implementation while providing reasonable shaping performance over the entire tuning range.

The datapath and coefficient widths for each DAC size are listed in Table 4.3. The widths are denoted by symbols that correspond to those shown in Figure 4.15. The datapath widths change according to the layer in which the sequence generator is operating. As in case of the CRFB architectures, the datapath progressively reduces in width as the lower layers of the tree are encountered.
Table 4.3: Coefficient and datapath widths for polar structure

<table>
<thead>
<tr>
<th>DAC elements</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total layers of tree</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Coefficient width $CW$ (bits)</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path width $PW$ (bits)</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>Sequence width $SW$ (bits)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Figure 4.16: Tuning coefficients used in the 2nd-order polar structure

Figure 4.16 shows the values of the coefficient $c$ plotted over the entire tuning range. The coefficients can easily be stored in a lookup-table or read-only memory (ROM). As the coefficient is a scaled cosine function of the tuning frequency, a direct digital frequency synthesizer (DDFS) can be used to generate the coefficient as needed [46][47], providing greater flexibility for selecting the tuning frequency.
4.3 Comparison of Tunable Tree Architectures

In this section, the various tree-structured architectures are compared. These include the 2nd- and 4th-order tunable CRFB structures described in Section 4.2.3 and the 2nd-order tunable polar structure described in Section 4.2.4. Each architecture has been implemented in VHDL and simulated using data generated by a tunable 6th-order bandpass $\Delta\Sigma$ modulator. The specifications of the modulator are shown in Table 4.4. An oversampling ratio of 64 allows coverage of the entire Nyquist band using only 64 tuning settings. The DAC mismatch errors are modeled as uniformly distributed random mismatch, with the standard deviation expressed as a percentage of a single unit element value.

Table 4.4: Tunable bandpass $\Delta\Sigma$ modulator specifications

<table>
<thead>
<tr>
<th>Modulator order</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSR</td>
<td>64</td>
</tr>
<tr>
<td>Quantizer levels</td>
<td>variable</td>
</tr>
<tr>
<td>Tuning settings</td>
<td>64</td>
</tr>
<tr>
<td>Normalized signal bandwidth</td>
<td>0.0078</td>
</tr>
</tbody>
</table>

**Shaping Performance**

Figures 4.17–4.23 compare the simulated SNR of all tree shapers with DAC sizes of 2, 4, 8, 16, 32, 64, and 128 unit elements, respectively. In each figure, the SNR with no mismatch is that of a DAC without any mismatch errors, where the performance is limited by the tunable bandpass $\Delta\Sigma$ modulator specified in Table 4.4. The unshaped SNR is the response when mismatch is added, but no shaping is performed. In this case, the element selection is
Figure 4.17: SNR with 3% mismatch for 2-element tree shapers performed using a simple thermometer code. The shaped SNR is the response when mismatch is added and mismatch shaping is performed using the corresponding shaper. Each data-point corresponds to the SNR average computed for a thousand different randomly-generated mismatch DAC configurations, at each tuning frequency setting. In all figures, the standard deviation of the applied mismatch is 3% of the unit element step-size. The cases of 1% and 2% mismatch are similar. As can be expected, the 2\(^{nd}\)-order mismatch shaper (CRFB4) shows the highest performance across all DAC sizes, while the two 1\(^{st}\)-order mismatch shapers (Polar2 and CRFB2) exhibit similar performances. It can also be noted that the effectiveness of mismatch shaping decreases with increasing DAC size.
Figure 4.18: SNR with 3% mismatch for 4-element tree shapers

Figure 4.19: SNR with 3% mismatch for 8-element tree shapers
Figure 4.20: SNR with 3% mismatch for 16-element tree shapers

Figure 4.21: SNR with 3% mismatch for 32-element tree shapers
Figure 4.22: SNR with 3% mismatch for 64-element tree shapers

Figure 4.23: SNR with 3% mismatch for 128-element tree shapers
Hardware Complexity

All the tree-structured mismatch shapers (2\textsuperscript{nd}- and 4\textsuperscript{th}-order tunable CRFB structures, 2\textsuperscript{nd}-order tunable polar structure) have been implemented using VHDL and tested for several DAC sizes. The designs for each DAC precision have been synthesized using Synopsys Design Compiler [50] under the conditions specified in Table 4.5.

Table 4.5: Hardware synthesis conditions

<table>
<thead>
<tr>
<th>Process</th>
<th>0.18 (\mu)m CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device corners</td>
<td>slow-NMOS / slow-PMOS</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.62 V</td>
</tr>
<tr>
<td>Temperature</td>
<td>125°C</td>
</tr>
</tbody>
</table>

Synthesis results are shown for each DAC size in Figures 4.24 and 4.25. For each structure, the critical path, or shortest cycle-time, is plotted against DAC size in Figure 4.24. The structures all exhibit similar critical path delays even as the DAC size is increased. This is because all three structures have a single MAC in the critical path of each sequence generator loop filter in the tree. Figure 4.25 shows the cumulative cell area corresponding to the points in Figure 4.24. The area is calculated in square-microns (\(\mu m^2\)) and plotted on a logarithmic scale. As can be expected, the 4\textsuperscript{th}-order tunable CRFB structure exhibits the greatest area for all DAC sizes. The 2\textsuperscript{nd}-order CRFB and the 2\textsuperscript{nd}-order polar tunable structures both exhibit similar hardware complexity, but it is notable that the polar structure is consistently smaller than the CRFB structure for all DAC sizes.
Figure 4.24: Comparison of tree critical path delays

Figure 4.25: Comparison of tree hardware complexity
4.4 Comparison with Tunable Vector Structure

In this section, the tunable tree structure is compared with the tunable vector structure. Each architecture has been simulated using data generated by the tunable bandpass ΔΣ modulator specified in Table 4.4. The tree structure used in this comparison is the tunable 2nd-order polar architecture for 1st-order mismatch shaping described in Section 4.2.4. The vector architecture is the tunable 2nd-order polar architecture for 1st-order mismatch shaping described in Chapter 3.

Shaping Performance

Figures 4.26−4.32 compare the simulated SNR of the tree and vector shapers with DAC sizes of 2, 4, 8, 16, 32, 64, and 128 unit elements, respectively. In each figure, the ideal SNR is that of the DAC without any mismatch errors and the performance is limited by the tunable bandpass ΔΣ modulator specified in Table 4.4. The unshaped SNR is the response when mismatch is added, but no shaping is performed. In this case, the element selection is performed using a simple thermometer code. In all cases, the standard deviation of the DAC mismatch is 3% of the unit element step-size. The performance of the two architectures is similar, with the vector shapers showing slightly higher performance over the tuning range as the DAC size increases.
Figure 4.26: SNR with 3% mismatch for 2-element vector & tree shapers

Figure 4.27: SNR with 3% mismatch for 4-element vector & tree shapers
Figure 4.28: SNR with 3% mismatch for 8-element vector & tree shapers

Figure 4.29: SNR with 3% mismatch for 16-element vector & tree shapers
Figure 4.30: SNR with 3% mismatch for 32-element vector & tree shapers

Figure 4.31: SNR with 3% mismatch for 64-element vector & tree shapers
Figure 4.32: SNR with 3% mismatch for 128-element vector & tree shapers

**Hardware Complexity**

Figures 4.33 and 4.34 compare the hardware complexity of the corresponding tree-structured architecture to that of the vector mismatch shaper described in Chapter 3. The critical path of the vector mismatch shaper grows much faster than that of all the tree-structured mismatch shapers, making the vector shaper unsuitable for high-speed applications with large DAC sizes. For lower DAC sizes, the cell area of the vector shaper is only slightly higher than that of the polar tree architecture, but starts to grow as the DAC size increases.
Figure 4.33: Comparison of tree and vector critical path delays

Figure 4.34: Comparison of tree and vector hardware complexity
4.5 Summary

The tree-structured mismatch shaper is a well-known technique for achieving higher-order mismatch noise shaping, and has been used for shaping mismatch noise away from signal bands at DC [15] and $F_S/4$ [44]. In this chapter, the tree-structured mismatch shaper is extended to allow the signal band to operate at an arbitrary center frequency. This is achieved by using a tree of tunable sequence generators, each of which employs a tunable loop filter to create a noise-shaped suppression band located at an arbitrary frequency in the tuning range. The tunable loop filters are retimed in two ways to improve the critical path. Both early- and late-retimed versions are used within the same tree, but at different depths.

The tunable $2^{nd}$-order loop filter for $1^{st}$-order noise shaping is implemented using both a CRFB structure and a polar structure. The tunable $4^{th}$-order loop filter for $2^{nd}$-order noise shaping is implemented using a CRFB structure. All three tunable architectures have been implemented in VHDL for DAC sizes of 2, 4, 8, 16, 32, 64 and 128 unit-elements. The designs have been synthesized using a 0.18$\mu$m CMOS technology, running at a nominal voltage of 1.8V. Synthesis results show that of the three architectures, the polar structure has the smallest cell area for all DAC sizes. The polar structure also has the shortest critical path delays for smaller DAC sizes. However, for larger DAC sizes the critical path delays are similar to the rest.
The tree-structured architectures have also been compared to the vector shaper from Chapter 3. The cell areas of the 1st-order tree shapers are comparable to those of the 1st-order vector shaper. However, the critical path delays of the tree structures are all much shorter than the vector shaper.

The designs have been simulated with data generated from a tunable 6th-order ∆Σ modulator. An oversampling ratio of 64 allows coverage of the entire Nyquist band using only 64 tuning settings. Mismatch errors have been modeled by uniformly distributed random variations, with standard deviations of 1%-3% with respect to the DAC unit step size. Simulation results show that all three mismatch shaper structures consistently improve performance for center frequencies across the entire tuning range. As would be expected, the two 1st-order structures show similar performance trends, while the 2nd-order structure shows far better shaping performance. When compared to the 1st-order vector shaper from Chapter 3, the 1st-order tree shapers show similar performance for smaller DAC sizes. For larger DAC sizes, the vector shaper shows slightly better performance.

In general, the 1st-order tree and vector structures exhibit similar shaping performance over the entire tuning range. However, the tree structure exhibits much shorter critical path delays as compared to the vector shaper, while requiring slightly lower hardware complexity. Finally, when hardware complexity is not of great concern, much higher performance can be achieved by using the tunable 2nd-order tree shaper.
Chapter 5

Tunable N-Path Mismatch Shaping

Mismatch noise shaping is a commonly used technique for shaping the noise created by mismatch errors in over-sampled digital-to-analog converters (DAC) [1][18]. Most mismatch shaping techniques reported operate with the signal band located at a fixed frequency [2]. The vector shaper [13] and the tree-based approach [15] can each be modified in order to allow the capability to arbitrarily tune the mismatch shaper center frequency. However, these techniques require relatively complex hardware to add tunable operation.

This chapter presents a relatively simple architecture based on the N-path principle for tuning the center frequency of the mismatch shaping suppression band. Details of hardware implementation and complexity are provided. Results from simulations show a consistently improved signal-to-noise ratio at tuning frequency settings across the entire Nyquist bandwidth.
5.1 Introduction

The concept of software-defined radio is now a reality as radio front-ends become more adept at managing multiple modulation standards and signal-bandwidths at various intermediate frequencies (IF). This has created the necessity to directly synthesize the IF signal within the digital domain [6][7][8]. However, realizing these systems within the digital domain requires a bandpass DAC with a fully programmable signal band location [5][9][10][11]. Bandpass Delta-Sigma (ΔΣ) data converters are widely used in applications where high linearity is required over a narrow bandwidth [1][2][3]. The high signal-to-noise ratio (SNR) is achieved at the expense of much higher quantization noise out of band. This noise-shaped characteristic directly follows the noise-transfer function (NTF) of the ΔΣ modulator itself.

The linearity of a multi-bit ΔΣ DAC can be severely hampered by DAC element mismatch error, which is an unavoidable side-effect of modern fabrication processes. As a result, many attempts have been made to eliminate or alleviate this nonlinearity in the DAC output, including trimming, calibration, digital correction, and mismatch shaping [1]. Of these techniques, mismatch shaping offers the most promising results for achieving the highest levels of linearity over the signal band [2]. Most mismatch shaping techniques strive to re-arrange the individual DAC unit-element controls in such a way that the noise power within the signal band is suppressed while the signal content is kept unchanged. Unfortunately, most mismatch shaping techniques employed thus far are effective only over a fixed signal band location [1][2].
The vector shaper [13] and the tree-based approach [15] are two exceptions that can each be modified and extended in such a way so as to allow tunable operation. However, each of these techniques requires relatively complex hardware to add the capability to arbitrarily tune the mismatch shaper center frequency in order to follow the signal-band of a tunable ∆Σ DAC. A novel approach to tunable mismatch shaping can be achieved by applying the $N$-path principle [38] and varying the value of $N$ according to the tuning frequency.

This chapter describes multiple architectures of a technique based on the $N$-path principle that enables the center frequency of the mismatch-shaping suppression band to be tunable over the full tuning range of a bandpass ∆Σ modulator. This allows a tunable bandpass ∆Σ modulator to take full advantage of multi-bit quantization. Section 5.2 describes the proposed tunable $N$-path technique. Section 5.3 provides implementation details of the prototype filter architectures employed within the tunable $N$-path mismatch shaper. Section 5.4 presents the simulated performance and hardware complexity of each of these proposed architectures. In Section 5.5, the proposed $N$-path architectures are compared to other techniques for tunable mismatch shaping. The chapter is summarized and concluded in Section 5.6.
5.2 Tunable \(N\)-path Technique

Most reported bandpass \(\Delta \Sigma\) data-converters operate with a fixed location of the signal band. The center frequency of the signal band is usually placed at \(F_S/4\), where \(F_S\) is the sample rate of the data converter. Centering the signal band at \(F_S/4\) provides the maximum distance between aliased copies of the signal band and this results in easier design restrictions on the analog anti-aliasing filters. Centering the signal-band at \(F_S/4\) also simplifies the necessary computation involved in quadrature modulation and demodulation [6]. In the context of mismatch shaping, a bandpass \(\Delta \Sigma\) data-converter with a signal band located at a fixed center frequency allows the use of a bandpass mismatch shaper with a mismatch suppression band located at a fixed center frequency. The choice of locating the signal-band specifically at \(F_S/4\) also allows the application of the \(N\)-path filter principle when \(N\) is chosen as 2 or 4 [38][39][40], as explained in Section 5.2.1.

5.2.1 \(N\)-path Filter Principle

The \(N\)-path transform is a well-known technique for compressing and replicating a given prototype filter’s transfer function multiple times around the unit circle in the \(z\)-plane, and is defined by: \(z \rightarrow z^N\). This transform can be realized by performing the polyphase decomposition of the DAC input data stream \(X(z)\):

\[
X(z) = \sum_{k=0}^{N-1} z^{-k}G_k(z^N) \tag{5.1}
\]
Each polyphase component $G_k(z)$ encounters a prototype filter $H(z)$, identical to all polyphase paths, such that $Q_k(z) = G_k(z)H(z)$. The filtered polyphase components $Q_k(z)$ are then recombined to form:

$$Y(z) = \sum_{k=0}^{N-1} z^{-k}Q_k(z^N)$$  \hspace{1cm} (5.2)

$$= \sum_{k=0}^{N-1} z^{-k}G_k(z^N)H(z^N)$$  \hspace{1cm} (5.3)

$$Y(z) = X(z)H(z^N).$$  \hspace{1cm} (5.4)

The result is that a higher-order transfer function $H(z^N)$ can be realized using $N$ copies of the lower-order function $H(z)$. The implementation of this structure is trivial once it is viewed as a commutator model, as shown in Figure 5.1.

Specific transfer functions respond particularly well to the application of the $N$-path transform. For example, a first-order highpass mismatch shaping prototype filter $H(z) = 1 - z^{-1}$ has a zero at $z = 1$ and a pole at $z = 0$. This mismatch-shaping response is ideally suited to operation with a signal band location of $z = 1$ (DC). Applying the 4-path transform $z \rightarrow z^4$ to this prototype shaper results in the aggregate response of $H(z) = 1 - z^{-4}$, with zeros located at $z = \pm 1, z = \pm j$ and all poles at $z = 0$. These zero locations provide
mismatch suppression at the center frequencies $F_c = 1, F_S/4, F_S/2$. This 4-path technique can be used for mismatch-shaping the DAC in a bandpass $\Delta\Sigma$ converter operating at $F_S/4$ [38][39].

An improvement to this approach is described in [40], where instead of using a 4-path transform $z \rightarrow z^4$ on a first-order highpass mismatch shaping response $H(z) = 1 - z^{-1}$, a 2-path transform $z \rightarrow z^2$ is applied to a lowpass response $H(z) = 1 + z^{-1}$. This also replicates the mismatch shaping response with zeros at $z = \pm j$, but only a 2-path transform is required instead of the 4-path. The lower number of paths reduces the hardware complexity, as well as increases the mismatch shaping bandwidth.

5.2.2 Modified $N$-path for Tunable Operation

A tunable bandpass $\Delta\Sigma$ data converter requires the use of a tunable mismatch shaping transfer function in order to continue enjoying the benefits of using multi-bit quantization. The tunable $\Delta\Sigma$ data converters under consideration have signal-bands that can be arbitrarily centered at $F_k, k \in [0, OSR]$: 

$$F_k = \frac{k \cdot F_S}{OSR \cdot 2}, \quad k = 0, 1, 2, \ldots, OSR$$  \hspace{1cm} (5.5)

where $F_S$ and $OSR$ are the sampling rate and the oversampling ratio of the data converter, respectively. The center frequency $F_k$ is referred to as the tuning setting. Mismatch noise can only be suppressed at each tuning setting if the mismatch shaper transfer function zeros can be adjusted to a location at or near $F_k$.  

84
As described in Section 5.2.1, the \( N \)-path transform \( z \rightarrow z^N \) compresses and replicates a prototype filter \( H(z) \) a number of times \( N \) around the unit circle in the \( z \)-plane, producing the aggregate response of \( H(z^N) \). The heart of the proposed technique is to use the \( N \)-path principle with a variable value of \( N \), each of which yields a different set of mismatch shaping zero locations around the unit circle. Using a given value of \( N \), a bandpass \( \Delta \Sigma \) modulator can operate with a signal band located at any of the \( N \) replicas of the mismatch-shaping zeros. Conversely, mismatch shaping can be performed at each tuning setting \( F_k \) by selecting the value of \( N \) such that there are zeros near \( F_k \) available for mismatch noise suppression.

For example, Figure 5.2(a) shows the locations of the \( N \)-path zeros for \( N = 4 \) when a highpass mismatch shaper prototype with \( H(z) = 1 - z^{-1} \) is used. In order to suppress mismatch noise at frequencies that coincide with the zeros of \( H(z) = 1 - z^{-4} \), it is necessary to increase the number of paths from \( N = 4 \) to \( N = 8 \), as shown in Figure 5.2(b). Figure 5.3 shows the corresponding mismatch-noise frequency response.

It is evident from Figures 5.2 and 5.3 that increasing the number of paths \( N \) lowers the spacing between successive mismatch suppression bands around the unit circle. This provides denser coverage of signal band locations, thereby improving tuning granularity. However, this also reduces the impact of mismatch shaping by compressing the effective band of mismatch suppression. In fact, the prototype mismatch shaper undergoes band compression by a factor of \( N \).
Figure 5.2: $N$-path pole-zero locations for (a) $N = 4$ and (b) $N = 8$

Figure 5.3: Mismatch error-shaping when (a) $N = 4$ and (b) $N = 8$
As a result, the need to increase the tuning granularity of the mismatch shaper transfer function needs to be balanced with the desire to maintain a sufficiently effective band of suppression for the mismatch noise.

5.2.3 Prototype Diversity

A major advantage of the tunable $N$-path technique is the ability to use any mismatch-shaping algorithm as the prototype shaper. Most shapers can be used as long as band-compression does not significantly degrade performance. The locations of the $N$-path transformed zeros depend on both the number of paths $N$ as well as the locations of the zeros in the prototype itself. As described in Section 5.2.2, the prototype mismatch shaper experiences band compression by a factor of $N$, and this makes it very appealing to minimize the number of paths $N$. Therefore, for a fixed value of $N$, the only way to increase the tuning granularity is to increase the diversity of prototype mismatch shaper responses available for use, each of which can perform mismatch noise suppression in a different frequency band.

For example, a 3-path transform applied to a mismatch shaper prototype with a single zero produces three zeros equally-spaced around the unit circle, but the angular orientation of the zeros changes according to the initial location of the zero. Specifically, when the 3-path transform $z \rightarrow z^3$ is applied to a first-order highpass prototype $H_{HP}(z) = 1 - z^{-1}$, the resulting zeros are located at:

$$z = 1, -\frac{1}{2} \pm j \frac{\sqrt{3}}{2}$$  \hfill (5.6)
If instead a lowpass prototype $H_{LP}(z) = 1 + z^{-1}$ is used, the resulting zeros are located at:

$$z = -1, \frac{1}{2} \pm j \frac{\sqrt{3}}{2}$$

(5.7)

Figure 5.4(a) shows a 3-path transform applied to a lowpass mismatch shaper prototype of $H(z) = 1 + z^{-1}$, and Figure 5.4(b) shows a 3-path transform applied to a highpass mismatch shaper prototype of $H(z) = 1 - z^{-1}$. Achieving the same tuning granularity from just a single prototype (either $H(z) = 1 - z^{-1}$ or $H(z) = 1 + z^{-1}$) requires a 6-path transform. Therefore, tuning granularity is affected by both the number of paths $N$ and the different types of available prototype responses.

Figure 5.5 shows an example of the tunable $N$-path technique using only two 1st-order prototype shapers, one each with zeros at $z = 1$ and $z = -1$. The $\Delta\Sigma$ modulator frequency response is shown at various tuning frequencies
and plotted alongside the corresponding $N$-path mismatch shaper. The solid line shows the modulator response, and the dashed line shows the response of the $N$-path mismatch shaper. This example illustrates how the proper selection of the number of paths $N$ and the prototype shaper response can lead to alignment between the signal band of the tunable $\Delta\Sigma$ modulator and the spectral nulls in the $N$-path mismatch shaping response.

5.2.4 Proposed Tunable Structure

The proposed tunable $N$-path mismatch shaper can be realized by implementing the structure shown in Figure 5.6. The lookup table contains a list of tuning pairs $(N_k, p_k)$ each of which is indexed by the subscript $k$, and corresponds to the current tuning setting $F_k$. Each tuning pair consists of the appropriate combination of the number of paths $N_k$ and the prototype shaper response $p_k$, that yields the best mismatch suppression at or near the center frequency $F_k$.

The programmable mismatch shaper in Figure 5.6 conceptually implements the basic structure shown in Figure 5.1, where the number of active paths $N$ is adjustable and controlled by $N_k$. Each prototype mismatch shaper $H_p(z)$ resides on a single path and implements a finite set of $R$ shaping responses $p \in [1, R]$. The prototype response in active use is indicated by $p_k$. For example, in the case of a tunable $N$-path shaper with only two prototypes ($R = 2$) given by the $1^{st}$-order highpass and lowpass shaping responses, the set of shaping functions is limited to $\{1 - z^{-1}, 1 + z^{-1}\}$. 

89
Figure 5.5: Mismatch shaping response at various tuning frequencies
The primary advantage of the tunable $N$-path technique is the ability to extend a low complexity mismatch shaper to operate at multiple center frequencies without significantly adding to the implementation complexity. Thus, in order to justify the use of the tunable $N$-path technique, the complexity of the prototype must be kept lower than that of a fully tunable approach, such as the vector or tree-based techniques described in Chapter 3 and Chapter 4, respectively. Additionally, the process of switching between different prototype shaper responses should be made possible with relatively low overhead in complexity.

**First-Order Mismatch Shaping**

Tunable 1st-order mismatch shaping can be achieved by employing a technique like data-weighted averaging (DWA) [20][23]. to implement the prototype shaper response. Conventional DWA can be used to create a 1st-order highpass prototype response with a zero at $z = 1$ [38]. This response is easily converted to a 1st-order lowpass prototype response with a zero at $z = -1$ [40].
Implementation details of the tunable $N$-path structure employing the combined $1^{st}$-order DWA prototype shapers [16][55] are described in Section 5.3.1.

If the vector- or tree-based prototype mismatch shaping structures are used, the $1^{st}$-order shaping response can be implemented with zeros at either $z = 1$, $z = \pm j$ or $z = -1$. These are simply extensions of the highpass vector mismatch shaping technique described in [13], or the highpass tree-structured technique described in [15]. Further simplification can be achieved by noting that the prototype shaper response with zeros at $z = \pm j$ can be obtained from a 2-path transform of the prototype response with zeros at $z = -1$ [40].

Mismatch shaper responses with zeros located at other angles around the unit circle can be implemented using the full tunable architectures described in Chapter 3 and Chapter 4. However, the complexity of implementing these prototype responses approaches that of the full tunable architectures described in those chapters. Using these prototype shapers would increase the complexity to an extent that effectively exceeds the benefit provided by the tunable $N$-path technique in the first place.

Implementation details of the tunable $N$-path structure employing the combined $1^{st}$-order vector-structured prototype shapers are described in Section 5.3.2, and those of the combined $1^{st}$-order tree-structured shapers are described in Section 5.3.3.
Second-Order Mismatch Shaping

Data-weighted averaging with a $2^{nd}$-order mismatch shaping response can be implemented, but requires DAC elements with multiple analog output levels [14]. In this work, only DAC element with two analog output levels are considered. However, $2^{nd}$-order mismatch shaping responses using 2-level DAC elements can be implemented with vector- or tree-structured mismatch shaping architectures [13][15].

In general, $2^{nd}$-order mismatch shapers using the vector and tree structures tend to be more susceptible to band compression when used as prototype shapers in a tunable $N$-path system. This is because their mismatch transfer functions have poles located closer to the unit circle, instead of at the origin $z = 0$. This is done in order to reduce the chances of $\Delta\Sigma$ loop instability [2][53]. As the number of paths $N$ is increased, the poles move closer to the zeros (which lie on the unit circle) causing the mismatch noise spectrum to rise rapidly around the zeros (which are placed to create the mismatch suppression bands). This exacerbates band compression.

As a result of this sensitivity to band compression, the value of $N_{MAX}$ must be kept relatively low when using $2^{nd}$-order prototype shapers. A lower value of $N_{MAX}$ drastically reduces the achievable tuning granularity, effectively rendering the approach unsuitable for tunable applications. However, good mismatch suppression can still be achieved if the desired location of the signal band happens to fall at or near an $N$-path replica zero.
5.3 Prototype Implementation

This section describes three architectures for implementing the tunable $N$-path technique, using three different mismatch shaping algorithms. All three structures implement 1\textsuperscript{st}-order prototype mismatch shaping responses of $H(z) = 1 - z^{-1}$ or $H(z) = 1 + z^{-1}$, using the same basic mismatch shaper. Section 5.3.1 describes the operation of the tunable $N$-path approach using a DWA prototype mismatch shaper. Section 5.3.2 describes a prototype shaper based on the vector mismatch shaper described in Chapter 3, and Section 5.3.3 describes a prototype shaper based on the tree-structured mismatch shaper from Chapter 4.

The combined mismatch shaper prototype can be implemented in hardware by realizing the conceptual commutator structure shown in Figure 5.1, where the number of active paths $N$ can be adjusted. However, it is more efficient to apply a literal interpretation of the $N$-path transform $z \rightarrow z^N$: every single delay element, denoted in the $Z$-domain as $z^{-1}$, is replaced with a series of delay elements $z^{-N}$, each with length $N$. This essentially translates to a \textit{variable-length shift register} whose length is controlled by $N$.

Each setting of the signal-band center frequency requires a corresponding tuning pair consisting of the number of paths $N$ and a prototype selection control $p$ indicating the type of prototype shaper response to use. These tuning pairs $(N_k, p_k)$ can be stored in a small lookup-table, indexed by the $k$\textsuperscript{th} tuning frequency $F_k$, and referenced whenever the location of the signal band is updated.
Figure 5.7: N-path tuning pairs

For the example of 64 tuning settings, \( k \in [1, 64] \), the tuning pairs can be visualized as shown in Figure 5.7. The number of paths \( N_k \) is plotted against the frequency setting \( F_k \), with frequency normalized to \( 2\pi \). Each point is marked to indicate which prototype \( p_k \) (highpass or lowpass) is in use at that frequency setting \( F_k \).
5.3.1 DWA Prototype

Both highpass and lowpass 1st-order prototype shapers can be implemented using a simple element rotation technique. The highpass response can be realized by implementing the Data-Weighted Averaging (DWA) algorithm [23]. The lowpass prototype can in turn be implemented using a modified version of the DWA algorithm [40].

Figure 5.8 shows the architecture of a mismatch shaper based on element rotation. The quantized and binary-encoded data is converted to a thermometer code in order to drive the individual unit elements of the DAC. The pointer update block uses the input data to compute a new shift pointer. This pointer is used to perform a circular shift of the thermometer encode DAC data before driving the array of DAC unit-elements. The mismatch shaping process does not change the DAC data, which is the actual number of unit-elements enabled at any point. In fact, mismatch shaping is achieved by re-arranging the order in which the DAC unit-elements are used.

The $W$-bit binary-encoded DAC output data word needs to be converted to an $(2^W - 1)$-bit thermometer encoded word, in order to drive the
DAC unit elements. This can be performed by a Binary to Thermometer Converter such as a tree of multiplexers controlled by the binary input word, as shown in Figure 5.9 for $W = 3$.

Before controlling the DAC unit-elements, the $(2^W-1)$-bit thermometer-encoded DAC data needs to be circularly shifted by an amount equal to the $W$-bit shift pointer. This can be accomplished by using a barrel shifter connected in a circular fashion. An example is shown in Figure 5.10, with $W = 3$. 

97
The pointer update unit takes the DAC input data and produces a sequence of pointer values, each of which determines the number of places by which to rotate the thermometer-encoded DAC value. Different pointer update sequences are required for producing highpass and lowpass mismatch shaping responses.

The highpass prototype mismatch shaping transfer function \( H_{HP}(z) = 1 - z^{-1} \) is shown in Figure 5.11(a). This can easily be realized by implementing the Data-Weighted Averaging (DWA) algorithm [1]. As an example, the unit-element selection pattern resulting from an example data sequence is shown in Figure 5.11(b). The selection pattern can be generated by employing a simple modulo (overflowing) accumulator.

The lowpass prototype mismatch shaping transfer function \( H_{LP}(z) = 1 + z^{-1} \) is shown in Figure 5.12(a), and can be implemented using the algorithm described in [40]. The direction of rotation is reversed every sample. Figure 5.12(b) shows the unit-element selection pattern resulting from the the same data sequence shown in Figure 5.11(b). The hardware necessary to implement this modified element rotation algorithm is only slightly more complicated than the highpass prototype.

The above two prototype mismatch shaping functions can be combined such that a single selection signal is used for choosing between the two functions of the pointer update block. An example of such a hybrid structure is shown in Figure 5.13.
Figure 5.11: highpass (a) response and (b) rotation example

Figure 5.12: lowpass (a) response and (b) rotation example
The control signal $p_k$ for selecting between the two prototype transfer functions is defined as:

$$
p_k = \begin{cases} 
0 & \text{for } H_{LP}(z) = 1 + z^{-1} \text{ prototype} \\
1 & \text{for } H_{HP}(z) = 1 - z^{-1} \text{ prototype}
\end{cases} \quad (5.8)
$$

If the number of DAC levels is not a power of 2, the overflowing accumulator needs to be replaced with a modulo subtraction unit.

### 5.3.2 Vector Prototype

The structure in Figure 5.6 can be realized using the vector mismatch shaper described in Chapter 3 to implement the highpass $H_{HP}(z)$ and low-pass $H_{LP}(z)$ prototype shaping responses. Figure 5.14 shows the structure of the vector shaper including an array of identical loop filters each of which determines the overall mismatch noise transfer function. As described in Section 3.2, it is more efficient to implement the modified vector shaper structure shown in Figure 3.3. When used within the tunable $N$-path structure, the modified vector shaper is shown in Figure 5.15. The $N$-path transform $z \rightarrow z^N$ is applied to the loop filter to give $\text{NTF}(z) \rightarrow \text{NTF}(z^N)$. 

![Figure 5.13: 1st-order hybrid pointer update structure](image)
The filter is selected from highpass $H_{HP}(z^N)$ and lowpass $H_{LP}(z^N)$ prototypes such that:

$$NTF_P(z^N) = \begin{cases} H_{LP}(z^N) & \text{when } p_k = 0 \\ H_{HP}(z^N) & \text{when } p_k = 1 \end{cases}$$

(5.9)

The $N$-path transformation can be implemented by replacing every single delay element $z^{-1}$ with a variable-length shift register $z^{-N}$ of length $N$. The noise-transfer function of the 1st-order highpass $H_{HP}(z^1)$ prototype is given by:

$$H_{HP}(z) = \frac{1 - z^{-1}}{1 + d \cdot z^{-1}}$$

(5.10)

This has a zero at $z = 1$ and a pole at $z = -d$, where $0 < d < 1$. The constant $d$ is included in order to push the pole further away from the zero at $z = 1$. This widens the mismatch suppression band to alleviate the effects of band compression as $N$ is increased. However, increasing $d$ also increases the value of $max|NTF(z)|$, which can threaten the stability of the $\Delta\Sigma$ mismatch shaping loop (refer to Section 4.2.2 for a discussion on this issue).
Figure 5.15: N-path vector shaper prototype

The corresponding noise-transfer function of the 1\textsuperscript{st}-order lowpass $H_{LP}(z)$ prototype is given by:

$$H_{LP}(z) = \frac{1 + z^{-1}}{1 - d \cdot z^{-1}}$$  \hspace{1cm} (5.11)

The noise-transfer functions of the highpass and lowpass prototype responses can be generalized by:

$$NTF_P(z) = \frac{1 + c \cdot z^{-1}}{1 - c \cdot d \cdot z^{-1}}$$  \hspace{1cm} (5.12)

where the coefficient $c$ is defined as $c = +1$ when $p_k = 0$ (lowpass prototype) and $c = -1$ when $p_k = 1$ (highpass prototype). This general form can be implemented using the structure shown in Figure 5.16. There are $M$ copies of this, as required by the vector shaper.
The coefficient $c$ only takes on values from $\{+1, -1\}$, which eliminates the need for a multiplier. The unit-element DAC control $v_k[n]$ is a single-bit value, also eliminating the need for a multiplier to compute $(1 + d) \times v_k[n]$. A multiplier is required for $(1 + d) \times s[n]$, but this computation is shared across all copies of the loop filter because the same minimum value $s[k]$ is distributed by the vector quantizer to all loop filters, as shown in Figure 5.15. In this design, the value of the constant $d = 0.390625$ and the datapath width is $PW = 8$ bits. The value of $d$ is empirically chosen using computer simulations to provide a good trade-off between expected loop stability and mismatch suppression performance.
5.3.3 Tree Prototype

The tree-structured mismatch shaper described in [15] can be used as the prototype mismatch shaper in a tunable $N$-path architecture. The tree-structured mismatch noise shaper performs element selection through the use of a tree of switching blocks. Figure 5.17 shows an example of an 8-element DAC that uses the tree structure to perform element selection. Each of the blocks labeled $S_k, r$ is a switching block that routes the input data in two possible directions. There are $\log_2 M$ layers of switching blocks, and each switching block in the final layer drives out a pair of single-bit values to control a pair of DAC elements.
Each switching block operates according to:

\[ x_{k-1,2r-1}[n] = \frac{1}{2}(x_{k,r}[n] + s_{k,r}[n]) \quad (5.13) \]

\[ x_{k-1,2r}[n] = \frac{1}{2}(x_{k,r}[n] - s_{k,r}[n]) \quad (5.14) \]

where \( s_{k,r}[n] \) is a switching sequence generated within each switching block. The spectral properties of the mismatch error sequence at the output of the DAC are determined by those of switching sequence [15]. Therefore, in order to create an \( L^{th} \)-order noise-shaped DAC mismatch error sequence, with the noise suppression band centered at \( F_c \), it is necessary to create a switching sequence \( s_{k,r}[n] \) as an \( L^{th} \)-order noise-shaped sequence, also with a suppression band centered at \( F_c \) [17].

In the context of the tunable \( N \)-path structure, a pair of highpass \( H_{HP}(z) \) and lowpass \( H_{LP}(z) \) prototype shapers can be created using the tree structure by allowing the sequence generators to place the suppression bands at the corresponding locations of \( F_c \). Since the sequence generator creates the noise-shaped sequence using a \( \Delta \Sigma \) modulator whose spectral characteristics are controlled by a loop filter [15], the locations of the suppression band can be tuned by simultaneously controlling the loop filters of all sequence generators in the tree.

Figure 5.18 shows the tree structure used in a tunable \( N \)-path architecture. Each switching block receives the tuning pair \((N_k, p_k)\), which controls the location of the suppression band. Figure 5.19 shows the structure of the sequence generator using a \( \Delta \Sigma \) modulator loop. The loop filter directly receives
the tuning pair \((N_k, p_k)\) in order to determine the location of the noise-shaped suppression band. The loop filter is transformed by the \(N\)-path transform \(z \rightarrow z^N\) to give \(NTF(z) \rightarrow NTF(z^N)\), and the filter prototype is selected as follows:

\[
NTF_P(z^N) = \begin{cases} 
H_{LP}(z^N) & \text{when } p_k = 0 \\
H_{HP}(z^N) & \text{when } p_k = 1 
\end{cases} \tag{5.15}
\]

The highpass \(H_{HP}(z) = 1 - z^{-1}\) and lowpass \(H_{LP}(z) = 1 + z^{-1}\) responses are selected by \(p_k\), which forms part of the tuning pair \((N_k, p_k)\). The \(N\)-path transformation can be implemented by replacing every single delay element \(z^{-1}\) with a variable-length shift register \(z^{-N}\) of length \(N\).
The noise-transfer functions of these prototype responses can be generalized by:

\[ NTF_P(z^N) = 1 + c \cdot z^{-N} \]  

(5.16)

where the coefficient \( c \) is defined as \( c = +1 \) when \( p_k = 0 \) (lowpass prototype) and \( c = -1 \) when \( p_k = 1 \) (highpass prototype). The loop filter is implemented by the structure shown in Figure 5.20. The coefficient \( c \) only takes on values from \{+1, -1\}, which effectively eliminates the need for a multiplier.

The loop filter output \( y[n] \) is quantized and limited before creating the output sequence, as shown in Figure 5.20. The single-bit quantizer produces two different outputs: one for creating the sequence output \( s_{k,r}[n] \), and another for creating the internal feedback sequence \( s_{fb}[n] \). For both outputs, the quantizer produces +1 when \( y[n] > 0 \), and -1 when \( y[n] < 0 \). In the special case of when the loop filter output \( y[n] \) is zero, the quantizer randomly selects a value from \{+1, -1\} for the output sequence path, and \{+0.5, -0.5\} for the feedback path. Pseudo-random selection is controlled using a linear-feedback shift register (LFSR) [45].
Figure 5.20: Sequence generator detail for N-path tree prototype

The limiter block ensures that parity is maintained between the sequence output and the input data $x_{k,r}[n]$ to this particular switching block. This is done by performing a logical AND of the quantizer output and the least-significant bit $x_{LSB}$ of the input data $x_{k,r}[n]$. The output sequence $s_{k,r}[n]$ and the internal feedback sequence $s_{fb}[n]$ can be defined as:

$$s_{k,r}[n] = \begin{cases} 
+ x_{LSB} & \text{when } y > 0 \\
- x_{LSB} & \text{when } y < 0 \\
\pm x_{LSB} & \text{when } y = 0 
\end{cases} \quad (5.17)$$

$$s_{fb}[n] = \begin{cases} 
+ x_{LSB} & \text{when } y > 0 \\
- x_{LSB} & \text{when } y < 0 \\
\pm \frac{x_{LSB}}{2} & \text{when } y = 0 
\end{cases} \quad (5.18)$$
5.4 Comparison of Tunable $N$-path Architectures

In this section, the mismatch shaping performance and hardware complexity of the three $N$-path prototype architectures described in Section 5.3 are compared. Tunable $N$-path architectures using the DWA, vector and tree prototypes are implemented in VHDL and simulated using data generated by the tunable bandpass $\Delta\Sigma$ modulator specified in Table 5.1. An oversampling ratio of 64 allows coverage of the entire Nyquist band using 64 tuning settings. DAC errors are modeled as uniformly distributed random mismatch, with the standard deviation expressed as a percentage of the unit element value.

Table 5.1: Tunable bandpass $\Delta\Sigma$ modulator specifications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Modulator order</td>
<td>6</td>
</tr>
<tr>
<td>OSR</td>
<td>64</td>
</tr>
<tr>
<td>Quantizer levels</td>
<td>variable</td>
</tr>
<tr>
<td>Tuning settings</td>
<td>64</td>
</tr>
<tr>
<td>Normalized signal bandwidth</td>
<td>0.0078</td>
</tr>
</tbody>
</table>

5.4.1 Shaping Performance

Figures 5.21—5.27 compare the simulated SNR for the three $N$-path prototype shapers using DAC sizes of 2, 4, 8, 16, 32, 64 and 128 unit elements, respectively. In each figure, the SNR with no mismatch is that of a DAC without any mismatch errors, where the performance is limited by the tunable bandpass $\Delta\Sigma$ modulator specified in Table 5.1. The unshaped SNR is the response when mismatch is added, but no shaping is performed. The shaped SNR is the response when mismatch is added and mismatch shaping
Figure 5.21: SNR with 3% mismatch for 2-element $N$-path shapers

is performed using the corresponding shaper. Each data-point corresponds to
the average SNR computed for a thousand different randomly-generated DAC
mismatch configurations, at each tuning frequency setting. The standard de-
viation of the mismatch error is 3%.

All three tunable $N$-path architectures exhibit similar performance over
the tuning range, as the SNR is constrained not by the prototype shaper but by
the limited tuning granularity and excessive band compression. It is notable
that the tunable $N$-path technique becomes less effective as the DAC size
increases.
Figure 5.22: SNR with 3% mismatch for 4-element N-path shapers

Figure 5.23: SNR with 3% mismatch for 8-element N-path shapers
Figure 5.24: SNR with 3% mismatch for 16-element $N$-path shapers

Figure 5.25: SNR with 3% mismatch for 32-element $N$-path shapers
Figure 5.26: SNR with 3% mismatch for 64-element $N$-path shapers

Figure 5.27: SNR with 3% mismatch for 128-element $N$-path shapers
5.4.2 Hardware Complexity

The three tunable $N$-path architectures have been implemented using VHDL and tested for several DAC sizes. The designs for each DAC precision have been synthesized using Synopsys Design Compiler [50] under the conditions specified in Table 5.2.

Table 5.2: Hardware synthesis conditions

<table>
<thead>
<tr>
<th>Process</th>
<th>0.18 µm CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device corners</td>
<td>slow-NMOS / slow-PMOS</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.62 V</td>
</tr>
<tr>
<td>Temperature</td>
<td>125°C</td>
</tr>
</tbody>
</table>

Figures 5.28 and 5.29 show the synthesis results for each DAC size. For each structure, the critical path delay in nano-seconds (ns) is plotted against DAC size in Figure 5.28. The critical path of the vector prototype shaper grows much faster than that of the tree-structured or DWA prototype mismatch shapers. This is primarily due to the rapidly growing path delay through the vector quantizer, as the DAC size increase [13]. The critical path of the DWA prototype is by far the shortest, making it very suitable for high-speed applications.

Figure 5.29 shows the cumulative cell area corresponding to the points in Figure 5.28. The area is calculated in square-microns ($\mu m^2$) and plotted on a logarithmic scale. The tunable $N$-path shaper with vector prototype occupies the largest area for DAC sizes. This is primarily due to the vector quantizer and sorting network. The tree-structured prototype shaper exhibits
Figure 5.28: Comparison of $N$-path critical path delays

Figure 5.29: Comparison of $N$-path hardware complexity
a similar rate of area growth with increasing DAC size, but is continues to remain much smaller than the vector approach. The architecture using the DWA prototype is by far the smallest, and exhibits the slowest rate of area growth with increasing DAC size.

5.5 Comparison with Tunable Tree Structure

In this section, the mismatch shaping performance and hardware complexity of the \( N \)-path architectures described in Section 5.3 are compared to the tunable tree structure described in Chapter 4. The specific tunable tree structure compared is the 2\(^{nd}\)-order polar architecture for 1\(^{st}\)-order mismatch shaping structure described in Section 4.2.4. The tunable \( N \)-path architectures in the comparison use the DWA and tree-structured prototype shapers described in Section 5.3.1 and Section 5.3.3, respectively. All architectures are implemented in VHDL and simulated using data generated by the tunable bandpass \( \Delta \Sigma \) modulator specified in Table 5.1. DAC errors are modeled as uniformly distributed random mismatch, with the standard deviation expressed as a percentage of the DAC unit element value.

5.5.1 Shaping Performance

Figures 5.30–5.36 compare the simulated SNR of three tunable mismatch shapers using DAC sizes of 2, 4, 8, 16, 32, 64 and 128 unit elements, respectively. In each figure, the SNR with no mismatch is that of a DAC without any mismatch errors, where the performance is limited just by the
Figure 5.30: SNR with 3% mismatch for 2-element $N$-path & tree shapers tunable bandpass $\Delta\Sigma$ modulator specified in Table 5.1. The unshaped SNR is the response when mismatch is added, but no shaping is performed. The shaped SNR is the response when mismatch is added and mismatch shaping is performed using the corresponding shaper.

Each data-point corresponds to the average SNR computed for a thousand different randomly-generated DAC mismatch configurations, at each tuning frequency setting. The standard deviation of the mismatch error is 3%. The performance figures for 1% and 2% mismatch are similar to the case of 3% mismatch, and these figures are included in Appendix 1. For all DAC sizes, the tunable tree-based architecture clearly exhibits the best performance with consistency over the entire tuning range.
Figure 5.31: SNR with 3% mismatch for 4-element $N$-path & tree shapers

Figure 5.32: SNR with 3% mismatch for 8-element $N$-path & tree shapers
Figure 5.33: SNR with 3% mismatch for 16-element $N$-path & tree shapers

Figure 5.34: SNR with 3% mismatch for 32-element $N$-path & tree shapers
Figure 5.35: SNR with 3% mismatch for 64-element $N$-path & tree shapers

Figure 5.36: SNR with 3% mismatch for 128-element $N$-path & tree shapers
Figure 5.37: Comparison of $N$-path and tree critical path delays

### 5.5.2 Hardware Complexity

Figures 5.37 and 5.38 compare the hardware complexity of two $N$-path prototype architectures to the tunable tree-structured mismatch shaper referenced above. Figure 5.37 shows the critical path delay in *nano-seconds* (*ns*) plotted against DAC size. The critical paths of both $N$-path mismatch shapers increase much slower than the tunable tree-structure using the polar architecture. The critical path of the $N$-path mismatch shaper using the DWA prototype shows the lowest critical path delay.
Figure 5.38: Comparison of $N$-path and tree hardware complexity

Figure 5.38 shows the cell area corresponding to the points in Figure 5.37. The area is calculated in square-microns ($\mu m^2$) and plotted on a logarithmic scale. The tunable $N$-path shaper with DWA prototype occupies by far the smallest cell area, for all DAC sizes, and exhibits the slowest rate of area growth with increasing DAC size.

Both the $N$-path tree shaper and the tunable polar tree shaper show similar cell areas over all DAC sizes. This appears to be counter-intuitive as the $N$-path tree shaper is expected to be more area efficient due to fewer multipliers needed in the loop filter data-paths.
Figures 5.39 and 5.40 show the corresponding cell areas of combinational and non-combinational logic plotted versus DAC size. Non-combinational logic essentially comprises all delay elements and registers in the design. Combinational logic includes everything else. The tunable tree polar architecture consistently exhibits higher combinational area across all DAC sizes, as shown in Figure 5.39. As expected, this is due to the multipliers present in the tunable loop filter data-paths.

However, Figure 5.40 shows that the tunable $N$-path tree structure exhibits much higher non-combinational area across all DAC sizes, increasing at a slightly higher rate than the tunable tree polar architecture. This can be attributed to the fact that in any tree structure, increasing DAC size leads to wider data-paths and greater tree depths.

The tunable $N$-path structure requires each delay element $z^{-1}$ be replaced by a variable-length shift register $z^{-N}$, with a maximum length of $N_{max}$. When a tree structure is used as a prototype with the tunable $N$-path architecture, the number of registers greatly increases. Therefore, the area savings from the multipliers (due to less complex loop filter implementations) are offset by the vastly increased numbers of registers (due to the additional $N$-path shift registers).
Figure 5.39: Comparison of combinational logic area

Figure 5.40: Comparison of non-combinational logic area
5.6 Summary

The $N$-path principle is an existing technique for transforming a simple prototype shaper that operates at $DC$ to instead operate at $F_s/4$ [38][40]. This chapter presents a new technique for implementing a tunable mismatch shaper based on applying the $N$-path transform to a small set of prototype shapers and choosing the value of $N$ according to the desired tuning frequency. This technique is marked by implementation simplicity and flexibility, as it allows the use of any structure for implementing the prototype shaper responses.

Each tuning frequency setting requires prior knowledge of the best tuning pair that produces the highest level of mismatch suppression at that setting. These tuning pairs are pre-computed and stored in a small lookup-table or ROM. Three different tunable $N$-path architectures have been designed and implemented in VHDL for DAC sizes of 2, 4, 8, 16, 32, 64 and 128 unit-elements. Each architecture requires a pair of unique prototype responses: a highpass and a lowpass response. The first $N$-path architecture uses the DWA [23] and modified DWA [40] algorithms to realize the two prototype responses. The second and third $N$-path architectures respectively realize the prototypes using the tunable vector (Chapter 3) and tunable tree (Chapter 4) shapers, each designed for just a pair of tuned settings that correspond to the required highpass and lowpass responses.

All designs have been synthesized using a 0.18$\mu$m CMOS technology, running at a nominal voltage of 1.8V. Synthesis results show that of the three architectures, the DWA prototype structure exhibits the smallest cell area as
well as the shortest critical path delay, for all DAC sizes. The designs have been simulated with data generated from a tunable 6th-order ∆Σ modulator. An oversampling ratio of 64 allows coverage of the entire Nyquist band using only 64 tuning settings. Mismatch errors have been modeled by uniformly distributed random variations, with standard deviations of 1-3% with respect to the DAC unit step size. Simulation results show that all three mismatch shaper structures consistently improve performance for center frequencies across the entire tuning range, with similar mismatch noise suppression performance.

The \( N \)-path DWA and tree-structured architectures have also been compared to the tunable polar tree-structured 1st-order mismatch shaper from Chapter 4. The cell areas of the both tunable polar and \( N \)-path tree shapers are virtually identical, while that of the DWA \( N \)-path structure is the smallest. The critical path delays of the DWA \( N \)-path structure are consistently lower than all other architectures, sometimes by wide margins. However, the simulated mismatch shaping performance of all tunable \( N \)-path architectures is noticeably lower than the tunable polar tree structure.

These results show that the tunable \( N \)-path structure with DWA prototype is an effective approach to high-speed mismatch shaping where lower performance is sufficient. If higher performance is desired, the tunable tree architectures from Chapter 4 may be used, with the added cost of hardware complexity and slower speeds of operation.
Chapter 6

Tunable Quadrature Mismatch Shaping

Quadrature bandpass $\Delta \Sigma$ data converters are widely used in low-IF transceiver applications where high linearity is required over a narrow bandwidth. The pair of feedback DACs for the in-phase and quadrature paths can be combined into a single complex-valued DAC in order to eliminate the quadrature path mismatch through these blocks. Existing mismatch shaping techniques for complex-valued DACs require the signal band be located at a fixed frequency.

This chapter presents an architecture for quadrature bandpass mismatch shaping that allows the center frequency of the mismatch suppression band to be tunable over the entire Nyquist range [17]. The approach is based on the tunable tree-structured mismatch shaper described in Chapter 4, and extends the complex-valued tree-based mismatch shaper from [44] to allow tunable operation. The design has been implemented using VHDL and synthesized to logic gates. The hardware complexity and mismatch shaping performance of the proposed architecture are compared to that of a conventional approach using separate tunable mismatch shapers for each path.
In Section 6.1, the need for quadrature mismatch shaping is established. Section 6.2 provides an overview of quadrature bandpass mismatch shaping, along with previously reported techniques. Section 6.3 describes the proposed tunable quadrature mismatch shaping technique, and Section 6.4 presents the hardware implementation. Section 6.5 compares the mismatch shaping performance of the proposed architecture to that of a reference architecture, and Section 6.6 compares the implementation complexity for various DAC sizes. Finally, the chapter is concluded in Section 6.7.

6.1 Introduction

Modern wireless systems are facing a proliferation of wireless standards, making it necessary to maximize hardware reconfigurability in order to minimize costs. The continued scaling of CMOS technology has led to greater degrees of chip integration between the analog radio frequency (RF) front-ends and the digital signal processing (DSP) back-ends. As a result, it is far more attractive to process the intermediate frequency (IF) signal directly within the digital domain [56]. Due to their high linearity over narrow bandwidths, bandpass delta-sigma ($\Delta\Sigma$) modulators are rapidly becoming the data converter of choice for these applications [6][7][8][9][12].

In a typical low-IF receiver architecture, the RF signal is first demodulated into a complex IF signal, which consists of the in-phase ($I$) and quadrature ($Q$) component signals. These component signals are then individually digitized using a pair of bandpass $\Delta\Sigma$ modulators, as shown in Figure 6.1(a).
One complication that arises from this method is the potential for path mismatch between the $I$ and $Q$ channels, which leads to performance degradation [41][42]. This architecture can be improved by using a single complex-valued or quadrature bandpass (QBP) $\Delta \Sigma$ modulator, as shown in Figure 6.1(b). This not only reduces the path mismatch, but also lowers the hardware complexity by employing only a single complex loop filter in place of the pair of individual real-valued loop filters required for the previous approach. The corresponding separate and quadrature low-IF transmitter architectures are shown in Figures 6.2(a) and 6.2(b), respectively.
Higher-order $\Delta \Sigma$ analog-to-digital converters (ADC) that use multibit internal quantizers can provide a much higher signal-to-noise ratio (SNR) for a given oversampling ratio (OSR) [1]. However, dynamic range performance is limited due to distortion caused by device mismatch errors when the quantized signals emerge from the DAC embedded within the $\Delta \Sigma$ ADC feedback loop. In order to reduce these non-linearities in the DAC transfer function, the mismatch noise caused by the presence of these errors can be whitened or spectrally noise-shaped away from the signal band using a mismatch noise shaping DAC [1][13][57].
An important feature of data converters used in multi-standard wireless transceivers is the ability to place the center of the signal band at different frequencies within the Nyquist band. The need for this feature might arise out of requirements placed by the system design, or a desire to support multiple wireless standards. ∆Σ data converters can easily be designed with programmable loop filters in order to achieve this. However, ∆Σ modulators employing multi-bit quantization still require the mismatch noise to be removed or noise-shaped away from wherever the signal band has been placed. This can be achieved with the use of a tunable mismatch shaper [9][16][58].

In the case of a quadrature bandpass ∆Σ modulator employing a multi-bit quantizer, the center frequency of the suppression band in the mismatch transfer function (MTF) through a mismatch shaping complex DAC needs to be made tunable. This chapter extends previously reported mismatch shaping techniques to enable such a feature.

### 6.2 Overview of Mismatch Shaping

A unit-element DAC is commonly used in multibit ∆Σ systems. It consists of \( M \) unit-sized elements that can be combined to generate \( M + 1 \) different output levels, as shown in Figure 6.3. The element-selection logic (ESL) converts the binary DAC input \( x[n] \) to a vector of single-bit controls for the unit-element DAC:

\[
x[n] = \sum_{k=1}^{M} x_k[n], \quad x_k[n] \in \{0, 1\}
\]  

(6.1)
The DAC output $y[n]$ is similarly formed by summing the outputs of all the unit elements (where $\Delta$ denotes the nominal step-size of each element):

$$y[n] = \sum_{k=1}^{M} y_k[n], \quad y_k[n] = \Delta \cdot x_k[n]$$  \hspace{1cm} (6.2)

Component mismatch due to non-idealities in the manufacturing process leads to non-ideal values for the DAC elements. As a result, each DAC element exhibits errors in the output levels:

$$y_k[n] = \begin{cases} \Delta + \epsilon_{h_k} & \text{if } x_k[n] = 1 \\ \epsilon_{l_k} & \text{if } x_k[n] = 0 \end{cases}$$  \hspace{1cm} (6.3)

where $\epsilon_{h_k}$ and $\epsilon_{l_k}$ denote the static mismatch error when the DAC element is enabled and disabled, respectively. The overall DAC output is given by

$$y[n] = \alpha x[n] + \beta + \epsilon[n],$$  \hspace{1cm} (6.4)

where $\alpha$, $\beta$ and $\epsilon$ are the gain error, offset error and aggregate mismatch error, respectively, and all three quantities depend exclusively on the element mismatch errors [15].
Generally during element selection, there are multiple ways in which the input $x[n]$ can be used to select DAC elements to form the output, $y[n]$. The exceptions to this lie at the extremes: when $x[n] = 0$, no DAC elements are selected; when $x[n] = M$, all DAC elements are selected. However, the additional degrees of freedom available for the intervening input values can be exploited to vary the pattern of unit element selection in a way that spectrally shapes the mismatch error away from the signal band [13][15]. This is known as mismatch noise shaping, and is achieved without changing the actual number of selected DAC elements from the number that need to be activated in order to reproduce a given DAC input.

### 6.2.1 Quadrature Path Mismatch

In a quadrature bandpass $\Delta \Sigma$ modulator, both the $I$ and $Q$ components of the quantized complex signal require a DAC. In addition to the mismatch noise component $\epsilon[n]$, differences between mismatch errors in the two DACs results in path mismatch:

$$y_I[n] = \alpha_I x_I[n] + \beta_I + \epsilon_I[n] \quad (6.5)$$

$$y_Q[n] = \alpha_Q x_Q[n] + \beta_Q + \epsilon_Q[n] \quad (6.6)$$

Gain mismatch leads to folding from the negative frequency image band to the positive frequency signal band [43][44]. Since the complex $\Delta \Sigma$ modulator loop filter is asymmetric, there is little or no noise shaping in the image band. Therefore, gain mismatch will result in additional quantization noise appearing in the signal band.
Path mismatch can be reduced by combining the two real-valued DACs into a single mismatch shaping complex DAC, as shown in Figure 6.4. This allows both paths to use all available DAC elements in the process of shaping the mismatch noise. As a result, the average gain errors in the $I$ and $Q$ paths of the combined DAC become equal [43][44]. This is the key advantage to using a quadrature mismatch shaping DAC, but comes at the expense of increased hardware complexity.

### 6.2.2 Quadrature Mismatch Shaping

A mismatch shaping scheme for complex DACs is proposed in [43], which generalizes the vector-based mismatch shaper from [13] to complex-valued signals. The structure is realized by requiring each unit-element DAC to take values from $\{0, 1, j\}$, and implementing the mismatch shaping loop filter and vector quantizer using complex arithmetic. An approach using a modified element-rotation scheme is considered in [43], but resolving the contention between the $I$ and $Q$ rotation pointers requires complicated schemes that ultimately result in reduced mismatch suppression.
A complex tree-structured mismatch shaping DAC is considered in [44], along with a complex butterfly shuffler. The complex tree-structured approach is based on the mismatch shaper proposed in [15], but extended to complex signals. The complex butterfly shuffler proposed in [44] is an extension of previously reported butterfly shuffling methods used to whiten or shape the mismatch noise [59]. The butterfly shuffler method requires a higher level of hardware complexity as compared to the tree-structured approach [44].

6.2.3 Tunable Mismatch Shaping

A novel technique for performing tunable mismatch shaping on real signals has been proposed in [16]. This technique relies upon the generalized $N$-path filter principle in conjunction with a prototype mismatch shaper to replicate the mismatch transfer function $N$ times around the unit circle. By using the well-known data-weighted averaging (DWA) technique, a hardware-efficient first-order tunable mismatch shaper can be realized [58]. However, when applied to quadrature signals, DWA mismatch shaping exhibits lower levels of performance [43].

The technique proposed in this chapter extends the tree-based complex mismatch shaper from [44] to allow control over the mismatch transfer function so that the signal band center frequency can be tuned. The next section introduces the proposed tunable quadrature mismatch noise shaper.
6.3 Proposed Tunable Technique

The mismatch shaper used in this work is largely based on the tree-based approach described in [15] and adapted to handle complex signals as described in [44]. Figure 6.5 shows an example of a \( M + 1 \) level DAC, with \( M = 8 \), using the tree structure to perform element selection. Each of the blocks labeled \( S_{k,r} \) is a switching block that routes the input data in two possible directions. There are \( \log_2 M \) layers of switching blocks, and each switching block operates according to:

\[
x_{k-1,2r-1}[n] = \frac{1}{2}(x_{k,r}[n] + s_{k,r}[n]) \quad (6.7)
\]
\[
x_{k-1,2r}[n] = \frac{1}{2}(x_{k,r}[n] - s_{k,r}[n]) \quad (6.8)
\]
In Equations 6.7 and 6.8, $s_{k,r}[n]$ is a switching sequence generated within each switching block. The value of the switching sequence $s_{k,r}[n]$ at each sample interval $n$ dictates what portion of the input data $x_{k,r}[n]$ is routed through each of the outputs of the switching block [15]. As the input travels through the tree, portions of the input data word are spread across the branches of the tree until they arrive at the unit-DACs, where only a single bit determines whether or not the DAC element is selected for activation.

In the case of quadrature mismatch shaping, the input to the DAC is complex-valued. Therefore, each unit-DAC output $y_{k,r}[n]$ can produce one of three possible output values: $y_{k,r}[n] \in \{0, 1, j\}$ [43][44]. The combined DAC has twice the number of unit elements as each individual I and Q component DAC, so the total number of unit-DACs remains the same as when using a pair of real-valued DACs.

Figure 6.6 shows the structure of a switching block that implements the operations described by Equations 6.7 and 6.8 for complex-valued inputs. Since the actual data must remain unchanged from the input of the ESL block to the output, each switching block must ensure that it generates a switching sequence that forces the output data to satisfy this condition. This restriction is known as the number conservation rule [15]. These restrictions need to be modified for the case of quadrature signals in order to ensure that the complex data also satisfies this number conservation rule [44].
Figure 6.6: Switching block for complex data

If the switching sequence $s_{k,r}[n]$ is generated as an $L^{th}$-order noise-shaped sequence, uncorrelated with the switching sequences in the other switching blocks, this will result in an $L^{th}$-order noise-shaped DAC mismatch error sequence [15]. It follows that in order to gain control over the center frequency of the mismatch transfer function, there needs to be a way to control the center frequency of the noise shaping function within each switching sequence generator.

Figure 6.7 shows the structure of a complex-valued sequence generator. Noise shaping is achieved by employing a zero-input ΔΣ modulator with a complex loop filter. The complex-valued number conservation rule is enforced by the complex quantizer and limiter blocks [44]. The order of noise shaping and location of the signal band are determined by the complex loop filter. Therefore, a tunable mismatch shaping function can be achieved by simultaneously controlling the complex loop filters in all the switching blocks.

138
6.4 Hardware Implementation

This section describes the implementation details for the proposed tunable quadrature architecture. Section 6.4.1 describes the design of a tunable complex-valued loop filter which implements a 1\textsuperscript{st}-order mismatch shaping response. Section 6.4.2 describes the quantizer used for both I and Q paths. Section 6.4.3 describes the complex limiter used for ensuring the number conservation rule is obeyed. Section 6.4.4 presents a few simulation examples of the tunable mismatch shaper operating at different tuning settings.

As described in the previous section, each switching block in the mismatch shaping tree contains a sequence generator. The sequence generator shown in Figure 6.7 is essentially a zero-input ΔΣ modulator, and forms the heart of the proposed implementation. There are three parts to the sequence generator: (i) a tunable complex loop filter, (ii) a quantizer for each of the I and Q component paths, and (iii) a complex limiter.

![Figure 6.7: Complex switching sequence generator](image)
6.4.1 Tunable Complex Loop Filter

The loop filter within the switching sequence modulator is implemented as a tunable complex filter, with the center frequency tuned to the center frequency of the desired signal band. The structure of a tunable first-order complex filter is shown in Figure 6.8, where \( \omega_c \) is the center frequency of the signal band, \( 0 \leq \omega_c \leq \pi \), and \([I_x, Q_x]\) and \([I_y, Q_y]\) are the complex-valued input and output vectors of the loop filter. The core of this unit is a rotation unit whose operation is described by:

\[
\begin{bmatrix}
I_y \\
Q_y
\end{bmatrix} = z^{-1} \begin{bmatrix}
\cos \omega_c & -\sin \omega_c \\
\sin \omega_c & \cos \omega_c
\end{bmatrix} \begin{bmatrix}
I_y - I_x \\
Q_y - Q_x
\end{bmatrix}
\] (6.9)

The datapath and coefficient widths are listed in Table 6.1 for each DAC size. The path width corresponds to the final truncated width of the registers shown in Figure 6.8. In order to simultaneously tune the loop filters
of all the switching sequence generators, the same complex coefficients must be provided to all $\log_2 M$ switching blocks and their loop filters. The filter coefficients associated with each tuning setting can be stored in a lookup-table or read-only memory (ROM), and indexed by the tuning frequency. Since all switching blocks follow the same tuning frequency, their loop filters share the same coefficients at any given tuning setting. Therefore, the lookup-table provides the same set of coefficients to all $\log_2 M$ switching blocks.

Figure 6.9 shows the coefficients plotted over the entire tuning range. Since the coefficients are essentially a sine function computed at the tuning frequency, the coefficient storage requirements can be reduced by taking advantage of the natural quadrant-symmetry exhibited by the sine wave. The first quadrant of the cosine function is identical to the second quadrant of the sine function, and the second quadrant of the cosine function is identical to the second quadrant of the sine function, with the opposite polarity. The second quadrant of the cosine function can be reproduced from the first quadrant by reversing the phase. The size of the ROM can be further reduced by using a direct digital frequency synthesizer (DDFS), such as one using polynomial interpolation [46].
Alternatively, the coefficients can be computed using existing hardware from elsewhere in the wireless system. Most modern transceiver systems include a DDFS, which can be borrowed for a few cycles in order to compute the pair of coefficients. The coefficients would only need to be re-computed when the location of the signal band is changed.

6.4.2 Path Quantizer

As described in Section 6.3, each switching block in the tree must satisfy a restriction known as the number conservation rule [15]. This restriction is imposed in order to ensure that data remains unchanged between the input and output of each node in the tree, and hence the ESL block shown in Figure 6.3. This restriction can be transferred from the overall tree to each switching block by ensuring that the sum of the two outputs of each switching block always equals the input to the switching block.

Within each switching block, the sequence generator produces a number that is used to create the two switching block outputs, as shown in Figure 6.6.
The sequence output is first simultaneously added to and subtracted from the switching block input. These two results are then divided by two (or truncated by a single LSB) before supplying them to subsequent switching blocks. Due to this division by two, and since the switching block outputs cannot be allowed to take on fractional values, the results of the addition and subtraction must always be maintained as even numbers.

This condition can be satisfied by restricting the switching sequence to always be even-valued when the switching block input data is even-valued, and vice-versa. As described in [15], this can be achieved by selecting a mid-tread quantizer when the input to the switching block is even, and a mid-rise quantizer when it is odd. Since the addition and subtraction within the switching block occurs separately for each of the I and Q paths, it is sufficient to maintain separate quantizers for each path. Each of the quantizers must independently choose between mid-rise and mid-tread quantization according to the respective I or Q components of the switching block input.

6.4.3 Complex Limiter

The purpose of the limiter is to prevent the switching block from ever producing a negative number. In order to achieve this, the magnitude of the sequence generator output is forcibly limited according the switching block input data. In the case of the combined complex sequence generator, additional limiter constraints are needed to satisfy the number conservation rule [44].
The operation of the limiter in the lowest layers of the switching-block tree is shown in Table 6.2. The first column shows the possible switching block inputs. The middle columns show the real and complex components of the quantizer output. The last column shows the corresponding sequence output resulting from the combination of switching block input and quantizer output. These sequence generator outputs guarantee that each of the two switching block outputs take on valid control values for the complex DAC unit-element \( s_k[n] \in \{0, 1, j\} \).

There are a few input combinations where two different output sequence values can produce the same valid complex switching block outputs. These cases are also shown in Table 6.2, indicated with a pair of possible output sequence options. In the proposed technique, whenever such a condition arises, the choice between the two options is made according to a pseudo-random sequence, which is generated by a linear-feedback shift register (LFSR).

As shown in Figures 6.5 and 6.7, the combinatorial logic path connecting the DAC input to the DAC unit-element output contains every single quantizer and limiter block in the mismatch shaping tree. This path can be pipelined in order to shorten the critical path for high speed operation. However, pipelining creates additional clock cycle delays through the DAC, and any increase in this delay can pose modulator stability problems when these DACs are employed within the modulator feedback loop of a \( \Delta \Sigma \) ADC [1]. It is therefore beneficial to minimize the complexity of all limiter blocks.
Table 6.2: Limiter table for the lowest layer of the tree

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Re{Quan}</th>
<th>Im{Quan}</th>
<th>Sequence Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>2j</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>±1</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>−</td>
<td>X</td>
<td>−1</td>
</tr>
<tr>
<td>j</td>
<td>X</td>
<td>+</td>
<td>±j</td>
</tr>
<tr>
<td>j</td>
<td>X</td>
<td>−</td>
<td>−j</td>
</tr>
<tr>
<td>1 + j</td>
<td>0</td>
<td>0</td>
<td>±(1 − j)</td>
</tr>
<tr>
<td>1 + j</td>
<td>0</td>
<td>+</td>
<td>−1 + j</td>
</tr>
<tr>
<td>1 + j</td>
<td>0</td>
<td>−</td>
<td>1 − j</td>
</tr>
<tr>
<td>1 + j</td>
<td>+</td>
<td>0</td>
<td>1 − j</td>
</tr>
<tr>
<td>1 + j</td>
<td>−</td>
<td>0</td>
<td>−1 + j</td>
</tr>
<tr>
<td>1 + j</td>
<td>+</td>
<td>−</td>
<td>1 − j</td>
</tr>
<tr>
<td>1 + j</td>
<td>−</td>
<td>+</td>
<td>−1 + j</td>
</tr>
<tr>
<td>1 + j</td>
<td>+</td>
<td>&gt;</td>
<td>1 − j</td>
</tr>
<tr>
<td>1 + j</td>
<td>−</td>
<td>&gt;</td>
<td>1 − j</td>
</tr>
<tr>
<td>1 + j</td>
<td>+</td>
<td>&lt;</td>
<td>−1 + j</td>
</tr>
<tr>
<td>1 + j</td>
<td>−</td>
<td>&lt;</td>
<td>−1 + j</td>
</tr>
<tr>
<td>1 + j</td>
<td>+</td>
<td>=</td>
<td>±(1 − j)</td>
</tr>
<tr>
<td>1 + j</td>
<td>−</td>
<td>=</td>
<td>±(1 − j)</td>
</tr>
</tbody>
</table>

The limiter at the lowest switching levels need to be as complicated as described by Table 6.2, in order to ensure that the combined complex output is a valid control for the individual complex DAC elements. However, the limiters in the upper levels need only satisfy the number conservation rule. It is possible to reduce the complexity of the limiters used in the upper layer switching blocks by drastically simplifying the limiter rules.
Table 6.3: Limiter table for all upper layers of the tree

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Quantizer Output</th>
<th>Sequence Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>even</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>even</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>even</td>
<td>−</td>
<td>0</td>
</tr>
<tr>
<td>odd</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>odd</td>
<td>−1</td>
<td>−1</td>
</tr>
<tr>
<td>odd</td>
<td>&gt;1</td>
<td>1</td>
</tr>
<tr>
<td>odd</td>
<td>&lt;−1</td>
<td>−1</td>
</tr>
</tbody>
</table>

In the proposed technique, the switching blocks in upper layers of the tree all generate switching sequences whose values are limited to produce \(I\) and \(Q\) component values from the ranges \((-1, 0, 1)\) and \((-j, 0, j)\), respectively. This effectively places tighter constraints on values allowed in the switching sequence output. Table 6.3 shows the operation of the simplified limiter used in all switching blocks in the upper layers of the tree. The table applies to both \(I\) and \(Q\) paths. The first column shows the switching block inputs as either even or odd. The second column shows the possible quantizer outputs. The third column shows the corresponding sequence output resulting from the combination of switching block input and quantizer output.

Simulation results show that using this highly constrained limiter for upper layers does not significantly reduce mismatch noise suppression in the signal band. The advantages of using such a scheme extend beyond the hardware savings within the limiter blocks: the range of possible sequence values fed back into the loop filter now take on trivial values, thereby reducing the hardware complexity of the initial stages of the loop filter.
6.4.4 Simulation of Tunable Operation

To illustrate the operation of the tunable quadrature mismatch shaper, Figure 6.10 shows the frequency spectra at three different signal band tuning frequencies, centered at (a) \(0.1875\pi\), (b) \(0.2812\pi\), and (c) \(0.3750\pi\). Each figure shows three distinct frequency responses. The ideal response is labeled *no mismatch*, and corresponds to that of a complex DAC without any mismatch errors, where performance is limited only by the tunable bandpass \(\Delta\Sigma\) modulator itself. The *unshaped* response is when random mismatch is added to the DAC, but no mismatch shaping is performed. In this case, the element selection for each path is performed using a simple thermometer code. The *shaped* response is when mismatch is added and mismatch noise shaping is performed using the proposed technique.

In this example, the \(\Delta\Sigma\) modulator and DAC are using 16 complex unit elements. The sequence generators in the mismatch shaper tree all use the tunable 1\(^{st}\)-order complex filter described in Section 6.4.1, tuned to the same frequency setting. The DAC mismatch errors are modeled as uniformly distributed random mismatch, with a standard deviation of 3% of a single unit element value. Figure 6.11 shows the detailed views of the signal bands corresponding to each example tuning setting shown in Figure 6.10.

These simulation examples show that the proposed mismatch shaper provides consistent mismatch noise shaping performance over the tuning range. The performance is characterized by the signal-to-noise ratio (SNR) computed for the signal band.
Figure 6.10: Full spectrum with 3% mismatch error
Figure 6.11: Signal band detail with 3% mismatch error

(a) Center frequency tuned to 0.1875\(\pi\)

(b) Center frequency tuned to 0.2812\(\pi\)

(c) Center frequency tuned to 0.3750\(\pi\)
6.5 Comparison of Shaping Performance

In this section, the mismatch shaping performance of the proposed architecture is compared to a reference architecture. The latter is a conventional implementation using a pair of real-valued tunable bandpass mismatch shaping DACs, as described in Chapter 4. Each architecture has been implemented in VHDL and simulated using data generated by a tunable 4\textsuperscript{th}-order quadrature bandpass $\Delta \Sigma$ modulator whose specifications are shown in Table 6.4. An oversampling ratio of 128 allows coverage of the entire Nyquist band using just 64 tuning settings. The DAC mismatch errors are modeled as uniformly distributed random mismatch, with the standard deviation expressed as a percentage of a single unit element value.

Table 6.4: Quadrature BP $\Delta \Sigma$ modulator specifications

<table>
<thead>
<tr>
<th>Modulator order</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSR</td>
<td>128</td>
</tr>
<tr>
<td>Quantizer levels</td>
<td>variable</td>
</tr>
<tr>
<td>Tuning settings</td>
<td>64</td>
</tr>
<tr>
<td>Normalized signal bandwidth</td>
<td>0.0078</td>
</tr>
</tbody>
</table>

Figures 6.12–6.18 compare the simulated SNR of the proposed and conventional shapers with complex DAC sizes of 2, 4, 8, 16, 32, 64, and 128 unit elements, respectively. The input signal is a full-scale complex-valued sinusoid with a frequency randomly selected from within the signal band. This is repeated for each tuning setting across the entire tuning range.
In each figure, the SNR with no mismatch is that of a DAC without any mismatch errors, where the performance is limited by the tunable bandpass ΔΣ modulator specified in Table 6.4. The unshaped SNR is the response when mismatch is added, but no shaping is performed. In this case, the element selection is performed using a simple thermometer code. The shaped SNR is the response when mismatch is added and mismatch shaping is performed using the corresponding shaper. In all figures, the proposed tunable shaper is denoted by shaped complex tree while the conventional tunable shaper is denoted by shaped dual tree. Each data-point corresponds to the SNR average computed for a thousand different randomly-generated mismatch DAC configurations, at each tuning frequency setting. In all figures, the standard deviation of the applied mismatch is 3% of the unit element step-size. The performance figures for 1% and 2% mismatch are similar to the case of 3% mismatch, and these figures are included in Appendix 1.

The mismatch-shaped performance shows consistent improvement over the unshaped performance, for the entire tuning range. It is notable that the performance of the two architectures are very similar. Normally, the complex mismatch shaper would be expected to show higher performance due to the absence of complex-conjugate poles and zeros in the mismatch transfer function. However, the performance of the proposed implementation has been intentionally lowered to match the performance of the conventional approach, in order to lower the hardware complexity. Lowered performance is achieved by reducing the bit-widths of the datapath and the coefficients.
Figure 6.12: SNR with 3% mismatch for 2-element quadrature shapers

Figure 6.13: SNR with 3% mismatch for 4-element quadrature shapers
Figure 6.14: SNR with 3% mismatch for 8-element quadrature shapers

Figure 6.15: SNR with 3% mismatch for 16-element quadrature shapers
Figure 6.16: SNR with 3% mismatch for 32-element quadrature shapers

Figure 6.17: SNR with 3% mismatch for 64-element quadrature shapers
In this section, the hardware complexity of the proposed tunable architecture is compared to that of a reference architecture. The reference architecture has been implemented in order to compare the hardware overhead of using a tunable quadrature bandpass mismatch shaper. This consists of a pair of real-valued tunable tree-based mismatch shapers, each operating on either the $I$ or $Q$ paths of the $\Delta\Sigma$ modulator. The conventional mismatch shapers are based on the real-valued tree-based mismatch shaper from [15], but extended to operate at tunable signal band center frequencies, as described in Chapter 4.
Both sets of architectures have been implemented in a hardware description language (VHDL) and tested for several DAC sizes. The designs for each DAC size have been synthesized to logic gates using the Synopsys Design Compiler synthesis tool [50] under the conditions specified in Table 6.5.

Table 6.5: Hardware synthesis conditions

<table>
<thead>
<tr>
<th>Process</th>
<th>0.18 µm CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device corners</td>
<td>slow-NMOS / slow-PMOS</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.62 V</td>
</tr>
<tr>
<td>Temperature</td>
<td>125°C</td>
</tr>
</tbody>
</table>

Synthesis results are shown for each DAC size in Figures 6.19 and 6.20. For each structure, the critical path, or shortest cycle-time, is plotted against DAC size in Figure 6.19. For smaller DAC sizes, both the proposed and the reference architectures exhibit similar critical path delays. However, as the DAC size is increased, the critical path delay of the conventional approach rises faster than that of the quadrature approach.

Figure 6.20 shows the cumulative cell area corresponding to the points in Figure 6.19. The cell area is calculated in square-microns (µm²) and plotted on a logarithmic scale. As is expected, the proposed quadrature or complex tree structure requires a greater area than the conventional dual tree structure, for all DAC sizes.
Figure 6.19: Comparison of complex tree critical path delays

Figure 6.20: Comparison of complex tree hardware complexity
6.7 Summary

Wireless transceivers are increasingly employing quadrature bandpass ΔΣ data converters in applications where high linearity is required over a narrow bandwidth. The dynamic range of a ΔΣ data converter can be improved by using a multibit quantizer in the modulator loop. However, device mismatch errors in the multibit DAC can cause distortion that has a direct impact on the effective modulator performance. Mismatch shaping is an established technique for alleviating the effects of mismatch errors. In a quadrature ΔΣ modulator, any mismatch between the I and Q component paths can result in quantization noise folding back into the signal band. This path mismatch can be reduced by combining the two real-valued DACs into a single mismatch shaping complex DAC. This approach equalizes the average gain error through the I and Q component paths of the quadrature DAC. However, this benefit comes at the expense of greater hardware complexity.

Multi-standard wireless systems can also benefit from the ability to place the center of the signal band at arbitrary frequency locations within the Nyquist range of the oversampled data converter. This requires that the quadrature mismatch shaper also retain the ability to arbitrarily select the frequency location of the mismatch transfer function. However, previously reported techniques for quadrature mismatch shaping DACs require the signal band be located at a fixed frequency. This chapter extends previously reported mismatch shaping techniques to allow the center frequency of the quadrature mismatch suppression band to be tunable over the entire Nyquist range.
The proposed quadrature bandpass mismatch shaper is made tunable by employing a tunable quadrature loop filter within the tree-structured mismatch shaper. The limiter in upper layer switching blocks has been simplified in order to reduce the path delay from input to output. To evaluate the hardware complexity, a reference architecture employing a pair of real-valued tunable mismatch shapers has also been implemented.

The proposed and reference designs have been implemented in VHDL for various DAC sizes and synthesized to logic gates. Synthesis results show a slight but consistent increase in hardware complexity as compared to the conventional approach. The two approaches exhibit similar critical path delays, with the proposed design showing lower latency as the DAC size is increased. All designs have been simulated using a tunable quadrature bandpass ΔΣ modulator, with random mismatch errors added to the DAC unit elements. Simulation results show consistent mismatch shaping performance over the entire tuning range, as characterized by the in-band signal-to-noise ratio (SNR).
Chapter 7

Conclusion

This work explores a series of techniques that extend the advantages of mismatch shaping to tunable applications. This is done by allowing the suppression band of the mismatch noise shaping function to have an adjustable center frequency. The proposed tunable techniques have been implemented in hardware for various DAC sizes and evaluated according to mismatch shaping performance, latency and hardware complexity.

The designs are implemented in VHDL and synthesized to logic gates using a 0.18\(\mu m\) CMOS process technology. The standard cells in this library nominally run at a voltage of 1.8V. The designs are tested through simulation using data generated by a tunable bandpass \(\Delta\Sigma\) modulator. The oversampling ratio is chosen to allow coverage of the entire Nyquist frequency band using only 64 tuning settings. Mismatch errors are modeled by uniformly distributed random variations in the unit DAC elements, with standard deviations of 1\% – 3\% with respect to the DAC unit step size.
7.1 Tunable Mismatch Shaping for Real Signals

Two techniques are described that extend the previously reported vector-based and tree-based mismatch shapers in order to allow the signal band to occupy a tunable range of center frequencies. Chapter 3 presents the vector-based mismatch shaper extended to allow tunable operation. Chapter 4 proposes several architectures that enable tunable operation in the tree-structured mismatch shaper. These techniques achieve 1\textsuperscript{st}-order and 2\textsuperscript{nd}-order mismatch shaping, and simulation results show consistent performance improvements across the entire tuning range.

Chapter 5 proposes the tunable $N$-path technique for tunable bandpass mismatch shaping. This technique has been implemented using three different architectures (DWA, tree-based and vector-based) as the prototype mismatch shaper used within the $N$-path structure. Simulations of these architectures show similar mismatch shaping performance across the tuning range, but worse than both the tunable vector and tree shapers described in Chapters 3 and 4, respectively. Hardware synthesis results show that the $N$-path DWA architecture has the smallest cell area and is the fastest of the three.

Table 7.1 shows a decision matrix for selecting the best tunable mismatch shaping architecture based on mismatch shaping performance, hardware complexity and speed of operation. For applications that require less than 1\textsuperscript{st}-order mismatch suppression across the entire tuning range, the tunable $N$-path shaper with DWA prototype exhibits the highest speed and lowest complexity.
<table>
<thead>
<tr>
<th>Mismatch Shaper</th>
<th>Performance</th>
<th>Complexity</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-path DWA</td>
<td>&lt; 1&lt;sup&gt;st&lt;/sup&gt;-order</td>
<td>Small</td>
<td>Fast</td>
</tr>
<tr>
<td>Tunable Tree Polar2</td>
<td>1&lt;sup&gt;st&lt;/sup&gt;-order</td>
<td>Medium</td>
<td>Moderate</td>
</tr>
<tr>
<td>Tunable Tree CRFB4</td>
<td>2&lt;sup&gt;nd&lt;/sup&gt;-order</td>
<td>Large</td>
<td>Slow</td>
</tr>
</tbody>
</table>

When a 1<sup>st</sup>-order mismatch shaping performance is required across the tuning range, the tunable tree-based mismatch shaper using the 2<sup>nd</sup>-order polar architecture can be employed. This approach requires a moderate hardware complexity and achieves reasonably fast speeds of operation. For applications that require 2<sup>nd</sup>-order mismatch shaping performance, the tunable tree-based mismatch shaper with the 4<sup>th</sup>-order CRFB architecture can be used. This choice comes with a much higher cost of hardware complexity and greater limitations on the maximum speed of operation.

### 7.2 Tunable Mismatch Shaping for Complex Signals

In a quadrature ΔΣ modulator, mismatch between the I and Q paths can result in quantization noise folding back into the signal band. Combining the two real-valued DACs into a single mismatch shaping complex DAC averages the gain errors to remain equal through both I and Q paths. This is the primary advantage to using a quadrature mismatch shaping DAC. However, this advantage comes at the expense of increased hardware complexity of the mismatch shaping element selection logic.
Chapter 6 presents an extension to the tunable tree-structured mismatch shaper from Chapter 4 that allows the DAC to process complex-valued signals. This technique allows a quadrature bandpass DAC to share the benefits of tunable mismatch shaping. The structure employs a tunable complex loop filter and is implemented using a polar rotation architecture. The hardware implementation has been synthesized into logic gates in order to compare complexity and speed of this approach to that of a conventional structure employing dual real-valued tunable mismatch shapers, one for each path. Synthesis results show a moderate increase in hardware complexity as compared to the conventional approach. Simulation results show consistent SNR performance across the entire tuning range.
Appendix 1

Mismatch Shaping Performance

1.1 Comparison of Tunable Real Mismatch Shapers

Table 1.1: Index of performance figures for real-valued shapers

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Mismatch STD</th>
<th>DAC Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1%</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>2%</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>1%</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>2%</td>
<td>4</td>
</tr>
<tr>
<td>1.5</td>
<td>1%</td>
<td>8</td>
</tr>
<tr>
<td>1.6</td>
<td>2%</td>
<td>8</td>
</tr>
<tr>
<td>1.7</td>
<td>1%</td>
<td>16</td>
</tr>
<tr>
<td>1.8</td>
<td>2%</td>
<td>16</td>
</tr>
<tr>
<td>1.9</td>
<td>1%</td>
<td>32</td>
</tr>
<tr>
<td>1.10</td>
<td>2%</td>
<td>32</td>
</tr>
<tr>
<td>1.11</td>
<td>1%</td>
<td>64</td>
</tr>
<tr>
<td>1.12</td>
<td>2%</td>
<td>64</td>
</tr>
<tr>
<td>1.13</td>
<td>1%</td>
<td>128</td>
</tr>
<tr>
<td>1.14</td>
<td>2%</td>
<td>128</td>
</tr>
</tbody>
</table>
Figure 1.1: SNR with 1% mismatch for 2-element N-path & tree shapers

Figure 1.2: SNR with 2% mismatch for 2-element N-path & tree shapers
Figure 1.3: SNR with 1% mismatch for 4-element N-path & tree shapers

Figure 1.4: SNR with 2% mismatch for 4-element N-path & tree shapers
Figure 1.5: SNR with 1% mismatch for 8-element N-path & tree shapers

Figure 1.6: SNR with 2% mismatch for 8-element N-path & tree shapers
Figure 1.7: SNR with 1% mismatch for 16-element N-path & tree shapers

Figure 1.8: SNR with 2% mismatch for 16-element N-path & tree shapers
Figure 1.9: SNR with 1% mismatch for 32-element N-path & tree shapers

Figure 1.10: SNR with 2% mismatch for 32-element N-path & tree shapers
Figure 1.11: SNR with 1% mismatch for 64-element N-path & tree shapers

Figure 1.12: SNR with 2% mismatch for 64-element N-path & tree shapers
Figure 1.13: SNR with 1% mismatch for 128-element N-path & tree shapers

Figure 1.14: SNR with 2% mismatch for 128-element N-path & tree shapers
1.2 Comparison of Tunable Quadrature Mismatch Shapers

Table 1.2: Index of performance figures for complex-valued shapers

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Mismatch STD</th>
<th>DAC Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.15</td>
<td>1%</td>
<td>2</td>
</tr>
<tr>
<td>1.16</td>
<td>2%</td>
<td>2</td>
</tr>
<tr>
<td>1.17</td>
<td>1%</td>
<td>4</td>
</tr>
<tr>
<td>1.18</td>
<td>2%</td>
<td>4</td>
</tr>
<tr>
<td>1.19</td>
<td>1%</td>
<td>8</td>
</tr>
<tr>
<td>1.20</td>
<td>2%</td>
<td>8</td>
</tr>
<tr>
<td>1.21</td>
<td>1%</td>
<td>16</td>
</tr>
<tr>
<td>1.22</td>
<td>2%</td>
<td>16</td>
</tr>
<tr>
<td>1.23</td>
<td>1%</td>
<td>32</td>
</tr>
<tr>
<td>1.24</td>
<td>2%</td>
<td>32</td>
</tr>
<tr>
<td>1.25</td>
<td>1%</td>
<td>64</td>
</tr>
<tr>
<td>1.26</td>
<td>2%</td>
<td>64</td>
</tr>
<tr>
<td>1.27</td>
<td>1%</td>
<td>128</td>
</tr>
<tr>
<td>1.28</td>
<td>2%</td>
<td>128</td>
</tr>
</tbody>
</table>
Figure 1.15: SNR with 1% mismatch for 2-element quadrature shapers

Figure 1.16: SNR with 2% mismatch for 2-element quadrature shapers
Figure 1.17: SNR with 1% mismatch for 4-element quadrature shapers

Figure 1.18: SNR with 2% mismatch for 4-element quadrature shapers
Figure 1.19: SNR with 1% mismatch for 8-element quadrature shapers

Figure 1.20: SNR with 2% mismatch for 8-element quadrature shapers
Figure 1.21: SNR with 1% mismatch for 16-element quadrature shapers

Figure 1.22: SNR with 2% mismatch for 16-element quadrature shapers
Figure 1.23: SNR with 1% mismatch for 32-element quadrature shapers

Figure 1.24: SNR with 2% mismatch for 32-element quadrature shapers
Figure 1.25: SNR with 1% mismatch for 64-element quadrature shapers

Figure 1.26: SNR with 2% mismatch for 64-element quadrature shapers
Figure 1.27: SNR with 1% mismatch for 128-element quadrature shapers

Figure 1.28: SNR with 2% mismatch for 128-element quadrature shapers
Bibliography


182


[26] K. Chen and T. Kuo, “An improved technique for reducing baseband tones in sigma-delta modulators employing the data weighted averaging


[54] R. Schreier, “The Delta-Sigma Toolbox for Matlab.”.


Vita

Waqas Akram was born in Lagos, Nigeria. He received both a Bachelor of Science degree in Electrical and Computer Engineering, and a Bachelor of Arts degree in Computer Science from Rice University in 1997. He received a Master of Science degree in Electrical and Computer Engineering from the University of Texas at Austin in 2003. From 1997 until 2008, he worked as a design engineer at Crystal Semiconductor Corp., later called Cirrus Logic Inc., based in Austin, Texas. He has designed communications ICs for Ethernet and T1/E1 systems, digital signal processors for consumer electronics, and high-resolution data converters for audio applications.

Permanent address: 1713A Summit View Place
Austin, Texas 78703

This dissertation was typeset with \TeX† by the author.

†\TeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth’s \TeX Program.