On Generating Complex Numbers for FFT and NCO Using the CORDIC Algorithm

Examensarbete utfört i Datorteknik vid Tekniska högskolan i Linköping

av

Anton Andersson

LITH-ISY-EX--08/4197--SE

Linköping 2008
On Generating Complex Numbers for FFT and NCO Using the CORDIC Algorithm

Examensarbete utfört i Datorteknik
vid Tekniska högskolan i Linköping
av
Anton Andersson

LITH-ISY-EX--08/4197--SE

Handledare: Anders Nilsson
Coresonic AB

Examinator: Dake Liu
ISY, Linköpings universitet

Linköping, 14 November, 2008
On Generating Complex Numbers for FFT and NCO Using the CORDIC Algorithm

Title of series, numbering

This report has been compiled to document the thesis work carried out by Anton Andersson for Coresonic AB. The task was to develop an accelerator that could generate complex numbers suitable for fast Fourier transforms (FFT) and tuning the phase of complex signals (NCO). Of many ways to achieve this, the CORDIC algorithm was chosen. It is very well suited since the basic implementation allows rotation of 2D-vectors using only shift and add operations. Error bounds and proof of convergence are derived carefully.

The accelerator was implemented in VHDL in such a way that all critical parameters were easy to change. Performance measures were extracted by simulating realistic test cases and then compare the output with reference data precomputed with high precision. Hardware costs were estimated by synthesizing a set of different configurations. Utilizing graphs of performance versus cost makes it possible to choose an optimal configuration.

Maximum errors were extracted from simulations and seemed rather large for some configurations. The maximum error distribution was then plotted in histograms revealing that the typical error is often much smaller than the largest one. Even after trouble-shooting, the errors still seem to be somewhat larger than what other implementations of CORDIC achieve. However, precision was concluded to be sufficient for targeted applications.

Keywords

FFT, NCO, CORDIC algorithm, CORDIC convergence, CORDIC error analysis
Abstract

This report has been compiled to document the thesis work carried out by Anton Andersson for Coresonic AB. The task was to develop an accelerator that could generate complex numbers suitable for fast fourier transforms (FFT) and tuning the phase of complex signals (NCO). Of many ways to achieve this, the CORDIC algorithm was chosen. It is very well suited since the basic implementation allows rotation of 2D-vectors using only shift and add operations. Error bounds and proof of convergence are derived carefully.

The accelerator was implemented in VHDL in such a way that all critical parameters were easy to change. Performance measures were extracted by simulating realistic test cases and then compare the output with reference data precomputed with high precision. Hardware costs were estimated by synthesizing a set of different configurations. Utilizing graphs of performance versus cost makes it possible to choose an optimal configuration.

Maximum errors were extracted from simulations and seemed rather large for some configurations. The maximum error distribution was then plotted in histograms revealing that the typical error is often much smaller than the largest one. Even after trouble-shooting, the errors still seem to be somewhat larger than what other implementations of CORDIC achieve. However, precision was concluded to be sufficient for targeted applications.

Sammanfattning

Den här rapporten dokumenterar det examensarbete som utförts av Anton Andersson för Coresonic AB. Uppgiften bestod i att utveckla en accelerator som kan generera komplexa tal som är lämpliga att använda för snabba fouriertransformationer (FFT) och tunnling av komplexa signaler (NCO). Det finns en mängd sätt att göra detta men valet föll på en algoritm kallad CORDIC. Den är mycket lämplig då den kan rotera 2D-vektorer godtycklig vinkel med enkla operationer som bitskift och addition. Felgränser och konvergens härleds noggrann.

Acceleratorn implementerades i språket VHDL med målet att kritiska parametrar enkelt skall kunna förändras. Därefter simulerades modellen i realistiska testfall och resultaten jämfördes med referensdata som förberäknats med mycket hög precision. Dessutom syntetiserades en mängd olika konfigurationer så att prestanda enkelt kan viktas mot kostnad.
Ur de koefficienter som erhölls genom simuleringar beräknades det största erhållna felet för en mängd olika konfigurationer. Felen verkade till en början onormalt stora vilket krävde vidare undersökning. Samtliga fel från en konfiguration ritades i histogramform, vilket visade att det typiska felet oftast var betydligt mindre än det största. Även efter felsökning verkar acceleratorn generera tal med något större fel än andra implementationer av CORDIC. Precisionen anses dock vara tillräcklig för avsedda applikationer.
Acknowledgments

This final year project has been carried out at Coresonic AB, a company developing baseband processors in Linköping. I want to express my gratitude to Anders Nilsson, Björn Skoglund, Erik Alfredsson, Eric Tell and Jeanette Kilgren. They have been very helpful and full of ideas every time my progress stalled or when the computers misbehaved. I would also like to thank Dake Liu, my examiner at LiU. Martin Andersson and Robert Kihlberg have made large efforts proofreading various versions of this report.

Sincerely thank you!
# Contents

1 Introduction ................................................................. 1  
1.1 Problem description .................................................. 1  
1.2 Coresonic ................................................................. 2  
1.3 Method ................................................................. 2  
1.4 Assumed prior knowledge .............................................. 2  
1.5 Disposition ............................................................. 3  

2 Fast Fourier Transform .................................................. 5  
2.1 FT, DTFT, DFT, FFT .................................................. 5  
2.2 Cooley-Tukey DIT FFT ................................................ 7  
2.3 Radix-4 DIF FFT ....................................................... 8  

3 The CORDIC algorithm .................................................... 13  
3.1 Notation ............................................................... 14  
3.2 Theory ................................................................. 14  
3.3 Convergence ........................................................... 16  
3.4 Error analysis .......................................................... 19  
  3.4.1 Angle approximation error .................................. 20  
  3.4.2 Rounding error ............................................... 21  
  3.4.3 Total error .................................................. 23  

4 Implementation ............................................................. 25  
4.1 Package generator ................................................... 26  
4.2 Accelerator interface ................................................ 26  
  4.2.1 Control Register File ....................................... 26  
  4.2.2 Complex Network ............................................ 27
<table>
<thead>
<tr>
<th>x</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 CORDIC step</td>
<td>27</td>
</tr>
<tr>
<td>4.4 Prerotation stage</td>
<td>28</td>
</tr>
<tr>
<td>4.5 CORDIC rotator</td>
<td>29</td>
</tr>
<tr>
<td>4.6 Control and interface</td>
<td>31</td>
</tr>
<tr>
<td>5 Simulation and Synthesis</td>
<td>33</td>
</tr>
<tr>
<td>5.1 FFT SNR</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Twiddle Factor precision</td>
<td>34</td>
</tr>
<tr>
<td>5.3 Synthesis for silicon</td>
<td>34</td>
</tr>
<tr>
<td>6 Conclusion and discussion</td>
<td>37</td>
</tr>
<tr>
<td>7 Future Work</td>
<td>39</td>
</tr>
<tr>
<td>Bibliography</td>
<td>41</td>
</tr>
<tr>
<td>A Abbreviations</td>
<td>43</td>
</tr>
<tr>
<td>B FFT SNR plots</td>
<td>44</td>
</tr>
<tr>
<td>C Twiddle factor precision plots</td>
<td>46</td>
</tr>
<tr>
<td>D Absolute error histograms</td>
<td>49</td>
</tr>
<tr>
<td>E Synthesis for silicon plots</td>
<td>54</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

There are many ways to perform calculations even if the algorithm is well specified. In computer engineering it is seldom enough only to think of arithmetic operations, one has to regard data flow and storage of intermediate results as well. There is often a trade-off between computing efficiency and hardware cost that must be considered since nobody will benefit from overcapacity.

A very common way to analyze signals in DSP systems is to apply a fourier transform to determine its harmonic content. Such operations requires a sequence of complex coefficients regularly spaced on the unit circle, often called twiddle factors. The LeoCore processor developed by Coresonic AB in Linköping, Sweden uses a radix-4 DIF FFT algorithm. Previously all coefficients have been calculated beforehand and stored in a data memory during execution. This is wasteful use of memory since only four coefficients are used per cycle.

If the transmitter and receiver system in a communication link are slightly out of synchronization it will seem as if the recorded signal is revolving slowly even if it is supposed to be constant. It is possible to rotate the signal back to the intended phase by estimating the mismatch and multiply the signal with suitable complex exponentials. This signal can be generated by a numerically controlled oscillator, NCO.

Both applications require the same kind of complex numbers and it should thus be possible to generate them using a shared accelerator unit.

1.1 Problem description

An accelerator unit that may act as an NCO and produce coefficients for FFT shall be designed. It must fulfill the following requirements:

- It must be able to generate FFT twiddle factors for radix-4 and radix-2 operations. (FFT mode)
• Compute 0-4 complex numbers per cycle with consecutive angles. The increment should be possible to update at any time. (NCO mode)

• The design should be modular so that it is possible to upgrade it to support larger transform sizes.

• Important design parameters such as internal and external word lengths and number of iterations etc should be easy to change.

1.2 Coresonic

Coresonic AB was formed as a result of a need to develop a new wireless baseband processor architecture that was programmable but did not suffer the usual downsides of increased power consumption and cost.

They were founded in 2004 based on the outcome of a three-year research project at the research center Stringent at Linköping University, Sweden. The founders include Professor Dake Liu, Dr. Eric Tell, Dr. Anders Nilsson and Professor Christer Svensson from the electrical engineering department who bring together decades of experience from research, development and marketing of semiconductors and IP.

Their patented architecture offers a solution to the key issues faced by developers of baseband products for the WiMAX and 4G market. They provide a semiconductor core design and support the core with protocol specific firmware solutions.

1.3 Method

FFT coefficient generation and NCO operation essentially require a hardware unit that is capable of rotating vectors. Efficient algorithms and implementations have been under research for decades and much material have been published. Therefore the first step was to spend some time reading papers on different algorithms and implementations and then sketch different solutions on paper. The design was then transfered to RTL code in VHDL with the intention that it should be suitable for synthesis and well-documented.

Large amounts of information were then generated during many simulation and synthesizing runs of the accelerator. Matlab was then used extensively to generate reference signals for comparison and SNR estimation. It have also proved to be invaluable when compiling data for plots from many different files.

1.4 Assumed prior knowledge

This report is written under the assumption the reader is familiar with concepts of computer engineering. Some chapters are mathematically oriented and knowledge of calculus are probably required to understand the proofs and derivations.
Although those parts are interesting, they are not crucial in order to understand the function of the accelerator.

1.5 Disposition

The titles of the chapters should convey their content relatively clearly. It should be a good idea to read them in the given order.

Chapter 1 gives background information about this project.

Chapter 2 contains an introduction to the Fourier transform and how symmetry may be used to derive fast versions.

Chapter 3 introduces the CORDIC algorithm that the accelerator utilize. The chapter contains a thorough proof of convergence and an error analysis.

Chapter 4 describes how the algorithm was implemented in hardware.

Chapter 5 explains how the accelerator was simulated and synthesized.

Chapter 6 summarizes the results of the thesis work.

Chapter 7 lists some topics that could be investigated further.

Appendices A-E contain a list of common abbreviations and a collection of plots and graphs.
Chapter 2

Fast Fourier Transform

Fourier analysis as we know it today was derived by Josef Fourier in the 19th century while trying to mathematically describe the transport of heat. His main result was that a vast amount of different problems may be solved or at least be seen more clearly by expressing their functions as trigonometric series. A common interpretation is that the Fourier transform is a tool to investigate the harmonic content of signals, i.e., the frequencies they contain. In reality problems arise because we may only record a finite number of samples of a signal. This reduces the resolution of the transform and the amount of information that can be extracted is thus limited.

2.1 FT, DTFT, DFT, FFT

In many cases it is suitable to regard a signal as being composed of simpler signals added together. To analyze such a signal, it would therefore be beneficial to determine its harmonic content. The Fourier transform may be regarded as a dot product, similar to the basic ones we know from elementary linear algebra. In the signal processing context we regard the Fourier transform as an operator that maps a signal to orthogonal complex trigonometric base functions. For a continuous signal the transform may be expressed as

$$X(i\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt$$ \hspace{1cm} (2.1)

If the signal $x(t)$ is periodic it is possible to reduce the range of the integral to the period $P$. It should not come as a surprise to the reader that it is sufficient to describe such a signal with a discrete set of Fourier coefficients.

$$X_{FC}[k] = \frac{1}{\sqrt{P}} \int_{a}^{a+P} x(t)e^{-i2\pi\omega t}dt, \ a \in \mathbb{R}$$ \hspace{1cm} (2.2)
where \( \omega_0 = 2\pi/P \). Of course this and later transforms only exist for signals that make the integral or sum converge.

If the harmonic content of a signal is known it is possible to synthesize the signal, possibly after modifications, back to the time domain by adding together weighted complex exponentials. The inverse transform is thus expressed as

\[
x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} X(i\omega)e^{i\omega t} d\omega
\]

(2.3a)

\[
x(t) = \frac{1}{\sqrt{P}} \sum_{n=-\infty}^{\infty} X_{FK}[n]e^{in\omega_0 t}
\]

(2.3b)

So far the discussion has strictly considered signals that are continuous in time. In case the signal is only known at discrete points in time we must rely on the Discrete Time Fourier Transform (DTFT). It may be regarded as a Riemann approximation of the desired integral. The transform pair is defined as

\[
X_T(e^{i\omega T}) = T \sum_{-\infty}^{\infty} x[k]e^{-i\omega kT}
\]

(2.4a)

\[
x[k] = \frac{1}{2\pi} \int_{-\pi/T}^{\pi/T} X_T(e^{i\omega T})e^{i\omega kT}
\]

(2.4b)

where \( T \) is the sample period. Observe that the frequency domain is continuous and that it is normalized to the complex unit circle.

If one would use the DTFT to determine the harmonic content of a real world signal using a computer a major problem would arise, the summation limits. In order to compute the transform exactly all samples since the beginning to the end of time are required. Common practice is to truncate the signal to \( N \) samples and then assume that the signal is periodic with period \( NT \). Although this is seldom perfectly true, we will get good results if \( N \) is large. The following transform pair is known as the Discrete Fourier Transform (DFT).

\[
X[k] = \sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi}{N}nk} = \sum_{n=0}^{N-1} x[n]W_N^{-nk}
\]

(2.5a)

\[
x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]W_N^{nk}
\]

(2.5b)

A detailed discussion and derivation may be found in [2]. \( W_N \) is a root of unity and is often called \textit{twiddle factor} in DSP literature. Note that all twiddle factors have the same magnitude, most often 1, and that there is a fixed angle increment between two consecutive factors. This correlation makes it simple to compute twiddle factors with a regular structure.
2.2 Cooley-Tukey DIT FFT

Most signals will not have the same period as the DFT one would use to analyze it. A direct result is that the energy of interesting frequency components will leak over to adjacent frequencies resulting in a blurred spectrum. Common to all discrete-time transforms is the alias phenomenon, ie frequencies higher than \( f_s/2 = \frac{1}{2T} \), will be folded into the spectrum and will be interpreted as lower frequencies.

Since Fourier transforms are useful in many areas much research have been directly focused on how to efficiently calculating them. An extremely influential article was written by James Cooley and John Tukey in 1965. While working together at a research division at IBM they managed to show that the computational complexity of the DFT could be reduced by several magnitudes by using a divide and conquer approach and clever trigonometric substitutions.

A general distinction between early FFT algorithms is that they either cut the signal into shorter signals in the time domain or in the frequency domain. This is the reason that the abbreviations DIT and DIF are used in FFT contexts. The original algorithm is today known as the Cooley-Tukey DIT FFT. Somewhat later a similar algorithm called Sande-Tukey DIF FFT was developed that decimates the samples in the frequency domain rather than in the time domain. It is possible to further reduce the complexity by processing several samples per calculation. It can be proved that a radix-4 algorithm yields the smallest number of memory accesses. The LeoCore utilize a radix-4 DIF FFT.

Observe that 2.5a on the facing page may be expressed as two sums, one covering the even samples and one covering the odd ones.

\[
X[k] = \sum_{n=0}^{N/2-1} x[2n]W_N^{-2nk} + \sum_{n=0}^{N/2-1} x[2n+1]W_N^{-(2n+1)k}
= \sum_{n=0}^{N/2-1} x_{\text{even}}[n]W_N^{-2nk} + \sum_{n=0}^{N/2-1} x_{\text{odd}}[n]W_N^{-(2n+1)k}
\tag{2.6}
\]

Split the second twiddle factor and use the identity \( W_N^{2nk} = W_N^{nk}W_N^{N/2} \).

\[
X[k] = \sum_{n=0}^{N/2-1} x_{\text{even}}[n]W_N^{-nk}W_{N/2}^{-k} + W_N^{-k} \sum_{n=0}^{N/2-1} x_{\text{odd}}[n]W_N^{-nk}W_{N/2}^{-k}
= X_{\text{even}}[k] + W_N^{-k}X_{\text{odd}}[k], \quad k = 0, 1, \ldots, N - 1
\tag{2.7}
\]

The two transforms will be periodic with period \( N/2 \) so that \( X[k] = X[N/2 + k] \). Half of the coefficients does thus not require computations. The twiddle factor needs to be negated for the upper half since \( W_N^{N/2+k} = -W_N^k \).
Finally

\[
X[k] = X_{\text{even}}[k] + W_N^{-k}X_{\text{odd}}[k] \tag{2.8a}
\]

\[
X[k + N/2] = X_{\text{even}}[k] - W_N^{-k}X_{\text{odd}}[k] \tag{2.8b}
\]

\[
k = 0, 1, \ldots, N/2 - 1
\]

It is important to realize that \(X_{\text{even}}\) and \(X_{\text{odd}}\) are two general DFTs of length \(N/2\). The algorithm may thus be applied iteratively in a backwards fashion until the sums only contain 2 elements each. Since \(k = 0, 1, \ldots, N/2 - 1\) the twiddle factors of the first stage (the last disintegration) will be \(W_2^k = 1\). The transform may be split over \(\log_2 N\) layers, each requiring \(O(N)\) operations so the final computational complexity is \(O(N \log_2 N)\).

It is common to introduce the butterfly diagram to clarify how the computation is executed. Note that it has two nodes at both the input and the output. One of the input nodes shall be multiplied with a twiddle factor and the numbers travelling down the edges shall be added at the outputs. Note that one of the edges must negate its value.

\begin{center}
\includegraphics[width=0.5\textwidth]{butterfly.png}
\end{center}

Figure 2.1. Radix-2 DIT butterflies

### 2.3 Radix-4 DIF FFT

The transform 2.5a on page 6 may be expressed as four consecutive sums.

\[
X[k] = \sum_{n=0}^{N/4-1} x[n]W_N^{-nk} + \sum_{n=N/4}^{N/2-1} x[n]W_N^{-nk} + \sum_{n=N/2}^{3N/4-1} x[n]W_N^{-nk} + \sum_{n=3N/4}^{N-1} x[n]W_N^{-nk} \tag{2.9}
\]
2.3 Radix-4 DIF FFT

Figure 2.2. 32 point radix-2 DIT FFT
Make some substitutions

\[
\begin{align*}
\sum_{n=N/4}^{N/2-1} x[n]W_N^{-nk} &= W_N^{-Nk/4} \sum_{n=0}^{N/4-1} x[n+N/4]W_N^{-nk} = i^{-k} \sum_{n=0}^{N/4-1} x[n+N/4]W_N^{-nk} \quad (2.10a) \\
\sum_{n=N/4}^{3N/4-1} x[n]W_N^{-nk} &= W_N^{-Nk/2} \sum_{n=0}^{N/4-1} x[n+N/2]W_N^{-nk} = (-1)^{-k} \sum_{n=0}^{N/4-1} x[n+N/2]W_N^{-nk} \quad (2.10b) \\
\sum_{n=3N/4}^{N-1} x[n]W_N^{-nk} &= W_N^{-Nk3/4} \sum_{n=0}^{N/4-1} x[n+3N/4]W_N^{-nk} = (-i)^{-k} \sum_{n=0}^{N/4-1} x[n+3N/4]W_N^{-nk} \quad (2.10c)
\end{align*}
\]

add together

\[
X[k] = \sum_{n=0}^{N/4} (x[n] + i^{-k}x[n+N/4] + (-1)^{-k}x[n+N/2] + (-i)^{-k}x[n+3N/4]) W_N^{-nk}
\]
evaluate for \( k = 0, 1, 2, 3 \mod 4 \) and use the identity \( W_N^{4nk} = W_N^{nk} \)

\[
\begin{align*}
X[4k] &= \sum_{n=0}^{N/4-1} (x[n] + x[n+N/4] + x[n+N/2] + x[n+3N/4]) W_N^0W_N^{-nk} \quad (2.12a) \\
X[4k+1] &= \sum_{n=0}^{N/4-1} (x[n] - ix[n+N/4] - x[n+N/2] + ix[n+3N/4]) W_N^{-n}W_N^{-nk} \quad (2.12b) \\
X[4k+2] &= \sum_{n=0}^{N/4-1} (x[n] - x[n+N/4] + x[n+N/2] - x[n+3N/4]) W_N^{-2n}W_N^{-nk} \quad (2.12c) \\
X[4k+3] &= \sum_{n=0}^{N/4-1} (x[n] + ix[n+N/4] - x[n+N/2] - ix[n+3N/4]) W_N^{-3n}W_N^{-nk} \quad (2.12d)
\end{align*}
\]

It is thus possible to describe a DFT of length \( 4^l \), \( l \in \mathbb{Z} \) as four independent \( N/4 \)-point DFTs. Note that the input to each \( N/4 \)-point DFT is a linear combination of four signal samples multiplied by complex numbers \( \{1, W_N^{-n}, W_N^{-2n}, W_N^{-3n}\} \) called twiddle factors. To visualize the computational flow the radix-4 butterfly diagram is introduced. The edges in the graph symbolize multiplications with numbers \( \{1, i, -1, -i\} \), which may be computed trivially by trigonometric identities. Output nodes represent addition and multiplication with twiddle factors.
The LeoCore computes one radix-4 butterfly per cycle so the accelerator must be able to compute four twiddle factors in parallel.

It may not be obvious from the formula, but the impact on computational complexity is huge and very important. An ordinary $N$-point DFT requires $N^2$ complex multiplications. When the sum is split into four, only $4 \cdot (N/4)^2$ multiplications are required, a reduction to 25%. The divide and conquer approach may be repeated iteratively until the DFTs only contain 4 samples. That layer is trivial since all twiddle factors are $W_4^0 = 1$. It’s interesting to note that $W_N^{-nk}$ never needs to be computed, it too is equal to 1 in the last layer. A transform may be split over $\log_4 N - 1$ layers, each requiring $3/4(N/4 - 1)$ non-trivial multiplications. The expression one usually find in the literature is the similar $O(N \log_4 N)$. 

![Figure 2.3. Radix-4 DIF butterflies](image)
Figure 2.4. 16 point radix-4 DIF FFT
The CORDIC algorithm was developed by Volder [7] in 1959 and is an acronym for COordinate Rotation DIgital Computer, which was the name of the first computer system utilizing the algorithm. Apparently it was used in the navigational computer of the American bomber aircraft B-58 Hustler.

In its most simple form the algorithm may either be used to determine the argument of a given vector (vectoring mode) or to rotate the vector to a new angle (rotation mode). Since the only hardware needed for a basic implementation are adders, shifters and tables, the algorithm have been implemented in numerous
The CORDIC algorithm

computer systems, especially those lacking hardware multipliers. During the years following Volder’s report much work was done to develop the algorithm, extending it to new coordinate systems thus making it possible to calculate a large group of functions; exponentials, logarithms and trigonometric functions to mention a few.

In 1971, Walther[8] published the first article on the Unified CORDIC algorithm, elegantly summarizing all extensions so far. Most research since then has been on performance and accuracy. As stated earlier it is possible to operate the algorithm in circular, linear and hyperbolic coordinate spaces but only the circular case is concerned in this report.

3.1 Notation

• \( \hat{x}_i, \hat{y}_i, \hat{z}_i, \hat{u} \) represent values that would appear in a real world CORDIC implementation. Therefore they contain both angle approximation approximation and rounding error that will be dealt with in section 3.4

• \( x_i, y_i, z_i, u_i, \varphi, \varphi_i \) are values from a hypothetical CORDIC machine with infinite word lengths. Hence only the angle approximation error will appear in the result.

• \( u' \) and \( \theta' \) are the target vector and target angle that would be obtained if the rotation was ideal.

3.2 Theory

In order to rotate a vector \( u_0 = [x_0 \ y_0]^T \) by the angle \( \theta' \) one may use the following transformation matrix.

\[
T = \begin{bmatrix}
\cos \theta' & -\sin \theta' \\
\sin \theta' & \cos \theta'
\end{bmatrix}
\]

so that

\[
u' = Tu_0 \iff \begin{cases}
x' = x_0 \cos \theta' - y_0 \sin \theta' \\
y' = x_0 \sin \theta' + y_0 \cos \theta'
\end{cases}
\]

Note that \( \theta' \) is defined positive in counter-clockwise direction.

The identity is rather complex to compute. The straightforward approach would require four multiplications and two black boxes to supply the trigonometric factors. If the hardware is limited we have to transform the expression into something easier to compute. We start by breaking out \( \cos \theta' \).

\[
u' = \cos \theta' \begin{bmatrix} 1 & -\tan \theta' \\ \tan \theta' & 1 \end{bmatrix} u_0
\]

Even though the expression is somewhat cleaner than before since the matrix only contain one trigonometric function, it seems that multiplications must inevitably
be used. A clever trick is to only allow angles whose tangent is a negative power of two, \( \tan(\varphi_i) = \pm 2^{-i} \), effectively reducing all multiplications in the matrix to simple bit shifts!

This is a severe limitation though, since every angle on a circle must be possible to reach. The solution is to use an iterative approach. A rotation may be approximated to arbitrary precision by a series of micro rotations.

\[
\theta' \approx \varphi = \sum_{i=0}^{n-1} \lambda_i \arctan(2^{-i}) = \sum_{i=0}^{n-1} \lambda_i \varphi_i \tag{3.3a}
\]

\[
u' \approx u = \prod_{i=0}^{n-1} \cos \left( \arctan(2^{-i}) \right) \left[ \begin{array}{cc} 1 & -\lambda_i 2^{-i} \\ \lambda_i 2^{-i} & 1 \end{array} \right] u_0 = K_n \prod_{i=0}^{n-1} P_i u_0 \tag{3.3b}
\]

where \( \lambda_i \in \{-1, 1\} \) determines if the micro rotation is clockwise or counterclockwise. The combined scaling effect of the micro rotations can be expressed as a constant, \( K_n \), that only depend on the number of iterations and can thus be precomputed. It may be omitted from the calculations if we take care to either postscale the output vectors or alternatively prescale the input vectors. In every rotation stage, from here called cordic step, we need to perform three add/sub operations.

\[
x_{i+1} = x_i - \lambda_i y_i 2^{-i} \tag{3.4a}
\]

\[
y_{i+1} = y_i + \lambda_i x_i 2^{-i} \tag{3.4b}
\]

\[
z_{i+1} = z_i - \lambda_i \varphi_i \tag{3.4c}
\]

The new variable \( z \) is an angle accumulator and is needed to keep track of how much the vector have been rotated when it reach step \( i \). Every cordic step have an associated angle \( \varphi_i = \arctan(2^{-i}) \) that must be precomputed and stored in a memory. We have yet to decide what should be put in \( z_0 \) and what sign \( \lambda_i \) should have in the cordic steps. Naturally that depend on what we want to achieve. In vector mode one would set \( z_0 = 0 \) and choose \( \lambda_i \) as

\[
\lambda_i = \begin{cases} 
-1 & \text{if } y_i \geq 0 \\
1 & \text{if } y_i < 0
\end{cases} \tag{3.5}
\]

The scheme tries to minimize \( y_n \) by rotating the initial vector towards \( [1 \ 0]^T \). The argument of the initial vector may then be read from \( z_n \). In rotation mode one would set \( z_0 = \theta' \) and

\[
\lambda_i = \begin{cases} 
1 & \text{if } z_i \geq 0 \\
-1 & \text{if } z_i < 0
\end{cases} \tag{3.6}
\]

Interpretation: \( z_i \geq 0 \) means that the argument of the vector is too small, i.e \( \arctan(y_i/x_i) < \varphi \), so that a counter-clockwise micro rotation should bring it closer to \( \varphi \) and \( \theta' \). On the other hand, the argument is too large if \( z_i < 0 \) so that a clockwise rotation will be issued. The final rotated vector may in the end be read
The CORDIC algorithm

Figure 3.2. CORDIC approximation

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{sign } z_i$</th>
<th>$\lambda$-dir</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$+$</td>
<td>CCW</td>
</tr>
<tr>
<td>1</td>
<td>$-$</td>
<td>CW</td>
</tr>
<tr>
<td>2</td>
<td>$+$</td>
<td>CCW</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

from $x_n$ and $y_n$. The rest of the thesis will focus on the rotation mode if nothing else is said.

An example of how the CORDIC approximation works can be seen in figure 3.2. The target vector represented with an arrow is in the first quadrant so a prescaled starting point is chosen on the real axis. By evaluating the equations in rotation mode and noting that the angle is currently too small, it becomes clear that half of $x$ will be added to $y$ and that half of $y$ will be added to $x$. When the equations are evaluated the next time, a clockwise micro rotation will be issued so that half of $y$ is added to $x$ and half of $x$ is subtracted from $y$ etc. Seemingly the algorithm approach the target area in only a few cycles.

### 3.3 Convergence

The maximum angle that can be handled by the simple shift sequence explained above may be expressed as

$$\max(\varphi) = \lim_{n \to \infty} \sum_{i=0}^{n-1} \varphi_i = \lim_{n \to \infty} \sum_{i=0}^{n-1} \arctan(2^{-i}) = 1.743 \ldots \approx 99.9^\circ \quad (3.7)$$

This is generally not a problem since it is sufficient that the algorithm converges for $-\pi/2 \leq \theta' \leq \pi/2$. If the desired vector has an argument outside this region it is easy to choose an initial vector from $\{(1,0), (0,1), (-1,0), (0,-1)\}$ effectively reducing the required convergence range to $\pm \pi/2$.

So far we have only showed a possible way to choose the direction of the micro rotations and how to handle the growth of the vector magnitude. We have yet to
3.3 Convergence

prove that it really is possible to approximate an arbitrary angle to the required precision. The vector will be rotated in all stages since \( \lambda_i = \pm 1 \forall i \), even if the angle is perfectly correct. To achieve a satisfactory approximation we need to show that the remaining micro rotations are capable of bringing the vector sufficiently close to the target, ie \( \varphi_k - \sum_{j=k+1}^n \varphi_j \leq \varphi_n \).

The proof is adapted from [6] and [5]. The first step is to prove a general convergence theorem for decreasing sequences.

**Theorem 3.1 (Convergence Theorem)** Let \( \varphi_0 \geq \varphi_1 \geq \ldots \geq \varphi_n > 0 \) be a decreasing sequence of positive numbers satisfying

\[
\varphi_i \leq \varphi_n + \sum_{j=k+1}^n \varphi_j, \text{ for } 0 \leq i < n. \tag{3.8a}
\]

and that there exists a real number \( r \) such that

\[
|r| \leq \sum_{j=0}^n \varphi_j. \tag{3.8b}
\]

Define the sequence \( s_0 = 0 \) and \( s_{i+1} = s_i + \lambda_i \varphi_i, \) \( i = 0, 1, \ldots, n \) where

\[
\lambda_i = \text{sign}(r - s_i) = \begin{cases} 
1 & \text{if } r \geq s_i \\
-1 & \text{if } r < s_i 
\end{cases} \tag{3.8c}
\]

Then

\[
|r - s_i| \leq \sum_{j=i}^n \varphi_j + \varphi_n, \text{ for } 0 \leq i \leq n \tag{3.8d}
\]

and especially \( |r - s_{n+1}| \leq \varphi_n \)

**Proof** The theorem is proved by induction on \( i \), ie we show that it is true for a specific case and that if it is valid for \( i \) then it is also valid for \( i + 1 \). For \( i = 0 \) we have

\[
|r - s_0| = |r| \leq \sum_{j=0}^n \varphi_j \leq \sum_{j=0}^n \varphi_j + \varphi_n \tag{3.9}
\]

Consider \( |r - s_{i+1}| = |r - s_i - \lambda_i \varphi_i| \). We know from 3.8c that \( \lambda_i \) and \( (r - s_i) \) always have the same sign and therefore \( |r - s_i - \lambda_i \varphi_i| = ||r - s_i| - \varphi_i| \). Rewrite 3.8a

\[
- \left( \varphi_n + \sum_{j=i+1}^n \varphi_j \right) \leq -\varphi_i \leq |r - s_i| - \varphi_i \tag{3.10}
\]

and assume that the theorem is true for \( i \)

\[
|r - s_i| - \varphi_i \leq \varphi_n + \sum_{j=i}^n \varphi_j - \varphi_i = \left( \varphi_n + \sum_{j=i+1}^n \varphi_j \right). \tag{3.11}
\]
The CORDIC algorithm

Figure 3.3. \( \varphi_i = \arctan 2^{-i} \)

Combine and evaluate the inequalities

\[
\begin{align*}
|r - s_i| - \varphi_i & \geq - (\varphi_n + \sum_{j=i+1}^{n} \varphi_j) \\
|r - s_i| - \varphi_i & \leq (\varphi_n + \sum_{j=i+1}^{n} \varphi_j)
\end{align*}
\]

\( \Rightarrow ||r - s_i| - \varphi_i|| = |r - s_{i+1}| \leq \varphi_n + \sum_{j=i+1}^{n} \varphi_j \) (3.12)

The theorem is thus valid since it’s proved to be correct for \( 0, i \) and \( i + 1 \). Especially

\( -\varphi_n \leq |r - s_n| - \varphi_n \leq 2\varphi_n - \varphi_n = \varphi_n \) and thus \( |r - s_{n+1}| = ||r - s_n| - \varphi_n| \leq \varphi_n \). □

If we make sure that the angle sequence associated with the micro rotations comply with the requirement 3.8a for some \( n \) we may finally conclude that the CORDIC algorithm will rotate a vector to any given angle (possibly with prerotation).

**Theorem 3.2** For \( n > 3 \) the sequence \( \varphi_i = \arctan 2^{-i} \), \( i = 0, 1, \ldots, n \) satisfy the hypothesis 3.8a of the Convergence Theorem for \( |r| \leq \pi/2 \).

**Proof** Plot inspection reveals that the sequence \( \varphi_i = \arctan 2^{-i} \), \( i = 0, 1, \ldots, n \) is a decreasing sequence of positive numbers. Recall the Mean Value Theorem:

\[
f'(c) = \frac{f(b) - f(a)}{b - a}, \quad a < c < b \quad \text{(MVT)}
\]

Apply it to \( \arctan x \)

\[
\frac{\arctan b - \arctan a}{b - a} = \frac{1}{1 + c^2}
\]

let \( a = 2^{-(i+1)} \) and \( b = 2^{-i} \) so that the relative distance \( b - a = 2^{-(i+1)} \) and

\[
\frac{1}{1 + c^2} < \frac{1}{1 + a^2} = \frac{1}{1 + 2^{-2(i+1)}} = \frac{2^{2(i+1)}}{1 + 2^{2(i+1)}} \quad \text{(3.14)}
\]
3.4 Error analysis

Hence,

$$\phi_i - \phi_{i+1} = (b - a) \frac{1}{1 + c^2} \leq 2^{-(i+1)} \frac{2^{2(i+1)}}{1 + 2^{2(i+1)}} = \frac{2^{i+1}}{1 + 2^{2i}} \quad (3.15)$$

Now let $a = 0$ and $b = 2^{-i}$

$$\frac{1}{1 + c^2} > \frac{1}{1 + b^2} = \frac{2^i}{1 + 2^i} \quad (3.16)$$

$$\phi_i \triangleright \frac{b}{1 + c^2} \geq 2^{-i} \frac{2^i}{1 + 2^i} = \frac{2^i}{1 + 2^i} \quad (3.17)$$

Combine the inequalities using a telescoping series.

$$\phi_i - \phi_n = (\phi_i - \phi_{i+1}) + (\phi_{i+1} - \phi_{i+2}) + \cdots + (\phi_{n-1} - \phi_n) = \sum_{j=i}^{n-1} (\phi_j - \phi_{j+1}) \quad (3.18a)$$

$$\leq \sum_{j=i}^{n-1} \frac{2^{j+1}}{1 + 2^{2(j+1)}} = \sum_{j=i+1}^{n} \frac{2^j}{1 + 2^{2j}} \leq \sum_{j=i+1}^{n} \phi_j \quad (3.18b)$$

We have thus showed that

$$\phi_i \leq \phi_n + \sum_{j=i+1}^{n} \phi_j, \quad \text{for } 0 \leq i < n. \quad (3.19)$$

Examine the beginning of the sequence:

$$\arctan 1 + \arctan \frac{1}{2} + \arctan \frac{1}{4} + \arctan \frac{1}{8} \approx 1.618 > \pi/2. \quad (3.20)$$

Clearly

$$|r| \leq \frac{\pi}{2} < \sum_{j=0}^{3} \arctan 2^{-j} < \phi_n + \sum_{j=0}^{n} \phi_j. \quad (3.21)$$

Apparently it is sufficient with $n > 3$ micro rotations to cover a whole quadrant with predictable, although not very good, accuracy. \qed

### 3.4 Error analysis

The reasoning in the previous section was analytical in its nature. Surely it is possible to achieve arbitrary precision if the numeric quantities may be represented exactly, but what happens when the algorithm is implemented in a digital computer? Problems arise since a digital computer necessarily stores numbers with a precision given by the internal word length of that machine.

Walther\[8\] noted that the introduced errors are bounded and a number of extra bits should be enough to keep them insignificant. Hu\[3\] determined that the major
error sources are angle approximation errors and rounding errors and he managed to place tight bounds on their influence. Kota and Cavallaro[4] investigated the matter further and added the effects of finite word length in the z-datapath. The angle approximation error was discussed in section 3.3. There we showed that the error could be made arbitrarily small. We still need to look into how the actual output is affected. The rounding error is due to the finite precision of the arithmetic operations and storage registers in a real implementation.

A thorough analysis is required in order to choose a critical design parameters such as word length and number of micro rotations to achieve a given accuracy.

3.4.1 Angle approximation error

We know from section 3.3 that the CORDIC algorithm may be used to approximate a requested vector \( u' \) with \( u_n \) to an accuracy determined by the smallest available micro angle. Therefore the angle approximation error \( \delta = \theta' - \varphi \) will be bounded as \( |\delta| \leq \varphi_{n-1} \). With knowledge of \( \delta \) it would be possible to compute the correct vector as

\[
\begin{pmatrix}
\cos \delta & -\sin \delta \\
\sin \delta & \cos \delta
\end{pmatrix}
\begin{pmatrix}
u_n
\end{pmatrix}
\]

(3.22)

If we introduce an identity matrix we may determine boundaries on the relative approximation error.

\[
\begin{align*}
\mathbf{u}' - \mathbf{u}_n &= (\mathbf{D} - \mathbf{I})\mathbf{u}_n \\
\frac{\|\mathbf{u}' - \mathbf{u}_n\|}{\|\mathbf{u}_n\|} &\leq \|\mathbf{D} - \mathbf{I}\|
\end{align*}
\]

(3.23a)

(3.23b)

The \( l_2 \)-norm \( \|\mathbf{A}\|_2 \) may be calculated as \( \sqrt{\lambda_{\text{max}}(\mathbf{A}^*\mathbf{A})} \) where \( \lambda \) are the eigenvalues and \( \mathbf{A}^* \) is the conjugate transpose of \( \mathbf{A} \).

\[
\|\mathbf{D} - \mathbf{I}\| = \sqrt{(\cos \delta - 1)^2 + \sin^2 \delta} = \sqrt{2 - 2 \cos \delta}
\]

\[
= \sqrt{2 - 2 (1 - 2 \sin^2(\delta/2))} = 2 \sin(\delta/2) \leq |\delta|
\]

(3.24)

It may not be obvious at first sight, but the last step in the derivation is actually rather simple. If we evaluate \( \sin(\delta/2) \) in the origin we find that it’s value is 0 and its right hand derivative is 1, which happens to be its maximum slope. The inequality must thus be true since \( \delta \) grows faster in all other points.

Finally a tight bound has been established for the difference between the optimal result \( \mathbf{u}' \) and the output from an ideal CORDIC-machine, \( \mathbf{u}_n \).

\[
\|\mathbf{u}' - \mathbf{u}_n\| \leq |\delta| \|\mathbf{u}_n\|
\]

(3.25)

For a complete analysis of the approximation error in a real CORDIC computer we need to express \( |\delta| \) in terms of the machine’s precision. Finite precision in all
real datapaths will inevitably introduce truncation errors, i.e., the data is quantized.
Introduce the quantization operator.

\[ Q[a] = a - e \quad (3.26) \]

The error \( e \) depends on the precision of the datapath of \( a \). Depending on which representation that is used, integer or fractional, the truncation error will be bounded by 1 or \( 2^{-b} \) respectively where \( b \) is the effective word length of the datapath. In later derivations we will call the maximum truncation error in the datapaths \( \varepsilon_{xy} \) and \( \varepsilon_z \).

The following two lemmas deal with the effects of finite precision in the \( z \)-datapath.

**Lemma 3.1**

\[ \left| \theta' - \sum_{i=0}^{n-1} \lambda_i Q[\varphi_i] \right| = \hat{\varepsilon}_n \leq Q[\varphi_{n-1}] \quad (3.27) \]

**Proof** This is a variation of the convergence theorem derived earlier. In a real implementation all angles \( \varphi_i \) will be replaced by truncated versions, \( Q[\varphi_i] \). □

**Lemma 3.2**

\[ \left| \varphi - \sum_{i=0}^{n-1} \lambda_i Q[\varphi_i] \right| \leq n\varepsilon_z \quad (3.28) \]

**Proof** If fixed point arithmetic is used then all truncation errors will be bounded by \( \varepsilon_z \). The accumulated error after \( n \) iteration will consequently be bounded by \( n\varepsilon_z \). □

The triangle inequality allows us to relate the difference of two values through a third one.

\[ |a - b| \leq |a - c| + |b - c| \quad (3.29) \]

Use it to estimate the worst case angle error.

\[ |\delta| = |\theta' - \varphi| \leq |\theta' - \sum_{i=0}^{n-1} \lambda_i Q[\varphi_i]| + |\varphi - \sum_{i=0}^{n-1} \lambda_i Q[\varphi_i]| \]
\[ \leq Q[\varphi_{n-1}] + n\varepsilon_z \quad (3.30) \]

### 3.4.2 Rounding error

Reexamine 3.3b on page 15 and realize that \( u_i \) will be scaled with \( \frac{1}{\cos(\arctan 2^{-i})} = \sec \varphi_i \) if we multiply it with \( P_i \). A more convenient expression may be obtained with some trigonometry. Since \( \varphi_i = \arctan(2^{-i}) \) in the triangle of figure 3.4 on the following page it is easy to identify \( \cos(\arctan 2^{-i}) \). We have thus showed that the magnitude of the vector will be scaled with \( k_i = \sqrt{1 + 2^{-2i}} \) in iteration \( i \).
The CORDIC algorithm

\[ \sqrt{1 + 2^{-2i}} \]

\[ \varphi_i = \arctan 2^{-i}. \]

Figure 3.4. A helping triangle. \( \varphi_i = \arctan 2^{-i} \).

Every time we want to store a number in a computer, an error may be introduced due to the finite precision of the machine. Since the CORDIC algorithm is iterative, errors from previous stages will be scaled with \( k_i \) and propagate further down the datapath. Recall that quantities in real datapaths wear hats.

\[ \hat{u}_i = u_i + e_i \quad (3.31) \]

Observe that a stored vector consists of both the true vector \( u_i \) and the quantization error \( e_i = [e_{x_i}, e_{y_i}]^T \). If a fixed point machine is used, the errors will be equally bounded and the worst case error may be determined. Hence

\[ \|e_i\| = \sqrt{e^2_{x_i} + e^2_{y_i}} \leq \sqrt{2} \varepsilon_{xy}. \quad (3.32) \]

The following derivations assume that the precision is good enough so that \( P_i \) may be represented exactly for all \( i \). Assuming that \( \hat{u}_0 = u_0 + e_0 \) one has

\[ \hat{u}_1 = P_0 \hat{u}_0 = u_1 + P_0 e_0 + e_1 \quad (3.33a) \]
\[ \hat{u}_2 = P_1 \hat{u}_1 = u_2 + P_1 P_0 e_0 + P_1 e_1 + e_2 \quad (3.33b) \]

We can see that the error will also be multiplied and thus rotated and scaled by \( P_i \). The final result, after \( n \) stages will be

\[ \hat{u}_n = P_{n-1} \hat{u}_{n-1} + e_n \]
\[ = u_n + \sum_{j=0}^{n-1} \prod_{i=j}^{n-1} P_i e_j + e_n \quad (3.34) \]

And the total rounding error is thus

\[ \hat{u}_n - u_n = \sum_{j=0}^{n-1} \prod_{i=j}^{n-1} P_i e_j + e_n \quad (3.35) \]

The worst-case scenario is that the maximum possible quantization error \( \varepsilon_{xy} \) is introduced in every iteration. The upper bound for the total quantization error is
3.4 Error analysis

thus

\[ \| \hat{u}_n - u_n \| \leq \| e_n \| + \left\| \sum_{j=0}^{n-1} \prod_{i=j}^{n-1} P_i e_j \right\| \leq \| e_n \| + \sum_{j=0}^{n-1} \left\| \prod_{i=j}^{n-1} P_i \right\| \| e_j \| \]

\[ \leq \sqrt{2} \varepsilon_{xy} \left[ 1 + \sum_{j=0}^{n-1} \prod_{i=j}^{n-1} k_i \right] = \sqrt{2} \varepsilon_{xy} f(n) \quad (3.36) \]

Where

\[ f(n) = \left[ 1 + \sum_{j=0}^{n-1} \prod_{i=j}^{n-1} \sqrt{1 + 2^{-2i}} \right] \quad (3.37) \]

Observe that the expression is valid only if the input vectors are prescaled, so that all stages rotate and scale errors. If the system manages scaling after rotation, the sum will start from \( j = 1 \) instead.

3.4.3 Total error

The approximation error and the truncation error may be combined into an expression that describes the worst-case total error.

\[ u' - \hat{u}_n = (u' - u_n) + (u_n - \hat{u}_n) \quad (3.38a) \]

\[ \| u' - \hat{u}_n \| \leq \| u' - u_n \| + \| u_n - \hat{u}_n \| \]

\[ \leq \| u_n \| | \theta' - \varphi | + \sqrt{2} \varepsilon_{xy} f(n) \quad (3.38b) \]

\[ \leq \| u_n \| (Q[\varphi_{n-1}] + n\varepsilon_z) + \sqrt{2} \varepsilon_{xy} f(n) \quad (3.38c) \]

Two error bounds were computed from 3.38 since the angle error \( \delta = | \theta' - \varphi | \) could be estimated either as \( \varphi_{n-1} \) or more pessimistically as \( Q[\varphi_{n-1}] + n\varepsilon_z \). Figure 3.5 and 3.6 show the theoretical error bounds for an implementation with 14 bit output coefficients. The data have been scaled to match the integer representation used in the plots of appendix C. Figure C.1 show the errors extracted from a hardware simulation.

Both graphs give a rather pessimistic picture of the performance but observe that the limits give the errors in the absolutely worst case scenarios. Rigorous testing of different configurations will thus be necessary to find a suitable solution with a reasonable hardware cost.
Figure 3.5. Predicted worst case error ($\delta = \varphi_{n-1}$). 14 bit output.

Figure 3.6. Predicted worst case error ($\delta = Q[\varphi_{n-1}] + n\varepsilon_z$). 14 bit output.
Chapter 4

Implementation

In order to keep the complexity of the unit on a fathomable level, the design has been divided into many smaller blocks arranged in a hierarchy. A series of CORDIC steps together with a prerotation stage is considered to be a CORDIC pipeline. Four pipelines are grouped together with output logic to a unit called CORDIC rotator. To facilitate communication with the host processor and keep track of FFT layers and sizes, NCO angles and control which pipes that should keep their angles etc a control machinery with two finite state machines (FSM) are required. One FSM monitors requests from the complex network and keeps track on which pipeline that carry the next value to be delivered in NCO mode. Depending on its current state and the request it will send a proposal to the other FSM that decide which pipelines that should keep their angles and which that should advance. In most cases the proposal will be followed, but if the pipeline is not full or if a new FFT layer is to be initiated special clocking might be needed. The machines will be called CNW FSM and FFT NCO FSM from now.

Note that the figures in the following sections are not supposed to be perfect representations of the actual code implementation. Their purpose is rather to show how data flows through the unit and which signals controls it. Therefore most of the signal widths are not stated explicitly. Do note that all signals widths in the datapath may be configured easily by changing parameters in a package.

The asynchronous reset logic and the clock have been omitted on purpose and all registers are considered to be clocked by the main clock if nothing else is indicated. The RTL code assumes that registers have enable signals. Most often they are logic combinations of the signals steering the multiplexers in the figures. Adding keeper signals from the register output to the input of the multiplexer will give the correct behaviour for the reader.

Fixed point representation and 2's complement arithmetic have been used throughout the design. Since the precision and range of such numbers are limited we need to extend word lengths in the data path in both ends. Target angles close to multiples of $\pi/2$ will bring one of the coordinates very close to its maximum value.
Overflow is fatal since the paths of $x$ and $y$ are interconnected. Therefore the dynamic range is extended by using one guard bit inside the CORDIC pipes to allow momentary overflows. Possible overflow in output from the final stage will lead to rounding and truncation. Another concern is errors resulting from limited precision. They may be kept small enough by extending the lower end of all numeric quantities. Error bounds were derived in section 3.4 but word lengths will ultimately be chosen by simulation.

4.1 Package generator

VHDL is a strongly typed language and is a bit cumbersome to use for designs where component instantiations and table sizes are to be decided upon simulation or synthesis. An elegant solution is to store all parameters in a separate package and include it in all files that need them. Then only that package need to be altered to choose a new configuration. Since the data for the tables would be generated in another language, I found it reasonable to write the whole package in that same language. C++ was chosen since free compilers exist for next to all computer system and all necessary mathematical operators for the table generation are included in the standard libraries.

4.2 Accelerator interface

The LeoCore of Coresonic AB contains a couple of different networks for exchanging data between the core, memories, accelerators etc. The FFT NCO accelerator of this thesis exchange parameters and status information via the Control Register File (CRF) and complex data through the Complex Network (CNW).

4.2.1 Control Register File

The registers of the CRF are distributed between the core and connected peripherals. It offers a simple way to exchange control and status information as well as limited amounts of data. All peripherals share the read/write signal as well as address and data buses. Since the outputs of all peripherals are connected to common OR-gates, it is crucial that all units that are not addressed output zeros only. Each register contains 16 bits. Currently five registers are reserved for the accelerator whose base address is set by a generic.

**Control and status:** This register is used to exchange orders and status information between the accelerator and the core. Bit 3:0 bits store the size of the FFT transform in $\log_2$; 0 means NCO mode and 1 will not trigger the accelerator. At the moment two different magnitudes are offered by toggling bit 4. Finally the core may detect that the pipes are filled and ready to deliver complex numbers by checking the ready flag stored in bit 5.
4.3 CORDIC step

Increment HI & LO: Two registers are used to store the current NCO increment. It may be changed at any time so that an adaptive algorithm may be used to detect optimal increment.

Angle HI & LO: These registers store the angle of the next vector to be generated. If the accelerator is needed for another task, it is possible to store the contents of these registers and resume operation seamlessly at a later time.

4.2.2 Complex Network

The complex network is as the name implies used to transfer complex-valued data. A value consist of a 16 bit imaginary part and an equally long real part. Up to four values may be concatenated and transferred in each cycle. It is trivial to assemble the output vector if the accelerator is used in FFT mode since the order of the values is already fixed in the specification. NCO mode on the other hand puts some stress on the interface logic and target angle computation. For example, if the program demands 3 values to be sent 2 times and then 2 values to be sent 3 times, then the pipes should be mapped to the output vectors in the following order:

012-, 301-, 23--, 01--, 23--, ...

Obviously a state machine is required to keep track of which pipe that carry the next value to be put leftmost in the output vector and how many pipes that should generate new values in the next cycle. Additionally a careful startup procedure is required in order to make the pipes produce the correct values.

4.3 CORDIC step

Figure 4.1. A general CORDIC step
According to section 3.2 we need to perform a number of operations, called micro rotations, either iteratively in the same hardware or in a pipeline of similar steps. The latter solution was chosen so that a high throughput would be achieved easily.

\[
\begin{align*}
x_{i+1} &= x_i - \lambda_i y_i 2^{-i} \\
y_{i+1} &= y_i + \lambda_i x_i 2^{-i} \\
z_{i+1} &= z_i - \lambda_i \varphi_i
\end{align*}
\]

The hardware in figure 4.1 on the previous page allows \( x_i \) and \( y_i \) to be shifted down an arbitrary number of steps by passing \( i \) as a generic in the entity port. They are then added to or subtracted from the non-shifted numbers depending on the sign of \( z_i \). The sign of \( z_i \) is thus in control of the direction of the micro rotation. A positive value gives a counterclockwise rotation. Care has been taken to get the introduced truncation error unbiased. This is accomplished by inserting the most significant outshifted bits into the adder-subtracters as input carries. For this to work the adder-subtracters must realize the function \( a \pm (b + c) \). Note that \( a \) and \( b \) are vectors while \( c \) is a single bit.

Two versions of the simple CORDIC step were written, with the only difference that one of them use registers on the outputs of \( x_{i+1}, y_{i+1} \) and \( z_{i+1} \). Observe that the critical path through a stage is through one adder/subtractor.

### 4.4 Prerotation stage

![Figure 4.2. CORDIC prerotation stage](image)
The purpose of the prerotation stage is to make sure that a correct target angle is fed to the CORDIC pipeline at all times. In both FFT and NCO mode the angles should start from a given value and then be increased in fixed steps. An accumulator structure with multiple options for start angles and increments can be seen in 4.2 on the facing page.

Some logic is necessary to determine the quadrant of the desired vector and choose a good starting point for the following pipeline. This is absolutely necessary because the original CORDIC algorithm does not converge for $|\theta'| \geq 90^\circ$. With pre-rotation however, all angles are acceptable. When the rotator is reset or restarted, the register containing $\theta'$ is loaded with either `nco_init` if NCO numbers are to be generated or 0 if twiddle factors are requested. The decision is based on `fft_size_r` and "0000" corresponds to NCO mode. An increment is chosen and added depending on the the FFT size either from a lookup table or from `nco_incr`.

With a clever choice of number representation for $\theta'$ it is surprisingly easy to determine the target quadrant. If `theta_r` is considered to store a number in $[-\pi \pi)$ it is sufficient to look only at the two most significant bits. When the target angle is computed and stored in the register, start coordinates may be chosen for $x_0$ and $y_0$ with the aid of `quad` and `mag`. The angle of the inserted vector is well known and is subtracted from $\theta'$ giving $z_0$. `mag` set the target magnitude to either 1 or $1/\sqrt{2}$.

Even though the precision of the output vectors may not be very high per element it is desirable that the internal precision of $\theta'$ is much greater since the unit may be set to work in NCO mode for long periods of time. With a high precision in all signals related to $\theta'$, distortions due to truncation of the target angle will be kept small. For this design all such signals have a word length of 32 bits. An obvious consequence is that `nco_init` and `nco_incr` require two slots each in the register file.

### 4.5 CORDIC rotator

The CORDIC rotator assembles $n$ CORDIC steps together with a prerotation stage into a CORDIC pipeline. It is important to keep the critical paths short so that the design may use a fast clock to get high throughput. The package generator contains a parameter that allow the designer to decide the ratio of CORDIC stages with and without buffered outputs.

Four of those are then instantiated. Some of the control signals from the accelerator top entity are shared among the pipelines while the others need to be split up. The top three vectors in figure 4.3 on the next page decide what the pipelines should do in the next clock period. The options are:

- Compute the next angle.
- Keep the current angle stored in the accumulator. At the output it will seem that the same vector comes out multiple times.
• Reset the angle accumulator. This is used when the unit is reset and when a new FFT layer is initiated.

In NCO mode the programmer has the option to request 0-4 vectors to be delivered every clock cycle and naturally they should be put on the complex network in rising order. This puts some extra stress on the logic that concatenates the results. The last state of the CNW FSM, o_state_cnw, is used to determine which pipe that holds the value with the smallest angle. Since the angles increase with nco_angle_incr from pipe to pipe it is easy to create the output vector according to the scheme in table 4.1.

<table>
<thead>
<tr>
<th>o_state_cnw</th>
<th>acc_cnw_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>next in 0</td>
<td>pipe3 &amp; pipe2 &amp; pipe1 &amp; pipe0</td>
</tr>
<tr>
<td>next in 1</td>
<td>pipe0 &amp; pipe3 &amp; pipe2 &amp; pipe1</td>
</tr>
<tr>
<td>next in 2</td>
<td>pipe1 &amp; pipe0 &amp; pipe3 &amp; pipe2</td>
</tr>
<tr>
<td>next in 3</td>
<td>pipe2 &amp; pipe1 &amp; pipe0 &amp; pipe3</td>
</tr>
</tbody>
</table>

Table 4.1. Data order in acc_cnw_data.

If FFT mode is active then the output order is always the same since four values
will be delivered in all cycles. The specific order depends on the host system that use the accelerator. It is trivial to determine the current mode of the accelerator since the present state of the FFT NCO FSM is available.

### 4.6 Control and interface

The top entity of the accelerator is very advanced and it is outside the scope of this report to describe it fully since that would require a full listing of the HDL code. I will therefore describe what the control machinery does rather than how it is built.

First of all it contains two finite state machines. The FFT NCO FSM is in charge of applying a sequence of next/keep-angle directives to the CORDIC rotator. This is not a trivial task since special care is required to fill the pipes properly, especially in NCO mode. There are four special states reserved for this. By starting the pipes one by one it is guaranteed that they will produce coefficients with proper angle increments.

When operating in FFT mode it is crucial to keep track of the current layer and signal lengths. Remember from section 2 that every layer but the last in a transform increases the number of signals in a regular fashion. If the first layer consists of 1 signal of length $N$ samples, then the second will be composed of 4 signals of length $N/4$, the third layer will have 16 signals of length $N/16$ and so forth. Every signal in a layer requires the same set of coefficients and there are two ways to manage this. One way is to transform one signal at a time and thus
revolve the unit circle multiple times for coefficients. Another way is to generate the same coefficients for as many cycles as there are signals in the current layer. The latter solution was chosen since that was the order demanded by the CMAC unit that performed the butterfly operations. Power dissipation will probably be lower this way since the CORDIC pipes will be static for long periods in the final layers.

A set of counters and comparators aid the FFT NCO FSM. They inform when it is time to update the prerotation stages with new target angles. When a new layer is initiated, the prerotation stages will be instructed to reset their angle accumulators and start using new increments that suit the signal size of the new layer.

![Diagram](image)

**Figure 4.5.** CNW FSM

The interface to the CNW specifies that 0-4 values can be requested in a cycle. Naturally they should be concatenated to a long output vector in an order with increasing angles. By using a clever state representation which describes which pipe that will have the smallest angle in the next cycle and jump between those depending on requests from CNW it is possible to decide what pipes that should be clocked and how their output should be stitched together.
Chapter 5

Simulation and Synthesis

There are essentially three important design parameters that need to be considered when the accelerator is instantiated.

- $n$ is the number of iterations to be performed for a vector rotation. One can see that it influence all parts of the total error (3.38) since it determines the angle of the last stage as well as the number of stages that will propagate and amplify truncation errors.

- $b_{xy}$ and $b_z$ are the internal word lengths of the CORDIC pipelines. A long word length usually correspond to smaller errors, which will be apparent later.

- A parameter that will affect hardware cost but not precision is the number of CORDIC steps between pipeline registers. The latency will be longer with many registers but that shouldn’t be a problem since the unit will probably operate for thousands of cycles when it has been filled. Two steps per register gave good results in synthesis both for silicon and FPGA.

A designer will have to select a combination that gives a good enough precision without requiring too much area and power. Shorter word lengths give cheaper designs in those terms so they need to be kept on a minimum level. The design space is huge so an exhaustive test of all combinations would not be feasible. Literature and papers on the CORDIC algorithm suggest that the number of iterations should at least be equal to the number of bits in the output and that about $\log_2 n$ extra bits should be used to keep truncation errors small in the data path of $\hat{x}$ and $\hat{y}$. Little is said however about how many bits that are required for $\hat{z}$. When the tables containing the micro angles $\varphi_i$ were generated it was apparent that $b_z$ need be larger than or equal to $n + 1$ to ensure that $Q[\varphi_{n-1}] > 0$, which place a lower limit on $b_z$. Another common variant is to let $b_{xy}$ and $b_z$ to be equal. The latter variant seems to be used most often so it was used to generate the plots.
Note that points in the plots of this chapter are missing. If the registers in the data path are short in comparison with the length of the path, then the results of the shift operation in late stages will give only zeros. Affected stages will not be able to micro rotate input vectors and such design is considered inferior.

### 5.1 FFT SNR

A 4096-point test signal resembling a DVB-T broadcast in QPSK mode was generated using a software model. Matlab was used to apply an FFT transform to be used as reference. The spectrum generated by Matlab may be seen in figure 5.1. Later the same signal was transformed by an assembler program using the accelerator in different configurations. The spectrum from the accelerator was then subtracted from the reference so that the noise induced by the accelerator could be inspected. Figure 5.2 shows an appearance that is typical for most configurations. It is interesting to note that the noise is *coloured* meaning that the spectrum is not even.

Signal to Noise Ratio (SNR) was then estimated by dividing the signal energy by the error energy. Butterfly operations are conducted in a unit called CMAC. At the moment a precision of 14 bits are used in the multipliers which limits the sensible output precision of the accelerator. Graph B.1 to B.3 display the FFT SNR for some shorter output word lengths as well.

### 5.2 Twiddle Factor precision

A simple assembler program was written to make the accelerator produce twiddle factors for FFT transforms of different sizes. Matlab was then used to compute reference values for comparison. Graph C.1 to C.5 show a couple of different error measures recorded from a 4096 point transform. Output from the accelerator was rounded to 14 bits and was interpreted as integers in all tests, ie the value of the LSB was 1. A number of histograms were generated so that it is possible to see how the errors are distributed. Only output from pipe 1 to 3 were regarded since pipe 0 always output a correct number. The histograms were generated by computing the absolute error in the output of the pipes and distributing them in 15 bins. It is interesting to note that while the maximum error surely decreases with longer word lengths, the typical error grows. Increasing the number of iterations does not seem to improve performance very much. The histograms are located in appendix D.

### 5.3 Synthesis for silicon

Design Compiler from Synopsys was used for silicon synthesis in 65 nm technology. Cycle time was set to 4 ns which correspond to a clock frequency of 250 MHz.
5.3 Synthesis for silicon

Figure 5.1. Matlab FFT

Figure 5.2. FFT error
Hardware cost for some configurations are plotted in figure E.1 to E.3
Chapter 6

Conclusion and discussion

The main conclusion of this thesis work is that cheap configurations are sufficient for targeted applications. Section 3.4 showed us that a large number of iterations results in a small angle approximation error. This does not necessarily mean that the total error will be small since truncation errors will propagated and amplified in more computational stages.

Even though the error plots of appendix C show rather large errors for some configurations, we know from the histograms of appendix D that the expectation value most often lie well below the worst case. Another fascinating observation is the fact that the ‘hump’ tends to move towards a larger typical error when the internal word length is increased. This may explain why the SNR is not improved very much. Also note that the actual precision curves match the predicted error curves in the sense that only a few extra bits are needed in the data path except for a guard. Additional bits will increase gate count without significant performance improvement. It seems to be sufficient to keep the number of iterations slightly above the number of output bits in order to get the best ratio between performance and area cost.

Figure 5.2 show that generated spectrums will have rather large errors for low frequencies. This means that the SNR will be improved if this issue is sorted out. It is also apparent that 14 bit output precision is not absolutely necessary, at least not for 4096 point signals. A configuration that give 13 output bits might save some thousand gates.
Chapter 7

Future Work

The report have touched a number of different subjects that could be investigated further.

- It should be fairly easy to rewrite the top entity and the prerotation stages to support approximation of other elementary functions and other modes of operation.

- It is straightforward to make it possible for the programmer to decide the magnitude of the initial vectors so that the final vectors can be tuned further.

- Investigate other shift and add algorithms such as the more advanced BKM [1].

- Further investigate the reason why the induced noise is coloured and what overall effects it has on signal processing. The origin is probably not the FFT NCO accelerator but rather the CMAC unit.

- The quality of the output is good but not perfect. More time could be spent trying to track down eventual mistakes or find a reason in the algorithm.

- I synthesized the accelerator together with the LeoCore but I did not have time to test it on FPGA. It would be interesting to continue and write firmware for some current standards and see that the unit works in a real environment.
Bibliography


Appendix A

Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMAC</td>
<td>Complex Multiply and Accumulate</td>
</tr>
<tr>
<td>CORDIC</td>
<td>Coordinate Rotation Digital Computer</td>
</tr>
<tr>
<td>CCW</td>
<td>CounterClockWise</td>
</tr>
<tr>
<td>CW</td>
<td>ClockWise</td>
</tr>
<tr>
<td>CNW</td>
<td>Complex Network</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>DIF</td>
<td>Decimation In Frequency</td>
</tr>
<tr>
<td>DIT</td>
<td>Decimation In Time</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>DTFT</td>
<td>Discrete Time Fourier Transform</td>
</tr>
<tr>
<td>DVB</td>
<td>Digital Video Broadcasting</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>FT</td>
<td>Fourier Transform (continuous)</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>NCO</td>
<td>Numerically Controlled Oscillator</td>
</tr>
<tr>
<td>OFDM</td>
<td>Orthogonal Frequency Division Multiplexing</td>
</tr>
<tr>
<td>RF</td>
<td>Register File</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
</tr>
</tbody>
</table>
Appendix B

FFT SNR plots

Figure B.1. Signal to Noise ratio for 12bit output
Figure B.2. Signal to Noise ratio for 13bit output

Figure B.3. Signal to Noise ratio for 14bit output
Appendix C

Twiddle factor precision plots

Data in the plots were collected from simulated configurations with an output word length of 14 bits.

\[ b_{xy} = 16 \]
\[ b_{xy} = 17 \]
\[ b_{xy} = 18 \]
\[ b_{xy} = 19 \]
\[ b_{xy} = 20 \]
\[ b_{xy} = 21 \]

**Figure C.1.** Absolute error: \( \max |W_{acc} - W_{ref}| \)
Figure C.2. Magnitude error: $\max|W_{\text{acc}}| - |W_{\text{ref}}|$

Figure C.3. Angle error in degrees: $\max|\arg(W_{\text{acc}}) - \arg(W_{\text{ref}})|$
Figure C.4. X error: max|\Re(W_{acc}) - \Re(W_{ref})|

Figure C.5. Y error: max|\Im(W_{acc}) - \Im(W_{ref})|
Appendix D

Absolute error histograms

out  output bits
s   number of iterations
xyz internal bits for xyz

Figure D.1. Absolute error out14s15xyz16
Figure D.2. Absolute error out14s15xyz18

Figure D.3. Absolute error out14s15xyz20
Figure D.4. Absolute error output

Figure D.5. Absolute error output
Figure D.6. Absolute error out14s16xyz20

Figure D.7. Absolute error out14s17xyz16
Figure D.8. Absolute error out14s17xyz18

Figure D.9. Absolute error out14s17xyz20
Appendix E

Synthesis for silicon plots

Figure E.1. Equivalent number of gates for 12bit output
Figure E.2. Equivalent number of gates for 13bit output

Figure E.3. Equivalent number of gates for 14bit output
Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

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

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

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

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

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

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

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

© Anton Andersson