2008

Real-Time Hilbert Transform and Autocorrelation for Digital Wideband Communication Applications

Murthy, Dilip Srinivasa
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Electrical and Computer Engineering Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/258

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
REAL-TIME HILBERT TRANSFORM AND AUTOCORRELATION FOR DIGITAL WIDEBAND COMMUNICATION APPLICATIONS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering

By

DILIP SRINIVASA MURTHY
B.E, Electronics & Communication, Visvesvaraya Technological University, INDIA

2005

2008
WRIGHT STATE UNIVERSITY
WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

November 13, 2008


___________________________
Chien-In Henry Chen, Ph.D.
Thesis Director

___________________________
Kefu Xue, Ph.D.
Department Chair

Committee on Final Examination

___________________________
Chien-In Henry Chen, Ph.D.

___________________________
Yan Zhuang, Ph.D.

___________________________
Raymond E. Siferd, Ph.D.

___________________________
Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies
Abstract


Digital wideband receiver plays a key role in communication and radar systems, and it is used to detect and analyze a wide range of radio frequency (RF) signals. A digital Instantaneous Frequency Measurement (IFM) receiver is designed, where implementation of a real-time architecture of a Hilbert transform and autocorrelation algorithm is presented. Real-time architecture performs continuous monitoring and processing of the incoming signal. Prior to the previous one-bit design, the input data fed to the Hilbert transform is 8-bit wide at a clock rate of 320 MHz with the input data rate increased by a factor of eight, an 8-bit autocorrelation algorithm is designed to maintain the real-time data processing for signal detection. This new digital IFM design was implemented and tested on Delphi's ADC3255 (PCM digitizer with Xilinx Virtex-4 FPGA), and has capable of sorting the continuous and short pulse wave signal up to 1.2 GHz bandwidth with a frequency detection error less than 2 MHz for time resolution of 100 nsec.
# Table of Contents

## I. INTRODUCTION
1.1 Wideband Receiver........................................................................................................ 1  
1.2 Motivation.................................................................................................................... 2  
1.3 Research Approach.................................................................................................... 3  
1.4 Testing and Analysis................................................................................................. 3  
1.5 Document Organization............................................................................................ 4  

## II. DESIGN CONSIDERATION
2.1 Basic principle of operation...................................................................................... 5  
2.2 Hilbert Transform...................................................................................................... 8  
2.3 Delay Lines............................................................................................................... 13  
2.4 IFM Autocorrelation Algorithm............................................................................... 15  
2.5 Frequency mapping.................................................................................................. 20  

## III. DESIGN PLATFORMS
3.1 Xilinx Virtex-4 FPGA............................................................................................. 21  
3.2 Delphi ADC............................................................................................................. 23  
3.3 Xilinx System Generator......................................................................................... 24  
3.4 System Generator Design Flow................................................................................ 25  

## IV. DESIGN IMPLEMENTATION
4.1 Design Overview....................................................................................................... 28  
4.2 Elimination of DC Biasing...................................................................................... 29  
4.3 Real-Time Hilbert Transform................................................................................... 31  
   4.3.1 Basic Architecture............................................................................................... 31  
   4.3.2 Real-Time Implementation.................................................................................. 33  
4.4 Autocorrelation Algorithm..................................................................................... 36  
4.5 Reset Control Logic................................................................................................. 38  
4.6 In Phase and Quadrature Phase Autocorrelators..................................................... 40  
4.7 Crossing Clock Domains......................................................................................... 42  
4.8 Phase Detection....................................................................................................... 42  
4.9 Frequency Calculation............................................................................................. 46  
4.10 Threshold and Detection Variable.......................................................................... 47  

## V. EXPERIMENTAL AND SYNTHESIS RESULTS
5.1 Xilinx System Generator Results............................................................................... 49  
5.2 Xilinx ISE Synthesis Results................................................................................... 56  
5.3 Xilinx ChipScope Pro Verification........................................................................... 58  

## VI. CONCLUSION AND FUTURE WORK
6.1 Conclusion................................................................................................................. 63  
6.2 Future Work............................................................................................................... 63  

REFERENCES.................................................................................................................. 64
## List of Figures

<table>
<thead>
<tr>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Basic Analog IFM receiver</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Practical IFM receiver</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Hilbert Transform functionality</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Hilbert Transform waveform</td>
<td>10</td>
</tr>
<tr>
<td>2.5 Impulse Response of FIR Filter</td>
<td>12</td>
</tr>
<tr>
<td>2.6 FIR Filter Operation</td>
<td>12</td>
</tr>
<tr>
<td>2.7 Correlator Setup with ‘n’ number of delay lines</td>
<td>14</td>
</tr>
<tr>
<td>2.8 IFM Autocorrelation Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>2.9 Relationship between phase and frequency with four correlators</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Xilinx virtex-4 FPGA with Atmel ADC</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Delphi ADC3255 Top Level Schematic Diagram</td>
<td>23</td>
</tr>
<tr>
<td>3.3 Design Flow of the Xilinx System Generator</td>
<td>26</td>
</tr>
<tr>
<td>4.1 Digital IFM Design Overview</td>
<td>28</td>
</tr>
<tr>
<td>4.2 Gateway In Schematic and Properties</td>
<td>29</td>
</tr>
<tr>
<td>4.3 System Generator Implementation of DC biasing setup</td>
<td>30</td>
</tr>
<tr>
<td>4.4 Schematic Representation of One-Bit Hilbert Transform</td>
<td>31</td>
</tr>
<tr>
<td>4.5 Schematic Representation of Digitizer block</td>
<td>32</td>
</tr>
<tr>
<td>4.6 Schematic Representation of Real-Time Hilbert Transform</td>
<td>34</td>
</tr>
<tr>
<td>4.7 System Generator Implementation of Real-Time Hilbert Transform</td>
<td>35</td>
</tr>
<tr>
<td>4.8 Schematic diagram of Autocorrelation Algorithm</td>
<td>36</td>
</tr>
<tr>
<td>4.9 System Generator implementation of one of the 8-bit autocorrelation algorithm</td>
<td>37</td>
</tr>
<tr>
<td>4.10 System Generator Implementation of Reset Control Logic</td>
<td>39</td>
</tr>
<tr>
<td>4.11 Output Waveform of Counter(top) and Relation Block (bottom)</td>
<td>39</td>
</tr>
<tr>
<td>4.12 System Generator Implementation of In phase Autocorrelators</td>
<td>41</td>
</tr>
<tr>
<td>4.13 System Generator Implementation of Quadrature Phase Autocorrelators</td>
<td>41</td>
</tr>
<tr>
<td>4.14 System Generator Representation of Cross Clocking Domains</td>
<td>43</td>
</tr>
<tr>
<td>4.15 Hardware Implementation of Cross Clocking Domains</td>
<td>43</td>
</tr>
<tr>
<td>4.16 System Generator Implementation of Phase Detection block</td>
<td>44</td>
</tr>
<tr>
<td>4.17 System Generator Implementation of Phase LUT with 5:1 Muxes</td>
<td>45</td>
</tr>
<tr>
<td>4.18 System Generator Implementation of Frequency Calculation</td>
<td>46</td>
</tr>
<tr>
<td>4.19 System Generator Implementation of Detection Variable</td>
<td>48</td>
</tr>
<tr>
<td>5.1 System Generator Result of Hilbert Transforms I(top) and Q(bottom) data</td>
<td>49</td>
</tr>
<tr>
<td>5.2 System Generator Result of In phase Autocorrelators values for $f_c = 320\text{MHz}$</td>
<td>50</td>
</tr>
<tr>
<td>5.3 System Generator Result of Quadrature Autocorrelators values for $f_c = 320\text{MHz}$</td>
<td>51</td>
</tr>
<tr>
<td>5.4 System Generator Result of Phase Detection Block for $f_c = 320\text{MHz}$</td>
<td>52</td>
</tr>
<tr>
<td>5.5 System Generator Result of Frequency and Detection Flag</td>
<td>53</td>
</tr>
<tr>
<td>5.6 System Generator Output of Chirp Signal with -10dB noise</td>
<td>54</td>
</tr>
</tbody>
</table>
5.7 System Generator detected Frequency with -10dB noise (40 – 1240 MHz).................. 54
5.8 System Generator result of Detection variable.............................................................. 55
5.9 Xilinx Project Navigator Design Summary................................................................. 56
5.10 Xilinx Floorplanner of IFM Implemented Design....................................................... 57
5.11 Xilinx ChipScope pro and Vertex-4 FPGA Workstation Setup .................................. 58
5.12 Xilinx ChipScope Pro 8.2i Output for Test Frequency of 40 MHz............................... 59
5.13 Xilinx ChipScope Pro 8.2i Output for Test Frequency of 320 MHz............................ 60
5.14 Xilinx ChipScope Pro 8.2i Output for Test Frequency of 1240 MHz......................... 61
5.15 IFM receiver average error with 3,600 samples......................................................... 62
List of Tables

Table                    Page
2.1 Standard h (n) values................................................................. 11
4.1 Eleven-Tap FIR block input and output Pattern ............................ 32
4.2 Phase Modification base on signs of I and Q................................. 46
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off The Shelf</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>ECCM</td>
<td>Electronic Counter-Countermeasures</td>
</tr>
<tr>
<td>ECM</td>
<td>Electronic Countermeasures</td>
</tr>
<tr>
<td>ESM</td>
<td>Electronic Support Measure</td>
</tr>
<tr>
<td>EW</td>
<td>Electronic Warfare</td>
</tr>
<tr>
<td>$f_c$</td>
<td>Carrier Frequency</td>
</tr>
<tr>
<td>$f_s$</td>
<td>Sampling Frequency</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-In, First-Out</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FT</td>
<td>Fourier Transform</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>I</td>
<td>In phase</td>
</tr>
<tr>
<td>IF</td>
<td>Intermediate Frequency</td>
</tr>
<tr>
<td>IFM</td>
<td>Instantaneous Frequency Measurement</td>
</tr>
<tr>
<td>LUT</td>
<td>Lookup Table</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>PMC</td>
<td>PCI Mezzanine Card</td>
</tr>
<tr>
<td>PRI</td>
<td>Pulse Repetition Interval</td>
</tr>
<tr>
<td>PW</td>
<td>Pulse Width</td>
</tr>
<tr>
<td>Q</td>
<td>Quadrature phase</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Language</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very-High-Speed Integrated Circuit Hardware Description Language</td>
</tr>
<tr>
<td>XSG</td>
<td>Xilinx System Generator</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Henry Chen, Professor, Electrical Engineering Department, Wright State University, for his valuable suggestions, guidance and encouragement throughout my study period. Dr. Chen who came up with the project idea, has provided numerous insights and guidance throughout my thesis work.

I am obliged to Yu-heng George Lee and Ryan Thomas Bone for their help in the lab in course of obtaining results.

I want to thank my friends for all their help and constant support. I would also like to thank the staff of the Electrical Engineering Department for their co-operation. Finally, I express my sincere gratitude to Dr. Siferd and Dr. Zhuang for their willingness to serve as members of my thesis defense committee.
For my parents,

Srinivasa Murthy and Shantha Kumari
I. INTRODUCTION

1.1 Wideband Receivers

Wideband receivers are capable of monitoring number of channels simultaneously. Hence they are more suitable for detecting the signals that are spread across several channels and also present for short duration. There are number of wideband receivers which can be used in searching for communications transmitters, these receivers basically include a channelized receiver, compressive receiver and digital receiver. These types of receivers are also used in searching the other emitters such as radar [7].

Some of the examples of wideband receivers are as follow

- **Crystal video receiver**: This type of receiver consists of a tuning stage to select the band of interest followed by a detector. Multiple receivers are often used in parallel to monitor a number of bands.

- **Instantaneous Frequency Measurement (IFM) receiver**: The IFM receiver allows the frequency measurement of pulsed signal, but can only respond to one signal at a time. The sensitivity of IFM receivers tends to be low because of the inherent high signal-to-noise ratio required to measure frequency.

- **Bragg cell receiver**: An instantaneous measure of the spectrum of received signals can be obtained using a Bragg cell receiver. This type of receiver employs a surface-acoustic-wave device to perform a Fourier transform. The output may be coupled via photodiodes and then
to a charge-coupled device (CCD) that performs time averaging. The output from the CCD can then be displayed or read by a computer [7].

1.2 Motivation

The IFM receiver is one of attractive wideband receivers due to wide range of instantaneous RF bandwidths, where instantaneous input bandwidth can achieve 16 GHz (from 2 to 18 GHz) frequency measurement accuracy which is capable of high sensitivity and fine resolution. This is the type of receiver that can measure frequency accurately to 1MHz on a 100 nsec short pulse width (PW) and it is also appealing in terms of size, weight and cost.

Although IFM receivers are unable to process simultaneous signals, they are often used in conjunction with a form of frequency selectivity in front of the IFM receiver, limiting or eliminating the probability of intercepting simultaneous signals [3]. Generally, for wide band applications the IFM receiver is just one component of a hybrid receiving configuration [3].

This thesis contributes to an advancement of the existing IFM receiver. A one-bit digital IFM design was designed but it was not capable of handling all input data at real-time. Hence a real-time architecture is needed to overcome this problem. In this thesis a new 8-bit Hilbert Transform and Autocorrelation architecture are designed which operate at 320 MHz and handles all input data at real-time.

1.3 Research Approach

The goal of this thesis is to design, implement and test a real-time Hilbert transform and Autocorrelation design in Digital IFM receiver specified for wideband receiver applications.
In this thesis, first and foremost the mathematical theory and the function of the digital IFM design were presented. The one-bit Hilbert transform and autocorrelation were analyzed and a new design approach was presented to build a real-time design.

The theoretical design was constructed using MATLAB’s Simulink environment and then Xilinx’s System Generator (XSG). With the design thoroughly tested and the behavioral simulations completed, synthesis and timing-based simulation was performed to verify if the design would run as expected with the chosen parameters in the targeted FPGA.

1.4 Testing and Analysis

Initially, a real-time IFM design was developed and verified in an ideal environment using Xilinx platform, the complete design was synthesized and verified for area utilization, timing and power consumptions using Xilinx ISE version 8.2i. A complete design kit taken from Xilinx ISE was uploaded into Chip Scope Pro version 8.2i for real-time verification on the FPGA. The testing of the FPGA was performed using a Xilinx Virtex-4 board model XC4VSX55, and an onboard Atmel 10-bit ADC. Using a RF signal generator the real-time design is verified for the frequency range from 40 MHz to 1.24 GHz.

1.5 Document Organization

This thesis document is organized as follows. Chapter I presents a brief introduction to wideband receiver followed by motivation of this research. Chapter II discusses with the basic design methodology and working principle of the IFM receiver and its various components including Hilbert Transform, autocorrelation algorithm, phase detection and frequency measurement blocks. Chapter III introduces a design platform including the field
programmable gate array (FPGA), Analog-to-Digital convertor (ADC), the Xilinx System
Generator (XSG) software package, and also the System Design Flow. In Chapter IV, design
implementation of IFM receiver using Xilinx System Generator is discussed and Chapter V
presents the simulation results from XSG and Chip Scope pro. Finally, Chapter VI concludes
my work and research.
II. DESIGN CONSIDERATIONS

2.1 Basic Principle of Operation

A conventional IFM Receiver takes the input signal and splits it into two paths, one with a constant delay of $\tau$ and the other as the normal input; which intern creates a phase angle $\theta$ between the two signals and is given by the equation:

$$\theta = 2\pi f_o$$  \hspace{1cm} (2.1)

Where: $f_o$ is the frequency of the input signal. By determining the phase difference $\theta$ between the two signals, frequency of the input signal can be calculated [3].

A basic IFM receiver shown in the Figure 2.1, Under ideal conditions let us assume it employs analog components such as hybrid power divider, a delay line, a phase correlator, four diode detectors with a low pass filter and a couple of Differential amplifiers.

![Figure 2.1: Basic Analog IFM receiver](image)
The Delay line produces the necessary time delay of $\tau$. The Phase correlator provides a proper shift between delayed and non-delayed lines. The correlator shifts and combines these signals in several combinations as follows:

$$A1 = \cos[2\pi f_o (t - \tau)] - \sin[2\pi f_o t] \quad (2.2)$$
$$A2 = \sin[2\pi f_o (t - \tau)] - \cos[2\pi f_o t] \quad (2.3)$$
$$B1 = \sin[2\pi f_o (t - \tau)] + \sin[2\pi f_o t] \quad (2.4)$$
$$B2 = -\cos[2\pi f_o (t - \tau)] + \cos[2\pi f_o t] \quad (2.5)$$

By introducing the square-law detector and its frequency domain properties, the diode detectors will convert the four correlated signals into video signals by squaring the output of the low pass filters to provide the following equations:

$$C1 = 1 - \sin[2\pi f_o \tau] \quad (2.6)$$
$$C2 = 1 + \sin[2\pi f_o \tau] \quad (2.7)$$
$$D1 = 1 + \cos[2\pi f_o \tau] \quad (2.8)$$
$$D2 = 1 - \cos[2\pi f_o \tau] \quad (2.9)$$

The final signal processing stage consists of two differential amplifiers which yields the following signals:

$$E1 = 2 \sin[2\pi f_o \tau] = 2 \sin \theta \quad (2.10)$$
$$E2 = 2 \cos[2\pi f_o \tau] = 2 \cos \theta \quad (2.11)$$

The phase angle $\theta$ is given by

$$\theta = \tan^{-1} \left( \frac{E1}{E2} \right) = 2\pi f_o \tau \quad (2.12)$$

Where: $f_o$ is the input frequency of the receiver.

However converting radio frequency into video signals destroys the carrier frequency and signal phase information. This design is just one possible way to determine the input
frequency. The above results can be derived from other combinations of power dividers which are used to get the delayed signal, different hybrids to provide a required phase shift, limiting amplifiers to improve sensitivity of the RF signal and various mixers which can be replace the diode detector and differential amplifiers. [3]

The design of a practical IFM receiver consists of several parallel simple IFM receivers, the designs may use \( n \) number of delay lines. More delay line leads to more hardware and also introduce phase ambiguities but provide fine frequency resolution which in turn increases the accuracy of the detected signal. A practical IFM receiver with power divider, four delay lines, phase correlator and diode detectors is shown in Figure 2.2.

![Figure 2.2: Practical IFM receiver](image-url)

The IFM design in this thesis, the design is purely digital in nature, but the basic principles are based on the same fundamental theory of analog IFM receiver. The RF signals are digitized instead of getting converted into video signals. The design shown in this work
directly digitizes the input signal without using a crystal detector, power dividers, analog delay lines, hybrids or mixers. This approach utilizes small area and faster processing of the correlation variables and the overall signal; and also the power consumption is less compared to the conventional approach [3].

### 2.2 Hilbert Transform

The receiver receives the signal through the antenna, the RF signal received from the antenna is down converted to the intermediate frequency (IF). The IF is then passed through the high speed ADC. The next stage is to convert two output channels that have a 90 degree phase shift from the input of the carrier frequency (fc), which is real and imaginary and then combined together to form a complex signal These two signals are referred to as In-Phase (I) and Quadrature Phase (Q) channels which will be referred the same way throughout the thesis. In this thesis, these two forms of channels are obtained by using the Hilbert transform [8]. A Hilbert transform block is shown in the Figure 2.3.

![Hilbert Transform Functionality](image)

**Figure 2.3 – Hilbert Transform Functionality**

Hilbert Transform of a function $x(t)$ is defined as the convolution of $x(t)$ and a function $h(t)$. The relation can be shown mathematically in the following equation:
\[ H[x(t)] = x^h(t) = x(t) * h(t) = x(t) * \frac{1}{\pi t} \]  \hfill (2.13)

\[ = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{x(\tau)}{t-\tau} d\tau \]  \hfill (2.14)

Where: \( * \) represents convolution and \( H[x(t)] \) and \( x^h(t) \) represent a Hilbert Transform in time domain. The function \( h(t) \) is defined as

\[ h(t) = \frac{1}{\pi t} \]  \hfill (2.15)

Also, The Hilbert Transform can be represented in frequency domain as

\[ X^h(f) = X(f)H(f) \]  \hfill (2.16)

The Fourier transform of \( h(t) \) can be determined as

\[ F[h(t)] = H(f) = j \text{sgn}(f) \]  \hfill (2.17)

\[ = -j \begin{cases} 1 & f > 0 \\ -1 & f < 0 \end{cases} \]  \hfill (2.18)

Where \( \text{sgn} \) represents the sign function. Thus in frequency domain, the negative frequency of \( X(f) \) is multiplied by \( j \) and positive frequency is multiplied by \( -j \). Note that the term \([j \text{sgn}(f)] \) which implies passing \( x(t) \) through the system that leaves the magnitude of \( X(f) \) unchanged, while the phase is shifted by \( \frac{\pi}{2} \) for positive frequency and \( -\frac{\pi}{2} \) for the negative frequency [ref text]. If we consider an input square wave signal \( \sin(2\pi f_i t) \), then the Hilbert transform would be \(-\cos(2\pi f_i t)\), an exponential waveform. A MATLAB plot is obtained by using an inbuilt function known as \( y = \text{hilbert}(x) \) which computes the Hilbert transform for a real input sequence, \( x \) and returns a complex result of same length, where the real part of \( y \) is the original, real data and imaginary part is the actual Hilbert transform. ‘\( y \)’ is sometimes called the analytical signal, in reference to the continuous-time analytic signal. A key property of the discrete-time analytic signal is that its z-transform is ‘0’ on the
lower half of the unit circle. The magnitude of the analytic signal is the complex envelope of
the original signal. The Hilbert transform is related to the actual data by a 90° phase shift;
sines become cosines and vice versa. To plot a portion of data (solid line) and its Hilbert
transform (dotted line) the fig() show an example of Hilbert transform (red), an exponential
waveform which is 90° phase shifted from the square wave input signal (blue).

Figure 2.4: Hilbert Transform Waveform

In implementing a practical design, one has to consider the discrete time domain format of
the Hilbert transform. The function $H(f)$ is extended from $-\infty$ to $\infty$ in the frequency domain.
In the discrete domain the number of data points is limited to say, $2M + 1$ points, which is
similar to adding a rectangular window to the input signal. The z-transform of $h(nt_s)$ is
given as:
\[ H(z) = \sum_{n=-M}^{M} h(nt_s)Z^{-n} \]  

Instead of

\[ H(z) = \sum_{n=-\infty}^{\infty} h(nt_s)Z^{-n} \]  \hspace{1cm} (2.20)

These two equations are non-causal in nature as the summation begins from a negative value. Hence, in order to implement the filter in practice the above equations must be made causal, by using the finite impulse response filter (FIR) design scheme such that a discrete Hilbert transform can be achieved.

Since the value of \(n\) in \(h(nt_s)\) of equation 2.20 ranges from \(-M\) to \(M\), a shift in time is required in order to make it causal which is equivalent to multiplying the first result in Equation 2.20 by \(Z^{-M}\), and substituting with \(K = n + M\). Thus,

\[ H(z) = \sum_{n=-M}^{M} h(n T)Z^{-(n+M)} = \sum_{k=0}^{2M} h(kT - MT)Z^{-k} \]  \hspace{1cm} (2.21)

The above equation is causal since the value of \(k\) starts from 0 instead of a negative number. For a practical approach let us assume \(M = 5\) so that 5 data points are discarded from the beginning, and the end of the output can be considered as the steady state of the filtered output. Consider a 11-lag filter to perform Hilbert transform, which corresponds to \(2M + 1 = 11\). The corresponding \(n\) range is from -5 to 5. The co-efficient values are listed in the following table [1].

<table>
<thead>
<tr>
<th>(n)</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h(n))</td>
<td>-2/5(\pi)</td>
<td>0</td>
<td>-2/3(\pi)</td>
<td>0</td>
<td>-2/(\pi)</td>
<td>0</td>
<td>2/(\pi)</td>
<td>0</td>
<td>2/3(\pi)</td>
<td>0</td>
<td>2/5(\pi)</td>
</tr>
<tr>
<td>New (n)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2.1: Standard \(h(n)\) values
For the design the chosen co-efficient are \([6 0 10 0 30 0 30 0 10 0 6]\), And the impulse response of the FIR filter is as shown in Figure 2.5.

![Impulse Response of FIR Filter](image)

Figure 2.5: Impulse Response of FIR Filter

The Direct form representation of the filter from the above table is shown in the Figure2.6.

![FIR Filter Operation](image)

Figure 2.6: FIR Filter Operation
In general, if the input is a pulse of \( n \) data points and the filter has \( m \) data points, the output will have \( n + m - 1 \) data points and among them there are \( n - m \) points in steady state.

### 2.3 Delay Lines

In an IFM receiver, the length of delay lines must be very accurate. If not, the frequency calculated from the variables could be incorrect. Delay line with constant time \( \tau \) is used together with a correlator to obtain the auto correlation block, where in the auto correlation block variables are processed to obtain the desired input signal frequency.

The number of delay lines depends on the frequency range that has to be calculated and the accuracy of the detecting frequency. If two delay lines are considered to calculate the results, the shorter time delays are used to resolve the phase (frequency) ambiguities but limited by the bandwidth of the receiver and the longest time delays provide fine-frequency resolution but add phase ambiguities which are phase differences that exceed \( 2\pi \text{ rad} \). If the bandwidth of the receiver is \( B \), then the shortest time delay is given by

\[
\tau_{\text{short}} = \frac{1}{B} 
\]  

(2.22)

Consider a practical IFM receiver with an input frequency range of 2-6 GHz, frequency resolution of 1.25 MHz and minimum pulse width of 100ns. Thus, 12 bits are required to encode the frequency information. In Figure 2.7, a ‘n’ delay line design is shown. Consider \( n=2 \), the first delay line \( \tau_1 \) which is shorter than the second delay line \( \tau_2 \) has a small error of 1.25 MHz/bit and the other with a large noise error of 61.25MHz/bit which is unsatisfactory [3].
Normally, more than two delay lines will be considered in designing an IFM receiver. Consider the Figure 2.7 with four different delay lines. If the longest delay time is assumed to be 50ns, which represents an ambiguous frequency of 20MHz, then using 4 bits to represent 20 MHz will produce the necessary 1.25 MHz frequency resolution and other 8 bits are declared by the remaining delay lines. More than four delay lines designed can be used to reduce the sensitivity of the IFM receiver to noise; however, the additional delay lines add complexity and cost to the receiver design. Accordingly it may not be possible to add more delay lines in the present thesis and hence there has to be a tradeoff between the complexity and the frequency resolution [3].

From the previous project of one bit IFM receiver design, the delay lines were chosen to be $\tau = [1, 2, 4, 8, 16, 32, 64, 128]$ which correspond to 0.391, 0.781, 1.563, 3.125, 6.25, 12.5, 25 and 50 nanoseconds respectively. However, it was discovered that same performance was achieved using MATLAB with only five delay lines, thus eventually reducing the hardware
by close to 1/3\textsuperscript{rd}. And the newly implemented delay lines were chosen to be $\tau = [1, 2, 8, 32, 128]$. For a bandwidth of 2.56 GHz, the shortest delay equates to 0.391ns as stated, the longest delay value is 128 and it should be shorter than the minimal pulse width which the receiver is supposed to measure. In this case, the minimum pulse width is 100 ns and the longest delay is 50ns. The longest delay line is shorter than the minimum pulse width such that every delay line overlaps in time. Any value of $\tau$ greater than 256 will result in non-overlapping of the delay line resulting in erroneous frequency calculation [8].

2.4 IFM Autocorrelation Algorithm

In this thesis, the input signal is digitized from the Hilbert Transform and the two signals, I and Q with 90° phase difference are generated. The IFM receiver uses an algorithm to process the correlation variables from the Hilbert transform. If we assume the input to RF receiver is a sinusoid with slow varying amplitude then the complex signal can be described as:

$$x(n\Delta t) = a(t)e^{jn\omega \Delta t}$$

(2.23)

and the conjugate as:

$$x(n\Delta t)' = a(t)'e^{jn\omega \Delta t}$$

(2.24)

Where $\omega$ is the signal frequency, the amplitude of the signal is $a(t)$, and the sampling interval is denoted as $\Delta t$. Letting $N$ equate the number of data points received with $m$ sample delays, the complex autocorrelation can be described as [1]

$$S(m\Delta t) = \sum_{n=1}^{N} x(n\Delta t)'x((n - m)\Delta t)$$

(2.25)
\[
= \sum_{n=1}^{N} a(n\Delta t)' e^{-j\omega \Delta t} a((n - m)\Delta t) e^{j(n-m)\omega \Delta t}
\]

\[
= e^{-j\omega \Delta t} \sum_{n=1}^{N} a(n\Delta t)' a((n - m)\Delta t)
\]  

\[
\approx e^{-j\omega \Delta t} \sum_{n=1}^{N} |a(n\Delta t)|^2
\]

\[
= C_m e^{-j\omega \Delta t}
\]

Where: \(m\omega \Delta t\) is the phase of \(S(m\Delta t)\) and \(C_m\) is a constant. \(m\omega \Delta t\) is a linear function of \(\omega\) and the slope of the function is \(m\Delta t\). Since the deviation of amplitude during \(m\Delta t\) is negligible, the equation 2.29 can be approximated.

If we allow \(k\) complex autocorrelations with \(k\) different delays, then the detection variable can be defined as:

\[
D = \sum_{m} S(m\Delta t)' \times S(m\Delta t) = \sum_{m} C_m^2
\]

Where: \(m\) is equivalent to all \(k\) delays. The detection variable can be used to compare with a to-be-determined threshold based on numerical simulations to differentiate the signal from the noise, mitigating the number of false alarms [8].

The I and Q pairs are combined to form complex numbers. In this thesis, the design is concentrated on sampling frequency, \(f_s\) as 1/2.56GHz and the number of data points, \(N\) is chosen as 256 to meet the speed and accuracy of the current design requirements. This algorithm is applicable to any sampling frequency and any number of data points. In this IFM receiver, five autocorrelation variables with five different delays \([m = 1, 2, 8, 32, 128 ]\) are used. The general form of an autocorrelation variable \(S_m\) with input \(x(n)\) and delay of \(m\) is given by:
\[ S_m = \sum_{n=1}^{256} x(n)\, x(n - m) \quad (2.31) \]

The five complex autocorrelation variables are as shown.

\[ S_1 = \sum_{n=1}^{256} x(n)\, x(n - 1) \quad (2.32) \]

\[ S_2 = \sum_{n=1}^{256} x(n)\, x(n - 2) \quad (2.33) \]

\[ S_8 = \sum_{n=1}^{256} x(n)\, x(n - 8) \quad (2.34) \]

\[ S_{32} = \sum_{n=1}^{256} x(n)\, x(n - 32) \quad (2.35) \]

\[ S_{128} = \sum_{n=1}^{256} x(n)\, x(n - 128) \quad (2.36) \]

If a sequence of \( N \) ones and zeroes is represented by \( S_n \) and \( P_n \), which are the non-delayed and delayed data respectively, then correlation between them can be realized in the hardware as the following equation:

\[ 2 \sum_{n=1}^{N} S_n \otimes P_n - N \quad (2.37) \]

Where: \( \otimes \) represents the XNOR operations. The following Fig is replicates the above equation in hardware schematic form.
Let us consider an example, where $N$ is of 10 data points, $S_n = [1,1,1,0,0,1,0,0,1]$ and $P_n = [1,0,1,0,1,1,0,0,1]$. Once the data is passed through the XNOR block, it is collected in the accumulator and for the given $S_n$ and $P_n$ values the accumulator value will be 8 which is multiplied with 2 to give a value 16 and then summed with $-N$ resulting in a value 6. This is one bit operation and in an 8-bit design, autocorrelation is performed individually on each bit of the 8 bit $S_n$ and $P_n$ data, the results are added together to get a 8 bit data. The amplitude of the digitized input is constant due to the digitization of the signals to one bit and hence the phase of the autocorrelated signals is linear corresponding to the input signal frequency. The delay between the two signals can be directly related and the phase between the delayed signals is due to the linearity property between them.
Consider four correlators with individual delays of $\tau_1, \tau_2, \tau_3$ and $\tau_4$ as shown in Figure 2.9, where $\tau_1$ is the shortest delay and $\tau_4$ is the longest delay. In Figure 2.9, section b shows the linear relationship between phase and frequency where the delay $\tau_4$ has the deepest phase-frequency curve ($\omega \tau_4$) and it also illustrates the wrapping around attributes of phase at $2\pi[8]$. 

Figure 2.9: Linear relationship between phase and frequency with four correlators
2.5 Frequency mapping

The data from the Hilbert transform is processed through the autocorrelation block and one bit complex data, once every batch of 256 bits of data are collected. The phase of each of the signals i.e., from S1, S2, S8, S32 and S128 for both I and Q is given by

\[ \phi = \tan^{-1} \left( \frac{Q}{I} \right) \]  

(2.38)

The signal with the minimum delay S1 is first used to calculate the input frequency, but due to the noise in the system the estimation of the input frequency from this data may not be accurate, The Precise result from only one correlator is not possible. Therefore, the results from the signal S1 of the calculated frequency is used to determine the frequency zone of the S2 and also directs where should map its phase further. Similarly S2 will guide the signal S8 by determining its zone for S32, and S32 will further determines the zone of S128, by this kind of mapping will lead to more accurate calculations of required frequency and noise effects can also be minimized. The frequency of S1 is determined by

\[ f_1 = \frac{f_s \Theta_1}{2m\pi} \]  

(2.39)

Where: \( \theta \) is the Phase, \( m \) is the no. of sample delays, for this case \( (m = 1) \) and \( f_s \) is the sampling frequency. The Zone is then calculated by using the following formula:

\[ Z_m = \text{floor} \left( \frac{mf_{m-1}}{f_s} \right) \]  

(2.40)

Where: \( Z_m \) is Zone number and \( f_m \) is the mapped frequency. The Floor function rounds the value of \( \frac{mf_{m-1}}{f_s} \) to the nearest integer towards negative infinity. Finally the mapped frequency, \( f_m \), for \( m > 1 \), i.e. for S2, S8, S32, and S128 is obtained from:

\[ f_m = \frac{f_s}{2\pi} \left( \frac{\Theta_m}{m} + Z_m \frac{2\pi}{m} \right) \]  

(2.41)
III. DESIGN PLATFORMS

3.1 Xilinx Virtex-4 FPGA

A field-programmable gate array (FPGA) is a device consisting of an array of uncommitted elements that can be interconnected in a general way; the interconnections between the elements are user-programmable. FPGA consists of logic blocks which are in the form of 2-input NAND gates, other blocks have more complex structure, such as multiplexers, lookup tables, encoder/decoder or any mathematical functions. FPGAs can be used effectively in a wide variety of applications. FPGAs have lower prototype costs and shorter production times. FPGAs compared to ASIC have relatively low speed of operations and lower logic density (the amount of logic that can be implemented in a single chip). The propagation delay of an FPGA is adversely affected by the inclusion of programmable switches, which have a significant resistance and capacitance in the connections between logic blocks.

The xilinx virtex-4 FPGA is used in the implementation of the IFM receiver in this thesis. Xilinx virtex-4 devices incorporate up to 200,000 logic cells, 500 MHz performance, and this features to deliver twice the density, twice the performance, and half the power consumption of previous-generation FPGAs [13]. Xilinx’s advance silicon modular block is combined with 90nm process, copper metallization and 300 mm wafer technologies. Specifically, the xc4vsx55 will be targeted because of its ultra-high DSP
performance. Virtex-4 devices consume up to 94 percent lower in-rush power at start up, and up to 78 percent lower static power consumption, as compared to any competing 90nm FPGA. This significant power reduction is achieved through unique power-saving configuration circuitry and the use of 90nm triple-oxide technology. The power consumption advantages inherent to Virtex-4 FPGAs reduce system power supply and cooling loads, improving long-term system reliability and lowering total system costs. Designers can benefit from this power reduction while still achieving the highest levels of performance [13]. A photo of the Virtex-4 device is shown in Figure 3.1.

Figure 3.1 – Xilinx virtex-4 FPGA with Atmel ADC
3.2 Delphi ADC

The Delphi ADC3255 PCI Mezzanine card (PMC) is onboard Virtex-4 FPGA. This ADC converts an analog signal into digital signal for the input to the IFM receiver. The core of the design is Atmel 10-bit ADC, which is capable of operating at a clock rate of 2.56GHz or a Sampling frequency (fs) of 1/2.56 GHz. A top level schematic of the Delphi ADC3255 is shown in Figure 2.2 [14, 15].

![Figure 3.2: Delphi ADC3255 Top Level Schematic Diagram](image)

After the analog input is buffered and passed through the 10-bit ADC, the converted data is expanded by a demultiplexer. The model has been modified by taking the most significant bit (MSB) of the eight registers [R0:R7] following the demultiplexer, effectively providing
the required 8-bit input. The newly formed eight-bit wide bus is fed into the FPGA. Next, the data is passed through the 8-bit IFM Receiver design.

The clock input to the FPGA is provided from the Delphi chip, however is divided down to 1/8th the input of the board clock rate. More specifically, the clock rate entering the Atmel ADC is 2.56 GHz, which is 320 MHz after the division. This clock input is a specific requirement as governed by the timing reports of the receiver design.

3.3 Xilinx System Generator

In recent years, field-programmable gate arrays (FPGAs) have become key components in implementing high performance digital signal processing (DSP) systems, especially in the areas of digital communications. The logic fabric of today’s FPGAs consists not only of look-up tables, registers, and multiplexers, distributed and blocks memory, but also dedicated circuitry for fast adders, multipliers, and I/O processing. This makes the FPGA ideally suited for creating high-performance custom data path processors for tasks such as digital filtering, fast Fourier transforms, forward error correction, and highly accurate receivers [12]. Despite these characteristics, broader acceptance of FPGAs in the DSP community has historically been hampered by several factors. First, there is a general lack of familiarity with hardware design and especially, FPGAs. DSP engineers conversant with programming in C or assembly language are often unfamiliar with digital design using hardware description languages (HDLs) such as VHDL or Verilog. Furthermore, although VHDL provides many high level abstractions and language constructs for simulation, its synthesizable subset is far too restrictive for system design [12].
System generator is a software tool for modeling and designing FPGA-based DSP systems in Simulink. The tool presents a high level abstract view of a DSP system, yet nevertheless automatically maps the system to a faithful hardware implementation. What is most significant is that system generator provides these services without substantially compromising either the quality of the abstract view or the performance of the hardware implementation. Simulink provides a powerful high level modeling environment for DSP systems, and consequently is widely used for algorithm development and verification. System generator maintains an abstraction level very much in keeping with the traditional Simulink blocksets, but at the same time automatically translates designs into hardware implementations that are faithful, synthesizable, and efficient. The implementation is made efficient through the instantiation of intellectual property (IP) blocks that provide a range of functionality from arithmetic operations to complex DSP functions [12].

3.4 System Generator Design Flow

The first step is to come up with the required mathematical descriptions for the supporting algorithm and necessary operations for designing IFM receiver in this thesis, the first step is to understand the working of previous project. Realize the algorithm in the design environment, initially using double precision, but an efficient hardware implementation uses just enough fixed point precision to give correct results and most operations have a sufficiently small dynamic range that a fixed point representation is acceptable, and the hardware realization of fixed point is considerably cheaper. Hence trim double precision arithmetic down to fixed point, After this translate the design into efficient hardware. It can be difficult to guarantee the hardware implements the design faithfully. System generator eliminates this concern by automatically generating a faithful hardware implementation.
Figure 3.4 shows the Design flow the present thesis approach. Simulink provides a graphical environment for creating and modeling dynamical systems. System generator consists of a Simulink library called the Xilinx blockset which are used to created Simulink .MDL files, one can achieve this by invoking the system generator token within the model to automatically generate the register transfer language (RTL) VHDL code and cores to a user-specified locations. Xilinx ISE foundation tool is used to synthesis the newly developed design from system generator. After generation the file contains an .ISE file which can be launched via xilinx ISE tool.

**System Generator Flow**

![System Generator Flow Diagram](image)

Figure 3.3: Design Flow of the Xilinx System Generator
First the HDL files are synthesized into a NGC file, next the logic design file formats are converted into physical file format. According to the FPGA board that is chosen for this project (xc4vsx55-12ff1148), The ISE tool will provide mapping, place and route, static timing report and BIT generation report. Additionally, ModelSim can be used to simulate the behavioral model. Lastly, the bitstream is downloaded to the FPGA where design verification takes place using xilinx’s chipscope pro logic analyzer.
IV. DESIGN IMPLEMENTATION

4.1 Design Overview

The design methodology and techniques have been discussed in previous chapters, In this chapter, implementation of the IFM receiver using the Xilinx System Generator in MATLAB Simulink will be discussed. The main components of IFM receiver are as follows.

- Hilbert Transform
- Autocorrelation Algorithm
- Phase Mapping
- Frequency Detection and Calculation

A schematic diagram of digital IFM receiver is as shown in Figure 4.1.

![Digital IFM Receiver Diagram](image-url)

Figure 4.1: Digital IFM Design Overview

28
4.2 Elimination of DC Biasing

The Delphi ADC output data is of type floating point, but since the Xilinx System generator handles only fixed point data, it needs to be converted to fixed point notation, which is done using the Gateway In block from the Simulink Xilinx Blockset. It is possible to achieve an unsigned fixed point number that quantizes by truncation and wraps the overflow. A basic Gateway In block and its properties are shown below.

![Gateway In Schematic and Properties](image)

**Figure 4.2: Gateway In Schematic and Properties**

The ADC typically performs an analog to digital conversion to obtain digital samples of input analog signals. It operates by comparing the input voltage of the detected analog signal...
to a fixed reference voltage and quantizing the difference into a digital sample with a specified number of bits. In the Delphi ADC the output is of 10 bits; eight de-multiplexers (demuxes) are used to get a 320 MHz data for a sampling frequency of 2.56 GHz. The fixed reference voltage of an ADC can be interpreted as “zero”, or equivalently as the input signal voltage which translates to a “zero” for the digital sample. The reference voltage should always be a constant value. However, due to various factors like noise, tolerance of ADC components, the reference voltage is typically not fixed. This introduces a DC bias in the digital output. The following schematic diagram set up is used to eliminate the DC biasing from the ADC [16],

![Figure 4.3: System Generator Implementation of DC biasing setup](image)

The Figure 4.3 consist of a Gateway In block, couple of Register blocks for synchronization, Xilinx Subtraction block, a Constant block of value 507 and a Gateway Out block. Gateway Out blocks are basically the outputs from the Xilinx portion of the user’s Simulink design. This block converts the System Generator fixed point data type into Simulink Double type data. To remove this DC bias from the input, the digital input data is subtracted with a constant of 507 (this value is pre-determined by our experiments.) Once the DC bias is eliminated, the unsigned fixed point data is fed to the IFM receiver block.
4.3 Real-Time Hilbert Transform

4.3.1 Basic Architecture

A schematic of a basic Hilbert transform is shown in figure 4.4. ADC is sampled at a clock rate of 2.56 GHz. The data from ADC at the input of Hilbert Transform and it is of 8 bit wide at a clock rate of 320 MHz. Since this is an 8-bit data it needs to be down converted to 1 bit data, a FIFO (First In First Out) block is used for down conversion. The throughput rate at the FIFO output is 4 MHz. The size of FIFO is 2048(2K) deep with a length of 8 bits which can collect 16,384 data samples.

![Schematic Representation of One-Bit Hilbert Transform](image)

Figure 4.4: Schematic Representation of One-Bit Hilbert Transform

The Hilbert Transform has two output lines I and Q, where Q has a phase shift of $90^\circ$ from I. The 1-bit data is then passed through a FIR filter which is a 11-tap distributed arithmetic finite impulse response filter, with eleven coefficients of $[-6 0 -10 0 -30 0 30 0 10 0 6]$. This block gives 11-bit data every cycle. In table 4.1, the operation of the filter is explained. Let us consider that the 11-Tap FIR filter is supplied with a data bit pattern of [0, 0, 0, 0, 0, 1, 1, 1……] and let this 1-bit data pattern be repetitive; hence, for every 8 bits the data repeats. The coefficient values in any given cycle are added to obtain the resultant value, i.e. $Result = [6 + 0 + 10 + 0 + 30 + 0 - 30 + 0 - 10 + 0 - 6]$. Consider the values in table 4.1. For the first cycle, the value 0 which is the first bit data is only available to 6 and the result is 0. Now consider row 6, i.e.,1-bit data of logic 1 is available at column ‘6’ and...
others are zero, hence the Result = [6 + 0 + 0 + 0 + 0 + 0 – 0 + 0 + 0 + 0] = 6. If row 9 is considered where column “10” and “30” get logic 1, then the Result = [0 + 0 + 10 + 0 + 30 + 0 – 0 + 0 + 0 + 0] = 40.

Table 4.1.: Eleven-Tap FIR block input and output Pattern

<table>
<thead>
<tr>
<th>6</th>
<th>0</th>
<th>10</th>
<th>0</th>
<th>30</th>
<th>0</th>
<th>-30</th>
<th>0</th>
<th>-10</th>
<th>0</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-24</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-24</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
</tbody>
</table>

This 11-Tap FIR block gives 11-bit data every cycle. This 11-bit data is passed through the digitizer which converts it into logic ‘1’ if it is less than zero i.e., negative value and logic ‘0’ if it is greater than or equal to zero.

Figure 4.5 : Schematic Representation of Digitizer block.
To satisfy the constraints of the FIR filter, every 512 bits of data from the digitizer is shifted left by five which is equal to number of zeroes in the FIR coefficients. This can be achieved by using a serial to parallel block which converts 1-bit data into 512-bit data, on which a left shift of 5 is performed, i.e., first 5-bits are discarded and 5 zeroes are appended in the end. The shift block in the figure 4.4 performs the shift operation. Once this is achieved the 512-bit data is converted back to 1-bit data using parallel to serial conversion block. This result is stored in Q block of the Hilbert transform which is 90° phase shifted from the output I. The Hilbert Transform output I is in phase with the input data. But the delay line $Z^{-6}$ block and serial to parallel and parallel to serial block are added in line I to synchronize the output with Q. The main drawback in this design is that the FIFO operates at lower rate of 4 Mhz compared to the input rate which is 320 MHz. If the Hilbert Transform cannot keep up with the input data rate, then the FIFO will eventually run out of memory. To resolve this, first the FIFO is filled with 16,000 data samples and data acquisition is stopped until all data samples are processed. However, for continuous signal tracking this type of acquisition of data during the receiver processing creates a bottleneck for real-time applications.

4.3.2 Real-Time Implementation

The Real-Time Hilbert Transform architecture emphasizes more on maintaining the real-time throughput rate at 320MHz for all eight demux inputs. The real time schematic implementation is shown in Figure 4.7. The main aim in this design is to compute the results of the 11-Tap FIR Filter for all eight 1-bit inputs, and performing the Windowing operations on these results. The first 10 sampled inputs are saved, using two 8-bit registers, so a total of 16 samples are saved with most recent acquired sampled as the most significant bit (MSB) of
the 8-bit output. The third 8-bit register seems to be redundant but this introduces data synchronization and to accommodate delay obtained from place and route in overall system design.

![Schematic Representation of Real-Time Hilbert Transform](image)

**Figure 4.6: Schematic Representation of Real-Time Hilbert Transform**

The 11-Tap FIR filter and the Digitizer function in the basic Hilbert transform design is replaced with a $2^6 \times 1$ bit Look Up Table (LUT). This LUT is implemented using “Distributed RAM” from the Xilinx System Generator block set. The 11-bit result of the FIR filter depends on the summation of six non-zero coefficients and the digitized function outputs a logic ‘0’ or ‘1’ according to the sign of the FIR result. Eight Distributed RAMs are used to process the current batch of eight 1-bit input samples; the outputs of the LUT (RAMs) are concatenated together to form an 8-bit bus for Q. This RAM setup generates the most recent inputs samples as the MSB and delayed samples as LSB. The schematic representation of Real-Time Hilbert Transform is shown in Figure 4.8.
Figure 4.7: System Generator Implementation of Real-Time Hilbert Transform

Since the LUT (RAMs) setup takes one clock cycle to complete its operation, the data point of I needs to be delayed by one clock cycle to synchronize with the delay of Q, which is saved by R2 register as past samples for data processing. The windowing function block takes every 512 bit data from the output of LUTs and discards first 5-bits and then pads 5-zeroes at the end of the 512 data batch. Synchronization block is used to synchronize the windowing function block in the design. Bit basher blocks from Xilinx System Generator Block set are used to replicate the data bus. BitBasher1 and BitBasher2 are used to generate 8-bit bus line from 1-bit data lines.
4.4 Autocorrelation Algorithm

The autocorrelation algorithm is shown in fig 4.8, Here $S_n$ represents the non-delayed 8-bit input data to the algorithm and $P_n$ represents the delayed 8-bit input data with the delay lines of $Z^{-1}, Z^{-2}, Z^{-8}, Z^{-32}$ and $Z^{-128}$. The black and gray arrows indicate that the data is fed to autocorrelation block at different intervals. A reset line is used to control the data flow from the Hilbert Transform.

![Figure 4.8: Schematic diagram of Autocorrelation Algorithm](image)

Figure 4.8: Schematic diagram of Autocorrelation Algorithm
The 8-bit $S_n$ and $P_n$ data is converted into eight 1-bit individual data. This is obtained by passing the data line through Bit basher blocks, Each Bit basher is programmed to select 1-bit data. For example, “Bitbasher1” block gives LSB $S_n$ and “Bitbasher8” gives the MSB $S_n$ data. Similarly, “Bitbasher9” gives LSB of $P_n$ data and “Bitbasher16” gives MSB of $P_n$. LSBs of $S_n$ and $P_n$ from bit basher blocks are then fed through the XNOR block which performs bit operations, It’s output is then given to the ‘CMult1”’block which multiplies the XNOR output data with a value of 2; the result from this block may be integer 2 (positive value) or Zero. This resultant value is then passed through a subtraction block which gives either $-1$ or 1 as the result. This operation is performed on each individual bit of $S_n$ and $P_n$, from LSB to MSB and after one clock cycle of operation each Subtraction block (overall 8 blocks) holds either a positive or negative 1-bit data (total of 8-bit data). Finally these data
are added together using multiple stage adder setups as shown in Figure 4.10. These values are accumulated by the ‘Accumulator’ block and after every 256-bit data batch (i.e., for every 32 clock cycles) the accumulator is made to reset using the reset control logic. The multiple stage adder setup is located between Pipeline Registers (shown as black stripes in Fig 4.9 ). These Registers are used to increase the throughput and to achieve a minimum latency in Autocorrelation Algorithm. During the Synthesis and Implementation process, it was found out that without the Pipeline Registers the Xilinx ISE synthesized the design without any error, but during the generation of Post-Place and Route Static Timing analysis, there were few timing constraints which were not met with the original design leading to timing errors. To overcome these errors, the pipeline registers are placed as shown in figure 4.9 so that the desired latency is achieved. The “Register1” block contains eight registers of one bit for each data line, ‘Register2’ block contains four registers, ‘Register3’ block with two and ‘Register4’ consists of one Register block set. This pipelined set up can be applied to various blocks of the IFM receiver and no doubt there would be better throughput for the design, but more pipeline registers lead to more number of gates used on the FPGA and also latency increases. Hence there is a tradeoff between Throughput and Latency.

### 4.5 Reset Control Logic

The Autocorrelation block needs to generate 256-bit data batch, since every cycle there are eight 1-bit data available. Hence, at the 32\textsuperscript{nd} clock cycle, the accumulator collects the last 8-bit batch of the 256-bit data (i.e. 8-bits \( \times 32 = 256\)-bits). After this the accumulator should start collecting the next batch of data, hence it is made to reset. Using a 6-bit counter block
(which runs from 0 to 31), a constant block set ‘0’ and a Relation block set, a simple but effective reset logic is achieved as shown in figure 4.11.

![Figure 4.10: System Generator Implementation of Reset Control Logic](image)

The ‘Relational1’ block set gives output as logic ‘1’ when both the inputs are same. Since the constant value in the ‘Constant2’ block set is already chosen to be logic ‘0’, when the counter reads ‘0’ the result of Relational block is ‘1’ and this is used to reset the accumulator block. The output of counter and the Relational block is shown in Figure 4.12.

![Figure 4.11: Output Waveform of Counter(top) and Relation Block (bottom)](image)
In figure 4.12, the top waveform is of the counter as can be noticed, it is an up counter and the increment is from 0 to 31 and the bottom waveform is of the relational block, which gives the value of logic ‘1’ whenever the counter goes to zero.

4.6 In Phase and Quadrature Phase Autocorrelators

The Autocorrelation computation involves handling complex data. The output I (Real) and Q (Complex) data from Hilbert Transform are considered. Correlating ‘I’ data with delayed ‘Q’ data which is In Phase correlation and Quadrature Phase correlation is correlating ‘Q’ with delayed ‘I’ does not provide the adequate result. If one consider the following complex expressions

\[ Z_1 = X_1 - j \cdot Y_1 \]  
\[ Z_2 = X_2 - j \cdot Y_2 \]

Where: \( X_1 \) and \( X_2 \) represent I, \( Y_1 \) and \( Y_2 \) correspond to Q. Let \( Z_1 \) be non-delayed data and \( Z_2 \) be delayed data. Correlating \( Z_2 \) with \( Z_1 \), we get:

\[ S1 = \sum [(X_2 - j \cdot Y_2) \cdot \text{conj}(X_1 - j \cdot Y_1)] \]  
\[ (4.1) \]

Where \( \text{conj} \) is the conjugate function, further simplifying

\[ S1 = \sum [(X_2 - j \cdot Y_2) \cdot (X_1 + j \cdot Y_1)] \]  
\[ (4.2) \]

\[ S1 = \sum [X_2X_1 + jY_1X_2 - jY_2X_1 + Y_1Y_2] \]  
\[ (4.3) \]

\[ S1 = \sum [(X_2X_1 + Y_1Y_2) + j(Y_1X_2 - Y_2X_1)] \]  
\[ (4.4) \]

Rearranging, and replacing the values of \( X \) and \( Y \) with equivalent I and Q, we get

\[ S1 = \sum [(II_\Delta + QQ_\Delta) + (QI_\Delta - IQ_\Delta)] \]  
\[ (4.5) \]
Where: \( \Delta \) represents the delay.

Figure 4.12: System Generator Implementation of In phase Autocorrelators

Figure 4.13: System Generator Implementation of Quadrature Phase Autocorrelators
Hence, the result from equation 4.7 indicates that there should be two sets of autocorrelation variables one which adds \( I_\Delta \) with \( Q_\Delta \), this indicates the In Phase autocorrelation and another block which calculates difference between \( Q_\Delta \)and \( I_\Delta \), which is Quadrature Phase autocorrelation. Figure 4.13 and Figure 4.14, show Xilinx System Generator Implementation of Equation 4.7, Since the Autocorrelation block has five output lines, there will be total of five I/Q autocorrelation pairs in each block, which are added/subtracted and passed through the ‘To Register’ blocks. This “To Register” block set has two inputs, \( \text{din} \) and \( \text{en} \), the ‘din’ port accepts input data when the enable port ‘en’ is asserted to logic’1’. The ‘en’ port is controlled by reset control logic, these blocks share data with another section of the design at different latency and sample period.

### 4.7 Crossing Clock Domains

In an ideal design, a simple Register with enable port can be used to collect the data from the Adder/Subtraction blocks since the Register holds the data until the enable port is made high. The input of the Register operates at 320 MHz, in this design for every 32 clock cycles the enable port is made high, which means the output of the Register will hold a value for 32 clock cycles, and this is equivalent to the register output running at 10MHz. In the present design the output need not run faster than this since data from the register is sufficient enough to calculate the phase.

The remaining blocks from this point on operate at 10MHz. Phase Detection and Frequency calculation blocks need to be synchronized to this frequency, since the overall design is operating at two different frequencies. The ‘To Register and From Register’ block
pair is used to cross clock domain boundaries for which a single register is implemented in hardware and is clocked by the ‘To Register’ block clock domain.

Let us consider Hilbert Transform and Autocorrelation Algorithm together as Domain A and Phase Detection and Frequency calculation Blocks as Domain B. A shared register pair is used to cross the two clock domains as shown in the following figure 4.15.

![Figure 4.14: System Generator Representation of Cross Clocking Domains](image)

When the design is generated using the Multiple Subsystem Generator block, only one register is included in the design. The clock and clock enable register signals are driven from Domain A. This is shown in the figure 4.16 below.

![Figure 4.15: Hardware Implementation of Cross Clocking Domains](image)
These crossing clock domains are usually affected by metastability which can be reduced by including two register blocks immediately following the ‘From Register’ block to synchronize the data to the From Register’s clock domain.

4.8 Phase Detection

The Phase Detection block is used to determine the Phase of the autocorrelation values using the phase Look Up Tables (LUT). Consider the values from the In Phase Autocorrelation block as I1, I2, I8, I32 and I128 and for Quadrature Phase Autocorrelation block as Q1, Q2, Q8, Q32 and Q128. Figure 4.16 show the Xilinx System Generator Implementation of the Phase Detection Block.

![Diagram](image)

Figure 4.16: System Generator Implementation of Phase Detection block

Consider Equation15: $\phi = \tan^{-1}\left(\frac{Q}{I}\right)$, which is used to determine the phase of the autocorrelation values. First the values of I and Q are passed through “M code” block (all_pos Xilinx block shown in Figure 4.16) which acts as an absolute value function, based on the sings of I and Q it reduces the number of elements in the next LUT block sets. In the
Xilinx System Generator, division can only be performed on the elements which are powers of 2. Since the values of I and Q are not powers of two (I and Q are of 10-bits each), a method to divide Q by I must be established. A “M code” block, `q_over_i` is used, which holds a Look Up Table of predefined $\frac{1}{I}$ values from 1 to 512, these values are rounded off to 5 decimal places. They are then multiplied with Q to create $\left(\frac{Q}{I}\right)$ values. Once these values are determined, they are passed through another M code block set, `phase_LUT`, where they are compared with predetermined bits which calculate the output values in degrees from $0^0$ to $90^0$. The Phase “M code” Block set considers the values in degrees and based on the signs of I and Q the phase is determined for a pair of I and Q.

Figure 4.17: System Generator Implementation of Phase LUT with 5:1 Muxes
| Case 1 | I > 0 And Q <= 0 | 360 - deg |
| Case 2 | I <= 0 And Q > 0 | 180 – deg |
| Case 3 | I <= 0 And Q <= 0 | 180 + deg |
| Case 4 | Else | deg |

Table 4.2: Phase Modification base on signs of I and Q

Since there are five pairs of I and Q, calculating five phase values at once will lead to high hardware demand for LUTs. Hence, instead of using 5 separate LUT setups, a pair of multiplexers can be used to select one pair of I and Q values at a time. For the select line, a counter is used (To select the pairs I1 and Q1, I2 and Q2, and so on.) which counts form 0 to 4. The calculated phase value from Phase LUT block set is stored in Registers as shown in Figure 4.17. Once all the five phase values has been calculated and stored, they are ready for calculating the frequency values.

4.9 Frequency Calculation

In the IFM Receiver, the final stage is to map the phase into frequency. The Xilinx System Generator implementation of Equations 2.39, 2.40 and 2.41 which are used to calculate the frequency is shown in Figure 4.18.

Figure 4.18: System Generator Implementation of Frequency Calculation
All the blocks in the schematic are designed using “M code” block set. Each “M code” block set contains a user defined MATLAB code, details of this MATLAB code is shown in Appendix. First, phase1 which is \( \tan^{-1}\left(\frac{Q_1}{I_1}\right) \) determines the first frequency \( f_1 \), and then it is passed through “zone2” block which aids in the mapping of \( z_2 \). Frequency is calculated using “freq2” block set based on \( f_1, z_2 \) and Phase2. Similarly, this process continues till all the frequencies \( f_1, f_2, f_8, f_{32} \) and \( f_{128} \) are mapped. The final signal frequency is based on the frequency mapped from the longest delay line \( S_{128} \). The signal frequency is calculated using the formula:

\[
Signal \ Frequency = \left(\frac{f_{128}}{2\pi}\right) \times 2.56 \times 10^9
\]  

(4.8)

### 4.10 Threshold and Detection Variable

The dilemma of signal detection can be considered to “finding an appropriate threshold, one above which noise samples seldom rise and below which signal pulses seldom fall”[5]. Even though the frequency signal has been detected from the frequency calculation block, one must be able differentiate the signal from noise. The variable that can be detected is equivalent to sum of squares of all autocorrelation variables and it is given by

\[
D = S_1^2 + S_2^2 + S_8^2 + S_{32}^2 + S_{128}^2
\]  

(4.9)

Where: \( D \) is the detected variable

In Figure 4.19 to obtain the square of \( S_1 \), it is fed into ports of a and b of multiplier block, and the same is repeated for \( S_2, S_8, S_{32} \) and \( S_{128} \). These values are summed using the Adder block. These operations are performed in both In Phase and Quadrature Phase Autocorrelation blocks. The two detected variable values from these two blocks are passed
through a adder block which gives a 24-bit value and then fed to a “M code” block which gives the output as logic ‘1’ if the input is greater than 200,000 else gives a logic ‘0’. This Threshold value is determined through trial and error experimental results from the next section.

Figure 4.19: System Generator Implementation of Detection Variable
V. EXPERIMENTAL AND SYNTHESIS RESULTS

5.1 Xilinx system generator results

The IFM is realized using Xilinx system generator block set in Matlab’s Simulink Tool. First the key equations are implemented and tested for the input frequency ranging from 40MHz to 1.24GHz. For explanation purpose a sine wave with a carrier frequency of 320MHz is used as input signal. Figure 5.1 shows the output of Hilbert transform, the waveform of I and Q show the 8-bit values for the input frequency of 320MHz.

![Figure 5.1: System Generator Result of Hilbert Transforms I(top) and Q(bottom) data.](image)
The results of the Autocorrelation blocks are shown in Figure 5.2 and 5.3; the top most wave form is S1 of the In Phase/Quadrature Phase autocorrelation blocks followed by S2, S8, S32 and S128. In phase block holds all the real values and the Quadrature phase block holds the imaginary values.

Figure 5.2: System Generator Result of In phase Autocorrelators values for $f_c = 320$MHz
Figure 5.3 System Generator Result of Quadrature Autocorrelators values for $f_c = 320$MHz

The Phase Detection Block results are shown in Figure 5.4. From the Table 1.3, Phase values for five sets of I and Q values are plotted. Figure 5.4 is divided into five different sections. In the first section a waveform “Phase1” is plotted, The Phase value calculated via I1 and Q1 data form In phase and Quadrature Phase Autocorrelation blocks respectively.
The second section “Phase 2” is the second Phase wave calculated from I2 and Q2 respectively, and hence followed Phase 8, Phase 32 and Phase 128. The bottom three phases are wrapping around $2\pi$ switching from $360^0$ to $0^0$ which is in the acceptable range.

Figure 5.4: System Generator Result of Phase Detection Block for $f_c = 320$MHz
The Figure 5.5, shows the simulation results of overall detected signal frequency with an error of 1MHz and the value of the detection flag bit is also shown, The Detection flag bit goes high when the input is greater than 200,000.

Figure 5.5: System Generator Result of Frequency and Detection Flag.

Additional testing was carried out with a noisy signal. The receiver is tested with noisy chirp input signal shown in Figure 5.6 with a SNR equal to -10dB, ranging from 40MHz to 1.24GHz. A white Gaussian noise blocks was added to the sine wave to create a noisy input. The maximum error was 2 MHz with average error less than 1 MHz. The amplitude of the noisy signal can be set at 0.1V or greater than input signal power determined to be 0.0025 Watt as given by the following equation

\[ P = \frac{V_{RMS}^2}{R} \rightarrow P = \sqrt{\frac{(0.5)^2}{2}} = \frac{0.25}{50\,\Omega} = 0.0025\,W \]

When the amplitude is 0.5V and the resistance is set to be 50Ω.

The output of the Detected frequency from IFM receiver is shown in Figure 5.7, since the input is varied in step of 1MHz, the detected frequency looks like a linear plot. And this show that all the frequency from 40MHz to 1.2GHz is well detected.
Figure 5.6: System Generator Output of Noisy Signal with -10dB noise

Figure 5.7: System Generator detected Frequency with -10dB noise (40 – 1240 MHz)
Figure 5.8: System Generator result of Detection variable

Figure 5.8 shows the detection variable (top) as a sweep input changes from 40MHz to 1240 MHz, The bottom yellow line shows the detection variable with no signal. An appropriate threshold value for the detection variable is 200,000. This indicates that the output available above threshold value is free from noise; the detection flag value is made logic’1’ when the detection variable is greater than the threshold value.
5.2 Xilinx ISE Results

In Figure 5.9, the Design summary of the Xilinx ISE project navigator is shown. For the Delphi 3255 FPGA (xc4vsx55-12ff1148), total number of the FPGA slices consumed by this design is 21%. In the previous one bit IFM Design 18% of the slices were used, from the available 24,576 Slices, hence Only 3% more area has been used.

![IFM_12SEP2008_2 Project Status](image)

No partition information was found.

![IFM_12SEP2008_2 Partition Summary](image)

![Device Utilization Summary](image)

Figure 5.9: Xilinx Project Navigator Design Summary
Figure 5.10: Xilinx Floorplanner of IFM Implemented Design

Xilinx Floorplanner is a graphical placement tool that provides "drag and drop" control over design placement within an FPGA. Floor planning is particularly useful on structured designs and data path logic. With the Xilinx Floorplanner, designers can see where to place logic for optimal results, placing data paths exactly at the desired location on the board [17]. The device utilization layout of the complete floor plan on the Xilinx Virtex-4 FPGA can be viewed in Figure 5.10. The design hierarchy of the Real-Time IFM receiver was incorporated into the USER_APP System_IF source code of the Delphi design package shown in the legend at the top right corner.
5.3 Xilinx ChipScope pro Verification

The Bit stream generated from the Xilinx ISE simulator for the IFM receiver design is introduced to Xilinx Chip Scope Pro 8.2i logic analyzer for verification. The Workstation Setup used to confirm the functionality and the performance of the IFM receiver is shown in the Figure 5.11.

Figure 5.11: Xilinx ChipScope pro and Vertex-4 FPGA Workstation Setup

In this Lab setup, the Delphi ADC3255 and Xilinx Virtex-4 FPGA are clocked externally using an Agilent (Hewlett Packard) signal generator at an input sampling frequency of 2.56 GHz. The incoming RF input signal is provided using an Agilent RF signal generator from 40 MHz to 1.24 GHz at full scale amplitude value, The Xilinx Chip Scope verification tool is used to test the design for every 1MHz input signal data.

From the Xilinx Chip Scope verification tool, Figure 5.12, 5.13 and 5.14 show the output waveform generated for the RF signal generator input of 40 MHz, 320 MHz and 1240MHz. These tests shown are validated using the clock generator at an input sampling frequency with full scale amplitude of -9.3dBm.
Figure 5.12: Xilinx ChipScope Pro 8.2i Output for Test Frequency of 40 MHz
Figure 5.13: Xilinx ChipScope Pro 8.2i Output for Test Frequency of 320 MHz
Figure 5.14: Xilinx ChipScope Pro 8.2i Output for Test Frequency of 1240 MHz
From the Xilinx Chip Scope Pro, output waveform for three samples (triggers) for each input frequency of 40MHz to 1240MHz were recorded and overall an average error was 1.0513 MHz. A plot of 3600 samples is shown in Figure 5.15. The average error was 96% of the time having less than 1 MHz and 3.2% having an average error between 2 and 3 MHz and 0.8% having an average error between 3 and 4 MHz.

**Figure 5.15: IFM receiver average error with 3,600 samples**
VI. Conclusion and Future work

6.1 Conclusion

Real-time data processing of the digital IFM is critical for constant spectrum monitoring and RF signal recognition. Architecture improvement for Hilbert transform and Autocorrelation algorithm of the digital IFM receiver is both verified and validated for real-time processing. Using the Xilinx Virtex-4 SX55 FPGA with onboard Atmel ADC and Delphi’s design kit, Xilinx ISE Project Navigator, MATLAB, and ChipScope Pro, one is able to validate the proof-of-concept for a digital IFM receiver from test module to hardware verification. The real-time implementation shows a high input throughput rate improvement by a factor of 80 at 3200 Mbps. Continuous data acquisition also allows constant signal tracking without time gaps.

6.2 Future Work

Future work includes the ability to determine optimum bit truncation for accurate frequency detection and increase the data throughput rate of other components in the digital IFM receiver. Other work includes the ability to detect simultaneous signals, which has been a topic of discussion for some time. Also, a method of accurately determining the phases in parallel without tremendous amounts of hardware could prove as a useful addition.
REFERENCES


17. http://www.xilinx.com/products/design_tools/logic_design/design_entry/floorplanner.htm