Rapid Radio: Analysis-Based Receiver Deployment

Jorge A. Surís Pietri

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Peter Athanas, Chair
A. Lynn Abbott
Ali R. Butt
Cameron Patterson
Jeffrey H. Reed

August 7, 2009
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia
A large body of work has been produced in the area of productivity enhancers for the design of both Software-Defined Radio and Field Programmable Gate Arrays systems. These tool are created with the goal of aiding the user in the process of instantiating a design. They do not address, however, a specific use-case in which the user does not know or care about what the design of his system is. In this work, analysis-based design is presented and applied to FPGA-based SDRs. The RapidRadio framework abstracts away much of the knowledge required for analyzing an unknown signal and building an FPGA-based receiver. Resource utilization is traded-off for reduced implementation time and increased flexibility.

Automatic modulation classification is done with blind parameter estimation. Unlike other contemporary work, no \textit{a priori} knowledge about the signal being classified is assumed. This leads to the development of a system that does not depend on perfect synchronization to classify the signal. A new quasi-generic synchronization architecture that allows the synchronization of multiple modulations schemes is presented. The result of the modulation classification is used to automatically create an FPGA-based radio receiver.

This work is supported by the Harris Corporation, Government Communications Division.
Acknowledgments

Completing a doctoral dissertation is a very time consuming endeavor. One which I would not have been able to complete without the help of many people.

First I would like to thank my advisor, Peter Athanas. From the start he guided me and made sure I was always funded. He gave me the freedom to select the approach I though was best and always provided good feedback to ensure I was headed in the correct direction.

To Cameron Patterson, thank you for teaching me about FPGAs. A large part of what I learned about FPGAs I learned from you.

To my parent Juan and Ana María, and my sister Ana María who always believed in me. You instilled in me the work ethic required to complete this endeavor and the thirst for knowledge that brought me here.

To my friends Dave Raymond, Alex Marschner, Eric Lorden and Ryan Thomas who kept me sane throughout the four years of craziness.

To Adolfo Recio whose communications knowledge was invaluable to completing this project. Thank you for sharing your expertise and your friendship.

To Mike Shershin, Donna Strand and George Leier. You worked with me to get around my hectic schedule and my odd hours. Without your help I would not have had the financial support required to complete this degree.
To my brother Juan who got me interested in getting a PhD and often served as a sounding board for my ideas.

Lastly, I want to thank my wife Noreen and my two kids Jorge and Gustavo. You gave me the strength to keep going even when it seemed like I was not making any progress. You always encouraged me and believed in me. Noreen, you took on the burden of running the family so that I could concentrate on my studies. Thank You.
Contents

List of Figures x

List of Tables xv

Glossary xvi

1 Introduction 1

1.1 RapidRadio 2

1.2 Theoretical Foundations of RapidRadio 5

1.2.1 Value of Information Over Time 5

1.2.2 Value of Information Based on Completeness 7

1.3 Contributions of RapidRadio 8

2 Related Work 10

2.1 Software-Defined Radio 12

2.1.1 Cognitive Radios 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.2</td>
<td>Software Communication Architecture</td>
<td>15</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Software Design Platforms</td>
<td>16</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Automatic Modulation Classification</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Field Programmable Gate Arrays</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>FPGA Design Abstraction</td>
<td>21</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Coarse-Grained Architectures</td>
<td>21</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Model-based FPGA Design</td>
<td>22</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Hardware Description Language Generators</td>
<td>24</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Multi-Processor System-on-a-Chip</td>
<td>25</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Communication Abstraction</td>
<td>26</td>
</tr>
<tr>
<td>2.4</td>
<td>Summary</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>RapidRadio: Approach</td>
<td>28</td>
</tr>
<tr>
<td>3.1</td>
<td>Signal Analysis</td>
<td>30</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Spectrum Sampling</td>
<td>31</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Signal Identification and Isolation</td>
<td>33</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Spectral Fitting</td>
<td>33</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Synchronization</td>
<td>35</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Metrics and Constellation Evaluation</td>
<td>36</td>
</tr>
<tr>
<td>3.1.6</td>
<td>Radio Description File</td>
<td>36</td>
</tr>
</tbody>
</table>
3.2   Radio Synthesis .................................................. 37
      3.2.1  Module Instantiation .................................. 37
      3.2.2  Inter-Module Connections .............................. 38
      3.2.3  Configuration Generation ............................. 40

4   Concept Prototype  ........................................... 41
   4.1  Phase Shift Keying ......................................... 41
       4.1.1  Binary Phase Shift Keying ............................ 42
       4.1.2  M-ary Phase Shift Keying ............................ 43
       4.1.3  Raised Cosine ....................................... 45
   4.2  Analysis Phase .............................................. 45
       4.2.1  Artificial Intelligence Engine ...................... 47
       4.2.2  Signal Selection and Isolation Interface ........... 47
       4.2.3  System Initialization ................................ 49
       4.2.4  Spectrum Sampling .................................... 50
       4.2.5  Signal Isolation ...................................... 52
       4.2.6  Modulation Parameter Estimation ..................... 53
       4.2.7  Symbol Timing Recovery ............................... 56
       4.2.8  Carrier Recover ...................................... 63
   4.3  Hypothesis Fitness Evaluation ............................ 67
4.3.1 Phase Profile ................................................. 70
4.3.2 Amplitude Profile ........................................... 71
4.3.3 Symbol Transitions ........................................... 74
4.3.4 Combining the Metrics ....................................... 75
4.3.5 Parameter Refinement ........................................ 77
4.3.6 Radio Description File ..................................... 79
4.4 Radio Deployment Phase ........................................ 80
  4.4.1 Radio Synthesis ............................................... 82
  4.4.2 Platform Description File ................................... 82
  4.4.3 Module Selection and Generation ........................... 84
  4.4.4 Bit Precision .................................................. 87
  4.4.5 Inter-Module Connection Generation ....................... 87
4.5 Receiver Architecture and Implementation ...................... 88
  4.5.1 Downconverter ............................................... 88
  4.5.2 Matched Filtering .......................................... 89
  4.5.3 Resampler ................................................... 90
  4.5.4 Timing Recovery Implementation .......................... 91
  4.5.5 Derotator Implementation .................................. 94

5 Results 101
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Signal Classification</td>
<td>101</td>
</tr>
<tr>
<td>5.2</td>
<td>Synchronization Architecture</td>
<td>110</td>
</tr>
<tr>
<td>5.3</td>
<td>System Synthesis</td>
<td>114</td>
</tr>
<tr>
<td>5.4</td>
<td>Receiver Deployment</td>
<td>114</td>
</tr>
<tr>
<td>5.5</td>
<td>User Base Survey</td>
<td>116</td>
</tr>
<tr>
<td>5.5.1</td>
<td>User Background and Experience</td>
<td>120</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Framework Evaluation and Radio Deployment</td>
<td>122</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
<td>123</td>
</tr>
<tr>
<td>6.1</td>
<td>Future Work</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td><strong>Bibliography</strong></td>
<td>126</td>
</tr>
<tr>
<td>A</td>
<td>Sample XML Files</td>
<td>139</td>
</tr>
<tr>
<td>A.1</td>
<td>Radio Description File</td>
<td>140</td>
</tr>
<tr>
<td>A.2</td>
<td>Hardware Description File</td>
<td>141</td>
</tr>
<tr>
<td>A.3</td>
<td>Module Description File</td>
<td>141</td>
</tr>
<tr>
<td>B</td>
<td>User Survey</td>
<td>142</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Analysis-based design flow. ................................................. 2
1.2 Communications protocol stack. ............................................ 4
1.3 Value of information over time. .......................................... 5
1.4 Value vs. Completeness. ....................................................... 7

2.1 The relationship of software defined radio design abstraction techniques. . . 12
2.2 Basic Software-Defined Radio architecture. ............................... 13
2.3 Spectrum utilization with and without spectrum sharing .................... 14
2.4 Simplified representation of FPGA fabric ................................ 19
2.5 Digital circuit mapped to a look-up table. (a) Shows a digital circuit that determines if two point are in different quadrants. (b) Shows the look-up table that emulates the digital circuit. ............................... 20

3.1 Flow of control in a RapidRadio enabled device. .......................... 29
3.2 RapidRadio system architecture. ............................................. 30
3.3 High-level flow of the analysis phase. ...................................... 32
3.4 Ideal spectral shape for PSK/QAM and FSK modulation schemes.

3.5 Module instantiation flow.

4.1 Sample BPSK signal for alternating symbols.

4.2 Common Phase Shift Keying constellations.

4.3 Raised Cosine filter definition.

4.4 Frequency response of Raised Cosine filter with $\alpha = \{0, .50, 1\}$.

4.5 Time response of Raised Cosine filter with $\alpha = \{.25, .50, .75\}$.

4.6 RapidRadio GUI for signal selection and isolation.

4.7 Interface between the Matlab front-end and the CLIPs back-end during system initialization.

4.8 132k-point Discrete Fourier Transform (left) vs. Bartlett Periodogram with 64 windows (right).

4.9 A properly selected signal for analysis.

4.10 Ideal spectral shape of signal.

4.11 Spectral fitting of a signal.

4.12 BPSK signal with four samples per symbol.

4.13 Plots of the obtained symbols for BPSK signal. (a)n=0 (b)n=1 (c)n=2 (d)n=3

4.14 Symbol timing recovery architecture.

4.15 Spectral line generator.
4.16 Complex filters used in the spectral line generator .......................... 60
4.17 Phase response of the spectral line generator ................................. 61
4.18 Threshold crossings by the accumulator value ............................... 62
4.19 RapidRadio derotator architecture .............................................. 63
4.20 (a) Traditional 64QAM constellation (b) Power efficient 64QAM constellation 66
4.21 RapidRadio hypothesis evaluation Graphical User Interface ............... 69
4.22 Phase profile of QPSK ............................................................... 71
4.23 Amplitude profile of 8QAM ....................................................... 72
4.24 (a) Three dimensional plot of theoretical symbol distribution for QPSK (b) Contour plot of theoretical symbol distribution for QPSK (c) Three dimensional plot of empirical symbol distribution for QPSK (d) Contour plot of empirical symbol distribution for QPSK .................. 73
4.25 (a) Theoretical symbol transition distribution for 8QAM (b) Empirical symbol transition distribution for 8QAM ................................. 75
4.26 Signal classification Bayesian network ...................................... 76
4.27 Radio deployment Graphical User Interface ................................. 81
4.28 Radio synthesis flow ............................................................... 81
4.29 Cascading parameter definitions .............................................. 86
4.30 RapidRadio receiver architecture ............................................ 88
4.31 Symbol synchronizer implementation ........................................ 91
4.32 Spectral Line generator Diagram ................................. 92
4.33 Complex IIR ..................................................... 92
4.34 PLL Implementation .............................................. 93
4.35 Derotator FPGA implementation ............................... 95
4.36 Kalman filter FPGA implementation ......................... 96
4.37 Grid-based slicer architecture. ................................. 97
4.38 Distance block. .................................................. 98
4.39 Distance-based slicer architecture ............................. 100

5.1 Performance of signal classification algorithm with no victory margin .......................... 103
5.2 Comparison of metrics for hypothesis of 8QAM and 16QAM, given an actual constellation of 8QAM. .......................................................... 104
5.3 Performance of signal classification algorithm with a 5% margin of ambiguity 105
5.4 Performance of signal classification algorithm with a 10% margin of ambiguity 106
5.5 Comparison of the performance of the signal classifier for 16QAM under simulation and over the air. ................................. 108
5.6 Synchronization of over the air signals with 2% symbol frequency and 2% carrier frequency error ......................................................... 112
5.7 Synchronization of over the air signals with 4% symbol frequency and 2% carrier frequency error ......................................................... 113
5.8 Test platform for over-the-air experiments. .................... 115

xiii
5.9 Sample symbols extracted from the deployed receivers. Clustering of symbols indicates synchronization was properly achieved. 117

5.10 Constellation figures used in user survey. 121
List of Tables

4.1 Kalman filter implementation values ........................................... 65
4.2 Symbol synchronizer implementation values ................................. 94
4.3 Derotator implementation parameters .......................................... 96

5.1 Comparison between the RapidRadio classification architecture and other popular methods. ................................................................. 109
5.2 $E_bN_0$ of received signal for tested constellations. ....................... 111
5.3 Resource utilization of synchronization architecture .......................... 112
5.4 Modulation parameters obtained from RapidRadio framework for test over-the-air signals. The first row shows the nominal values used by the transmitter. The second row shows the values estimated by the spectral fitting. The third row shows the values used for receiver deployment. ......................... 118
5.5 Resource utilization for deployed receivers on a Xilinx Virtex 4 LX60 FPGA, using a sampling rate of 8MHz and an operating clock rate of 192MHz ........ 119
# Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>ALRT</td>
<td>Average Likelihood Test</td>
</tr>
<tr>
<td>ALU</td>
<td>Arithmetic and Logic Unit</td>
</tr>
<tr>
<td>AMC</td>
<td>Automatic Modulation Classification</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BER</td>
<td>Bit-Error Rate</td>
</tr>
<tr>
<td>BPSK</td>
<td>Binary Phase Shift Keying</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block RAM</td>
</tr>
<tr>
<td>CLIPS</td>
<td>C Language Integrated Production System</td>
</tr>
<tr>
<td>CMU</td>
<td>Configuration Management Unit</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CR</td>
<td>Cognitive Radio</td>
</tr>
<tr>
<td>DCM</td>
<td>Digital Clock Managers</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processors</td>
</tr>
<tr>
<td>FB</td>
<td>Feature-Based</td>
</tr>
<tr>
<td>FCC</td>
<td>Federal Communications Commission</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Arrays</td>
</tr>
<tr>
<td>GLRT</td>
<td>Generalized Likelihood Test</td>
</tr>
<tr>
<td>GPP</td>
<td>General Purpose Processors</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HAL</td>
<td>Hardware Abstraction Layer</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>HLL</td>
<td>High-Level Languages</td>
</tr>
<tr>
<td>HLRT</td>
<td>Hybrid Likelihood Test</td>
</tr>
<tr>
<td>IF</td>
<td>Intermediate Frequency</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>ISI</td>
<td>Inter-Symbol Interference</td>
</tr>
<tr>
<td>JNI</td>
<td>Java Native Interface</td>
</tr>
<tr>
<td>JTRS</td>
<td>Joint Tactical Radio System</td>
</tr>
<tr>
<td>LB</td>
<td>Likelihood-Based</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
</tr>
</tbody>
</table>
LUT  Look-Up Tables
MSB  Most Significant Bit
MPSoc  Multi-Processor System-on-a-Chip
NCO  Numerically Controlled Oscillator
NoC  Network-on-a-Chip
PDF  Probability Density Functions
PLL  Phase-Locked Loop
PSK  Phase Shift Keying
QPSK  Quadriphase Shift Keying
RDF  Radio Description File
RF  Radio Frequency
RCOS  Raised Cosine
SCA  Software Communication Architecture
SDR  Software-Defined Radio
SNR  Signal to Noise Ratio
SoC  System on a Chip
VLIW  Very Long Instruction Word
XML  eXtensible Markup Language
Chapter 1

Introduction

Although there have been many advances in abstracting away the complexity of designing hardware, it remains an arduous task. The designer must have a large knowledge-base in the area of hardware as well as in the domain area of the target application. Several domain-specific tools have been created to reduce the complexity of the design process [1, 2, 3, 4]. These tools, however, cannot aid a user who does not know or care about the low-level implementation details about the required system. With the growing interest in cognitive radios that adapt to their environment, this will become a more likely occurrence.

To illustrate this issue, consider the connectivity issue in wireless communications. An individual wishing to join an ad-hoc network may not care about the properties of the physical link being used as long as he can connect to the network. The actual implementation of the physical layer may also be of little interest. Moreover, there is no way to know what the communication architecture of the ad-hoc network is because it changes with time. Similarly, military vehicles may, in the course of their patrols, encounter an unknown radio signal. If the vehicle can determine the modulation scheme used, it could possibly identify the transmitter as a friend or a foe. In the former case, an ad-hoc connection can be setup to share tactical information effectively increasing the knowledge-base of both vehicles. In the
latter case, the vehicle could identify the source of the transmission, locate the enemy and possibly listen in on the conversation. Even if the signal is encrypted, the symbol stream can be stored for later analysis. De-modulating the signal would allow the vehicle to store a larger portion of the signal in local memory.

1.1 RapidRadio

This work investigates an analysis-based design methodology applied to the Software-Defined Radio (SDR) domain, referred to, in this dissertation, as the RapidRadio framework. Analysis-based design focuses on capturing the methodology undergone by a domain expert attempting to recover an unknown signal and deriving the system design from basic transforms applied in the process. For example, when the domain expert selects a section of the spectrum to analyze, the underlying system may interpret this as a complex sequence of signal processing operations (e.g. applying a bandpass filter and mixing the specified spectrum down to baseband). When the analysis process has concluded, a hardware system will be automatically constructed that will produce similar results to that achieved by the domain expert through the analysis process alone. Figure 1.1 shows a scenario where after applying \( n \) transforms the user transformed an incoming signal into a stream of symbols represented as alphanumeric characters. The resulting hardware system with \( m \) modules is also shown.
The goal of the RapidRadio framework is to create an analysis-based design framework for the SDR domain. Specifically, it seeks to reduce the knowledge-base required by a user for the classification of an unknown signal and the creation of a functional hardware receiver. The traditional approach to creating a radio is a slow progression of design steps involving signal modeling, receiver design, prototyping and system verification to name a few. This extensive process is done in part to obtain a near optimal and efficient receiver, where optimality may be measured in Bit-Error Rate (BER) and efficiency in the number of resources used, such as multipliers and gates, or power consumption. With the progression of technology, many of these metrics are outdated or even meaningless when instant gratification is needed. For example, minimizing the number of multipliers may be moot if the target platform contains multiple FPGAs with hundreds of otherwise unused multipliers. If the speed of deployment is paramount, and the prototyping platform has an abundance of resources, then the design process can be altered in a way to produce an operational yet possibly suboptimal receiver in a short time.

The RapidRadio framework provides such a design process guided by the principals of reduced implementation time and system versatility. The framework does not seek to replace the traditional design process of building a radio receiver, but to complement it by providing a functional prototype and a description of the architecture. If a more robust receiver is needed for the long-term, the specification provided by the framework can be handed over to the Radio Frequency (RF) and hardware engineers whereupon an optimal implementation can be produced. The prototype produced by the framework can be temporarily used until a longer-term design has been created. If on the other hand, the communications link is only temporary, then the traditional design process may be skipped altogether. Using parameterizable modules and basic communications transforms, the design abstractions common to software design platforms are brought to SDR development.

The design of a communications systems is an extensive task of which defining the physical link layer is just a small part. Several other layers transform the data between the physical
layer and the application. The progression of these transforms is often represented as a stack. Figure 1.2 shows a sample communications stack. De-modulation is done in the physical layer producing a symbol stream. The symbol stream is converted to a bit stream by mapping each symbol to a sequence of zeros and ones. The bitstream then undergoes many other transforms such as error correction, de-scrambling, de-spreading and decryption. The exact transformations applied depend on the link protocol.

RapidRadio focuses on the investigation of the design of the physical layer, providing a stream of symbols. If the communications protocol is a raw data stream and is unencrypted then a symbol stream is sufficient. If the data stream is scrambled or punctured due to encoders or error correction schemes, location awareness [5] could provide a hint as to the protocol stack. With the correct protocol stack the data could be successfully recovered. Determining the correct protocol stack though is beyond the scope of this work. If the data stream is encrypted, contemporary cryptographic analysis can applied, but this too is beyond the scope of this work.
1.2 Theoretical Foundations of RapidRadio

The presentation of timely information is inherently valuable as it adds to the sum of knowledge of a person or organization. Assigning a quantitative value to information however, is not an easy task because that value can be measured in multiple different ways. In the battlefield for example, the value of information can be measured in lives saved, equipment preserved, enemy casualties or ground gained to name a few. To complicate matters further, the value of the information changes with time and completeness. This section provides a high-level description of the value of information, how that value can be expressed and how it changes.

1.2.1 Value of Information Over Time

Although hard to quantify, it is easy to see that the value of information is a function of time ($v(t)$). Information on the target oil production set by OPEC is always valuable, but its value is obviously greater before they announce it. In some cases, however, obtaining the information too early may decrease its value. The exact relationship between value and
time is highly dependent on the specific piece of information or on the scenario. Figure 1.3 shows some likely relationships between the value of information about event \( e \) and time. Assuming that the information is correct the value of a specific piece of information will eventually diminish with time until it is almost worthless (information will always have value). This will happen at some point in time near \( t_e \), where \( t_e \) is the time of occurrence of \( e \). Obtaining information today about the quarterly profits on Eastern airlines in 1982 for example has virtually no value.

Let us consider the following message sent by enemy headquarters to soldiers in the field:

On November 24 at 6:00 hours, the left flank will attack from the south. Artillery will concentrate fire on the enemies airfield to prevent aerial cover. At 800 hours, the left flank will feign retreat to lure the enemy into extending its battle line at which moment the right flank will attach from the north. Ensure all soldiers are wearing their winter gear since the weather reports call for snow in the afternoon.

Obtaining this message on November 22, for example would allow the preparation of a suitable counter attack and the movement of troops. If the message is obtained on the 24th at 400 hours, less preparation is possible, reducing the value of the information. Obtaining the information on the 25th after hundreds of casualties and the loss of ground to the enemy has no value.

The value of information changing over time comes from its ability to affect decisions [6]. This is seen in the example above, where the closer to an engagement the information is obtained the number of decisions it can affect reduces. The impact of the decisions affected also increases with their distance from \( t_e \). Let us assume that \( N_e \) decisions can be made concerning event \( e \). The time in which the \( i \)th decision is made can then be stated as \( t_{di} \) for \( 0 < i \leq N_e \). \( v(t) \) is then maximized when the time the information is obtained \( t_o \) follows the following inequality: \( t_o \leq t_{d1} \). Notice that this implies that obtaining information about event \( e \) before any decision related to \( e \) can be made does not add to its value.
When gathering information from enemy transmissions, however, there is a delay between when the signal is received and when the information contained in the signal is actually obtained. The time it takes to analyze the signal and de-modulate can be stated as the difference between the time in which the signal is received \( t_r \) and the time the information is obtained. By reducing \( t_o - t_r \), the RapidRadio framework increases the \( v(t) \).

### 1.2.2 Value of Information Based on Completeness

The value of information is also affected by the completeness \( c \) of the acquired information. Too little information may result in a mis-understanding or mis-interpretation of the information causing it to have a negative value. There may exist, however, a point of diminishing return in which more information adds little to no value. Let us again consider the message in the above section. Notice that if only the first sentence of this message is obtained soldiers would be moved to the south to stop the enemy offensive. This would weaken the areas in which the enemy is really planning to attack. Because the information obtained was incomplete it actually has a negative value, regardless of when it is received. Obtaining the first
three sentences of the message would however divulge the entire plan making the information very valuable even though it is incomplete. Receiving the message in its entirety would provide very little extra value because the last sentence is of little strategic value. Figure 1.4 shows the relationship between value and completeness.

If a platform detects a signal of interest it has to store the entire signal. This can result in storing a large amount of data that contains little information, which is undesirable for embedded system with limited resources. A system using RapidRadio would be able to store a larger amount of information because data can be stored as symbols. This is possible because RapidRadio performs rapid prototyping of the receiver. Having the ability to store more information in the same allotted amount of space increase the completeness of the message received therefore increasing it’s value.

1.3 Contributions of RapidRadio

The RapidRadio framework builds on top of existing technologies to extend their functionality. The foreseeable contributions of this work are:

- **Combining blind modulation classification with receiver synthesis**: A great amount of research has been done in modulation classification, but assumptions such as perfect synchronization are made. The RapidRadio framework makes very few assumptions about the input signal. Additionally a radio synthesis tool is created that creates a platform specific implementation from a platform independent description. No assumption about vendor tools or platforms are made allowing portability to other platforms.

- **Analysis-based design**: The analysis process of a domain expert is captured and a system design is automatically generated.
• **Plug-in constellation**: The RapidRadio framework provides a methodology for the specification of a constellation and the synchronization algorithms are automatically generated.

Some basic background on software defined radios and Field Programmable Gate Arrays is presented in Chapter 2. Current research efforts in design abstraction are also discussed.
Chapter 2

Related Work

The transition from static radios that only implement one waveform to one that is software defined, capable of implementing multiple waveforms has necessitated a change in the overall structure of radio platforms as well as the hardware blocks used to implement them. As the performance of General Purpose Processors (GPP) and Digital Signal Processors (DSP) have improved, functionality is being migrated from static hardware to more dynamic mediums such as software and Field Programmable Gate Arrays (FPGA). These mediums can change their behavior over time providing increase flexibility to the system.

The new flexibility exhibited by SDRs has resulted in greater functionality being implemented on individual radios. Automatic Modulation Classification, for example, is an analysis technique that is being integrated as part of some modern SDRs [7, 8, 9], allowing the radio to communicate using multiple link protocols. The increase in functionality has in turn lead to an increase in the complexity of implemented algorithms as well as an in the computational requirements of the platform. To cope with the increased computational requirements, contemporary SDR platforms are including FPGAs and DSPs. These devices offer greater throughput and computational power than a GPP, but increase design complexity. This is especially true for FPGAs, which offer a level of abstraction much lower than
that of software.

The SDR designer is typically more interested in the design of the system than on the actual implementation details. Ideally, the designer should not have to know the target implementation platform of the SDR. To this end, contemporary research [10, 11] has focused on raising the abstraction level of SDR design. This allows the designer to focus on functionality not on implementation details such as hardware-software partitioning and FPGA design. This chapter focuses on some of the key technologies used to raise the abstraction level of SDR design.

For waveform development and management, the largest efforts have concentrated on the SCA, which attempts to separate the application from the platform. The SCA, however, fails to properly incorporate FPGAs. Some research has been done on FPGA management in SDRs but a clear strategy has not yet emerged. For FPGA development, multiple strategies have emerged to increase designer productivity. Some have focused on mid-level design, trying to decrease module development time. Others, have focused on the larger system design [12, 13, 14, 15, 16]. Although most of these strategies have met with some level of success, the abstraction level for FPGA design still remains low. RapidRadio combines abstraction strategies from both areas to create a domain-specific design tool with a high level of abstraction.

Figure 2.1 shows some of the design abstraction techniques used for FPGA-based SDR development. Situational awareness, user input and automatic modulation classification can all contribute to the design of the SDR. Once a design has been settled upon, a large number of tools are available to instantiate the design. To avoid hand-coding the many eXtensible Markup Language (XML) files used by SCA frameworks, SCA generators are commonly used. For FPGA design, tools can be generally broken up into two categories. The first group aids in the development of modules, such as C-to-gates and block libraries. The second group provide a framework to combine multiple blocks into one system. As mentioned above, most abstraction techniques fail to provide the level of abstraction desired by SDR designers. In
this chapter some of the techniques shown in Figure 2.1 are discussed. This discussion serves to understand the void filled by the RapidRadio framework.

### 2.1 Software-Defined Radio

As new RF protocols and standards arise, there is a growing need for a flexible communications platform that can implement more than one standard. This desire for flexibility leads to the development of the Software-Defined Radio [17], that strives to implement as much of the radio as possible on configurable resources such as software and FPGAs. The use of configurable resources allows the waveform implemented on the platform to be modified as needed. Although there is no consensus over the definition of SDR [18], most definitions center around the radio’s capability to change its functionality over time.

Figure 2.2 shows the most basic SDR. The signal received from the antenna is first filtered and amplified in analog circuitry. Analog filtering is typically required to reduce aliasing.
Down-conversion to baseband or some other Intermediate Frequency (IF) is done. Lastly the signal is digitized and passed to a GPP, FPGA or other device for processing. All of the baseband processing is done in a configurable device to ensure maximum flexibility. By modifying the frequency of the local oscillator and the bandpass filter, different areas of the spectrum can be captured. As can be seen in Figure 2.2 the transmit chain is very similar to the receive chain.

### 2.1.1 Cognitive Radios

A Cognitive Radio (CR) is a broad term to describe an SDR that adapts to its environment and learns from past experiences [19, 20]. Haykin defines a CR as follows [21]:

Cognitive radio is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and adapt its internal states to statistical variations in the incoming RF stimuli by making corresponding changes in certain operating parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in
Figure 2.3: Spectrum utilization: (a) shows the spectrum with just the primary user. (b) shows the spectrum with both primary and secondary users.

- highly reliable communications whenever and wherever needed;
- efficient utilization of the radio spectrum.

Although most of the electromagnetic spectrum in the US has been allocated for a specific use by the Federal Communications Commission (FCC), certain frequency bands suffer from underutilization. In 2003 the FCC recognized this and proposed a rule change to allow the use of CRs for more efficient spectrum utilization [22]. The proposed rule change allows the sharing of spectrum among several users. The primary user of the spectrum always has priority, all other secondary users can access the spectrum when the primary user is not using it. Figure 2.3(a) shows the spectrum usage for a primary user with reduced activity, such as a hand-held two-way radio. Notice that the spectrum is unused approximately half of the time. Figure 2.3(b) demonstrates how spectrum utilization could be maximized by allowing secondary users to access the spectrum when the primary user is inactive.
2.1.2 Software Communication Architecture

Interdisciplinary communication in the US military is often complicated by the fact that each branch uses different radio protocols and platforms. For example, a tank in the field cannot directly communicate with the airplanes providing it with air cover. The tank crew must communicate with their commander who relays the message to the airplanes pilot through the pilot’s commanders. This can result in critical delays and errors in the communication. In an attempt to alleviate this situation, the Joint Tactical Radio System (JTRS) program was created in the late 1990s [23, 24, 25]. The JTRS program defined a goal of moving all military communications platforms to software programmable systems that can support a wide variety of protocols.

To ensure that the SDRs developed are capable of working on all radio platforms, independent of manufacturer, the Software Communication Architecture (SCA) was developed [26]. The SCA is an object-oriented programming framework, that uses a Common Object Request Broker Architecture (CORBA) [27] infrastructure for communication between all modules. Messages sent through CORBA are routed through a common server regardless of the location of the module. This means that an SCA compliant application will work the same in a uni-processor system, in which all modules are executing on the same processor, as in a multi-processor system, in which modules may be executing on different processors. A specification of the hardware to be used, called devices, and the software modules to be used, called components, for a given SDR implementation is called a waveform. The waveform specifies the aforementioned information in an XML file. An XML file is also kept for every device and component. In the case of the device, it details the characteristics of the device, such as the amount of memory, the operating system if any, etc. In the case of the components it lists all the available implementations (one implementation per supported operating systems per platform), the modules communication interfaces and any other operating requirements, such as minimum amount of memory required.
2.1.3 Software Design Platforms

The SCA presents a flexible and robust platform for the deployment of SDRs but the development of SCA compliant waveforms is not a simple task. In-depth knowledge of both CORBA and the SCA, which the radio designer may not have, are required. To address this issues several software platforms have been created, that attempt to distance the radio designer from these details. PrismTech’s Spectra framework [28], for example, approaches the design problem with what they call model-driven engineering. Using an Eclipse-based application, it allows the designer to graphically specify a component or device, its ports and how it connects to others. The tool creates C++ classes with all the CORBA Application Programming Interface (API) calls, allowing the designer to focus on the behavior of the module without having to understand all the intricacies of CORBA. The Spectra framework also creates SCA compliant XML for the waveform. Purely a design tool, Spectra does not aid in the creation or management of bitstreams for FPGA-based applications.

Many hardware platform have been developed for the design, test and implementation of SDRs. Most of these contain at least one FPGA, allowing the platform to be customized for a given SDR implementation. One or more independent modules could be placed on the FPGA at any given time. The modules on the FPGA also change based on the needs of the application. In order to manage which modules are implemented on the FPGA and which are implemented in software, SDR-specific software platforms have been developed. In [29] configuration management is done at a low level, where a Configuration Management Unit (CMU) is assigned to each processing block. Each CMU implements the processing block in the FPGA as needed, facilitating context switching and time multiplexing of the FPGA. In [30] this platform is used to manage a CR.

Other software platforms focus on creating a Hardware Abstraction Layer (HAL) that creates an interface used to communicate with the underlying architecture. In [31] middleware is used by programs to communicate with the hardware. Using a HAL allows the commu-
communications protocol between a software module and the hardware to be static regardless of the architecture of the hardware. This leads to an SDR design that can be implemented on multiple hardware platforms with little or no change to the software.

### 2.1.4 Automatic Modulation Classification

Automatic Modulation Classification (AMC) is the process by which the modulation used to transmit a digital signal is identified. This is of increasing importance to military and civilian applications. With the increase in modulation schemes and secondary users in licensed bands [22], the probability of encountering a signal with an unknown modulation scheme also increases. For the primary user identifying the modulation scheme of a signal that is not adhering to its secondary user status (i.e. does not cede use of spectrum to primary user), might help it identify the secondary user. Knowing the modulation scheme of the secondary user also permits the primary user to communicate with the secondary user to notify it of its delinquent status. A secondary user might desire to communicate with other secondary users; this is not possible however if the modulation scheme is not known. Using AMC the secondary user can communicate with any other user in the spectrum. This is especially helpful for secondary users that change their location regularly, such as a military vehicle or a transport ship.

A overview of AMC techniques is provided in [32]. AMC algorithms can be classified into two general classes, Likelihood-Based (LB) and Feature-Based (FB). In LB algorithms the likelihood function is tested for the incoming signal across several constellation types. The constellation with the highest likelihood is then considered to be the modulation scheme. LB algorithms are optimal in the Bayesian sense, because they minimize the probability of false classifications [32]. These algorithms however tend to be computationally complex. FB algorithms, as the name implies, attempts to extract features of the received signal and a decision is made based on the extracted features. Reduced complexity makes FB approaches
desirable, even though they present sub-optimal performance.

The three most common LB algorithms are Average Likelihood Test (ALRT), Generalized Likelihood Test (GLRT) and Hybrid Likelihood Test (HLRT) which are discussed in detail in [32] and [33]. Variants of these algorithms can be found in [34, 35, 36] to name a few. These approaches assume carrier recovery and symbol timing recovery have already been accomplished. The problem lies in the fact that to accomplish symbol timing and carrier recovery, the modulation type must be known because most techniques are modulation specific.

Cyclo-stationary analysis is a commonly used FB technique [37, 38, 39, 40]. This type of analysis exploits the periodicity inherent to most digital modulation schemes. The shape of the spectral coherence and the spectral correlation functions are used to determine the modulation scheme [41, 42]. In [43] and [44] the expected distribution of symbols for a given modulation scheme is compared to measured distribution of symbols using the Hellinger distance. These algorithms also assume *a priori* knowledge of the signal and perfect synchronization. A more complete approach to AMC is taken in [45, 9, 46], but some assumptions are still made.

### 2.2 Field Programmable Gate Arrays

An FPGA is an array of uncommitted logic elements that can be connected in an arbitrary manner [47]. Logic elements are connected via inter-connection resources. These resources are wire segments of varying lengths. Programmable switches are used to establish connections between two wires or between a wire and a logic element. Digital logic is implemented by dividing the circuit into smaller logic blocks. Each block is emulated by a logic element and the programmable interconnect is used to connect the elements together. A simplified representation of an FPGA is shown in Figure —reffig:genFPGA. Typically logic elements
Figure 2.4: Simplified representation of FPGA fabric
contain flip-flops which allows the implementation of clocked logic by preserving the state of a logic operation. Digital logic is most commonly implemented in a Look-Up Tables (LUT). Most digital circuit with \( N \) inputs and 1 output can be represented with an \( N \)-input LUT. Figure 2.5 shows a digital circuit with four inputs and the corresponding LUT which emulates it. Chaining together multiple LUTs allows the construction of circuits with an arbitrary number of inputs and outputs. Because most FPGAs use static RAM to store the values of the LUTs, there is no limit to how many times it can be reconfigured. This has lead to the use of FPGAs as prototyping platforms.

Although most digital logic can be implemented with LUTs and registers alone, the performance of the resulting circuit may be unacceptable. With this in mind, FPGA manufacturers started introducing devices with dedicated hardware blocks to perform common tasks. Contemporary FPGAs for example, contain Block RAM (BRAM), dedicated multipliers, Digital Clock Managers (DCM) and built in carry logic to facilitate addition amongst other functions [48, 49, 50].

Figure 2.5: Digital circuit mapped to a look-up table. (a) Shows a digital circuit that determines if two point are in different quadrants. (b) Shows the look-up table that emulates the digital circuit.
2.3 FPGA Design Abstraction

In the software design space several levels of abstraction are available to the user. Each of these levels provides certain advantages at a cost. For example, if an optimal implementation of an algorithm is desired, assembly can be used. Using assembly however can result in longer development times and more complex code. If a near-optimal solution will suffice the user might opt to use a high-level language such as C that provides a simpler development environment and a set of pre-defined functions. These languages also insulate the user from hardware-specific details. Even higher-level languages such as C++ and Java hide some of the intricacies of C at the cost of an increased footprint in terms of memory usage, execution time and system requirements. The concept of layers of abstraction hiding the implementation of lower level functions is largely missing from the hardware design paradigm. Although several component libraries exist [13, 1, 51], the user still must be aware of bit-widths, ports, timing, clock frequency, pins and communication protocols to name a few. The following sections details some of the areas in which research is being done to increase the level of abstraction in FPGA design.

2.3.1 Coarse-Grained Architectures

Coarse-grained architectures have often emerged as a way to raise the abstraction layer. Instead of using low-level logic elements such as LUTs, high-level logic elements are used [52]. Assigning a small program to each element creates the equivalent of hardware threads running in parallel. This paradigm attempts to provides the parallelism associated with FPGAs with the ease of development associated with software. The SuperCISC architecture, for example, uses an Arithmetic and Logic Unit (ALU) as its processing unit to create a reconfigurable Very Long Instruction Word (VLIW) processor [53]. The Ambric architecture on the other hand has a heterogeneous fabric that contains both processing units and memory units [54]. Each processing unit has four 32-bit RISC processors. Similarly the Element architecture
has seven different unit types [55].

When designing a heterogeneous coarse grained architecture it is not clear what is the most effective mix of hardware blocks. In [56], a methodology for evaluating hybrid architectures is presented. A framework allows the user to determine the optimal mix of fine and coarse grained elements for a given application set. Including floating point units in an device, for example, may provide a large increase in performance for graphics processing, but does little to improve the performance of fixed point DSP applications. Evaluating the effects of dedicated logic on application performance allows the development of domain-specific reconfigurable architectures.

Design complexity for coarse grained architectures however is still high. Dividing a large sequential software program into smaller parallel programs is not always feasible. The sequential nature of the processing elements may also result in reduced performance when compared to an FPGA implementation.

### 2.3.2 Model-based FPGA Design

An important step in raising the abstraction level of FPGA design is to migrate from designs consisting mostly of specialized code, to using standardized blocks. Model-based design does this by creating a development framework of pre-existing blocks that have been verified to be functionally correct. The framework’s blocks are then combined to create a design. Using pre-existing blocks for a design results in large productivity gains because it allows the designer to focus on implementing the design and not the basic building blocks. Because pre-existing blocks have been verified to be functionally correct, system verification time is reduced. Consider for example, the the design of a digital-downconverter which requires three blocks: a multiplier, a numerically controlled oscillator and a lowpass filter. Using pre-existing blocks the designer need only worry about ensuring that no overflow or underflow occurs and the design could be completed in less than a day. Designing the system from
scratch would take considerably longer. The design and verification of the oscillator alone would take more than one day. A 10x improvement in productivity is shown in [10] over regular design methods for the design of an FPGA-based communications systems.

Several model-based frameworks have been created for FPGA design. Some have found success by building on-top of pre-existing design frameworks such as Simulink and LabView [57, 13, 2, 1]. Library of FPGA blocks are added to the base framework. Users can then design a system using the regular framework’s blocks that they are familiar with. Once a system has been simulated and verified the framework’s blocks can be slowly replaced with the FPGA blocks. This allows the designer to carefully migrate the system to a full FPGA implementation. Both software only and hardware assisted simulations allow for verification of the migrated design. The final design is then converted to either Hardware Description Language (HDL) or a bitstream.

Other frameworks are specialized for the design of GPP-centric systems [51]. These tools aid the designer in the process of connecting co-processors to a main GPP. Bus architectures are pre-defined and memory maps are automatically calculated. Some pre-defined blocks are available to the designer. These systems however are more oriented toward integration rather than design. Pre-defined blocks are mostly used to abstract away bus protocols and connection interfaces such as RS-232 and Ethernet. User logic is expected to be defined outside the system.

In [58], the concept of model-based design is extended to allow the exploration of various implementations of the same algorithm. Functionality is encapsulated in high-level parameterizable blocks are later simulated. By modifying a block’s parameters the designer can understand the performance characteristics of the block.

Although the use of model-based design frameworks is increasing, most solutions are still vendor specific making it hard to migrate designs from platform to platform. Because no standard for defining high-level blocks exists, most framework contain only a few domain
specific high-level blocks. This means that even if a platform can target various devices, such as Simulink, the design must undergo large changes when the target changes. Complicated algorithms still need to be designed in terms of relatively low-level blocks such as multipliers and multiplexers. Another requirement is that all blocks have uniform interfaces.

2.3.3 Hardware Description Language Generators

Several research efforts have concentrated on using a High-Level Languages (HLL) to define the system implemented on an FPGA. These efforts are often named C-to-gates because C is the design specified in a C-like syntax. Because most domain scientists are familiar with HLLs, providing an HLL interface to FPGA design would increase the FPGA user base. There are many C-to-gates frameworks with differing capabilities, but most fall under one of two categories: module generators or function accelerators. In the former case the goal is to write a standalone FPGA module in an HLL [59, 60, 61, 62, 63, 64, 65, 66]. These attempt to provide a more common interface to FPGAs and are often used as a first stage when porting applications from GPPs to FPGAs. In the latter case the goal is to accelerate a portion of a program by using the FPGA fabric as a co-processor [67, 68, 16].

Initially many of these compilers, such as Handel-C [69, 70] and DWARV [67], required the designer to add compiler specific annotations to aid the conversion process. Areas of the C code to be implemented in hardware needed to be encapsulated in a pragma block. As the technology progressed annotations stopped being mandatory, but are still used to provide hints to the compilers. This is the case with CHiMPS [60].

Most software programs have segments that are inherently sequential and do not benefit from hardware implementation. To address this C-to-gates compiler provide hardware instantiation at different levels. The Hybridthreads framework for example uses high-level partitioning of the program into threads[68]. The designer then decides what threads should be implemented in hardware. The Comrade compiler on the other hand allows a finer grained
Functions that are rarely called and are not easily implemented in hardware can be left in software even when the surrounding code is all in hardware. Direct memory access is provided to the hardware modules allowing the use of pointers and faster memory accesses. Hardware modules are also provided with the capacity to invoke software functions.

The quality and flexibility of C-to-gates compilers has consistently improved over the years. The problem with using current HLLs to create hardware is that the HLLs used today, such as C and Java are inherently sequential. Representing a parallel system in a language that is inherently sequential is neither easy nor convenient. The HLLs can be modified to be more parallel, but then the main advantage of using a widely known and understood platform disappears. C-to-gates compilers are also used to migrate applications from GPPs to FPGAs. This too presents a problem, because although the compiler may be good at finding parallelism in loops (loop unrolling) it is hard to automate the process of parallelizing an entire application.

2.3.4 Multi-Processor System-on-a-Chip

Development of Multi-Processor System-on-a-Chip (MPSoC) on FPGAs is a natural extension of the soft-GPPs/co-processor paradigm. In [72] the MPSoC concept is used as a design abstraction mechanism. Instead of trying to implement software functionality in hardware, the software is divided into smaller processes and executed in multiple GPPs. Heterogeneous systems tailored to the applications can be obtained by incorporating customizable function units in the GPPs.

Definition and implementation of MPSoCs on FPGAs is not a simple task. Many System on a Chip (SoC) design frameworks support the definition of MPSoC but are designed with the single GPP system in mind. The ESPAM framework created by Nikolovo aids the development of MPSoC by generating the vendor-specific project files required to build the
system and to initialize the applications running on the GPP [73]. The application is specified in the form of a Kahn Process Network, and a high-level system topology are inputs to ESPAM. Communications links with blocking reads and writes are automatically generated. The MAMPS framework created by Kumar eliminates the need for specifying system topology[15]. Applications are provided to the framework in the form of a synchronous dataflow graph and the system determines how many GPPs are required and how they should interconnect.

2.3.5 Communication Abstraction

As design abstraction increases, FPGA designers will find themselves using Intellectual Property (IP) blocks in their designs. Because IP blocks may come from a variety of sources, they have different interfaces. Interconnecting these interfaces is a tedious and error-prone task. Some research has focused on standardizing the interface of blocks to facilitate interconnections [51, 12, 11]. In [74] a layered approach is used to cross standards. Another approach is automatically create wrappers for the modules such as that presented in [75] and [76].

To circumvent the module interconnection problem, researchers have proposed the use of Network-on-a-Chip (NoC) as the communications backbone of the system. In [77], a SDR transmitter that uses an NoC to communicate between modules is presented. The configuration infrastructure inherent to FPGAs is used to implement a NoC in [78]. The latter approach has the benefit of requiring few resources and providing a large bandwidth.

2.4 Summary

Even though a large body of research has concentrated on improving the productivity of SDR and FPGA designers, a unified approach that is applicable to both fields has not
emerged. Despite many efforts, FPGAs have yet to be integrated into the SCA, which is a key technology for SDRs. FPGA design tools on the other hand have not been able to raise the abstraction layer enough. Designers still need to be aware of things such as timing, ports and delays to name a few.

In Chapter 3 the RapidRadio frameworks will be discussed. Building on some pre-existing techniques, a radio signal analysis flow is presented. Receiver synthesis is done using a combination of model-based design and module synthesis.
Chapter 3

RapidRadio: Approach

As mentioned in Chapter 1, RapidRadio is an analysis-based framework for the design of software-defined radios. The goal of the framework is to reduce the knowledge base required to implement a radio receiver on an configurable platform. The flow of data through a RapidRadio enabled platform is shown in Figure 3.1. There are two flows: signal detection and normal operations. During normal operations, data obtained from the Analog to Digital Converter (ADC) is passed to the reconfigurable resources for processing of the physical layer. The resulting symbol stream is routed to the GPP/application. When attempting to acquire a new signal, the flow of data is altered to route the digitized signal directly to the RapidRadio framework operating on the GPP. The framework guides the user through the signal classification process and produces a new configuration. The radio is then programmed with the new configuration and the platform can switch back to normal operations. Due to the lack of flexibility in analog front-ends, RapidRadio can only modify the processing of the digitized signal.

RapidRadio divides the process of radio creation into two phases; the analysis phase and radio synthesis phase. The analysis phase guides the user through the process of classifying an unknown signal and determining its modulation scheme and parameters, cast in the
domain of a signal intelligence analyst. Various transforms are provided to the user and the results of these transforms are presented. Using high-level transforms shield the user from the underlying operations. The user, for example, may need to determine if a signal is synchronized or not, but does not need to know how the synchronization is being performed. In addition to presenting the results of the transforms to the user, an expert system analyzes the results and suggest a possible course of action. The output of the analysis phase is an abstract representation of the architecture of the receiver called the Radio Description File (RDF). The receiver synthesis phase transforms the platform independent RDF into a platform specific description of the receiver. Figure 3.2 shows the high-level architecture of the RapidRadio framework and how these two phases interact.

Signal detection and classification is an inherently difficult problem. Once a signal has been identified, classification is attempted. Most classification systems depend on some artificial intelligence structures [41, 42] to make decisions. The problem with this is that the artificial intelligence structures need to be trained a priori. This makes it hard to expand the system to classify new signal types. The RapidRadio framework uses the “Human in the loop” approach to address this problem. “Human in the loop” implies that the user is prompted to validate the decisions made by the framework. All the data used to make a decision is presented to the user. If the user disagrees with the decision made by the framework,
then the framework’s decision can be overridden. The interaction between the user and the framework can be seen in Figures 3.1 and 3.2.

3.1 Signal Analysis

Signal analysis is the processes through which the modulation scheme used for a signal is identified. A large body of work has concentrated on AMC (see Section 2.1.4), but many assumptions are made about the signal that simplify the classification process. In many cases the system has \textit{a priori} knowledge about the signal being classified such that making assumptions such as perfect synchronization (synchronization is discussed in Section 4.2.7) is not unreasonable. When working with the emergency bands, for example, it may be acceptable to assume the receiver has \textit{a priori} knowledge of the signals because the number of users is limited to the police, fire-fighters and rescue personnel.
The RapidRadio framework is designed to perform blind signal classification in which no assumptions about the signal are made. As mentioned above, many contemporary signal classification systems assume perfect synchronization which is difficult to obtain if the modulation scheme is unknown. RapidRadio follows a different path to signal classification that allows it to avoid such assumptions. Figure 3.3 shows a high-level description of the signal analysis process. First, the spectrum is sampled and displayed. The user then selects the area of interest in the spectrum. Based on the spectral shape exhibited in the desired area, a possible modulation family is chosen and a spectral fitting is done. Using the values obtained via the spectral fitting, the signal is converted to baseband. For each hypothesized constellation in the modulation family, synchronization is attempted. Lastly, a set of metrics are obtained to determine the correctness of each hypothesis. The user can evaluate the metrics and determine which hypothesis is correct. These steps are discussed in more detail in subsequent sections.

### 3.1.1 Spectrum Sampling

The first step in signal analysis is to obtain a sample of spectrum. Spectrum sampling is the process by which the power received at a given frequency is determined. To obtain a sample of the spectrum, a sample of the signal is converted to the frequency domain via the Discrete Fourier Transform (DFT). The width of the RF spectrum to examined is limited by the bandpass filter in the RF front-end (see Figure 2.2). If a wider sample of the spectrum is desired, a configurable RF front-end such as the one shown in Figure 2.2 is needed. Emerging technologies such as microelectromechanical systems allow the creation of the programmable RF components required for the system shown in Figure 2.2 [79]. The area of the spectrum observed can be controlled by modifying the characteristics of the bandpass filter and local oscillator in the RF front-end. This type of approach can allow the close examination of a larger area of spectrum.
Figure 3.3: High-level flow of the analysis phase.
Depending on the goals of the system and on the current geographical location, the areas examined can change. For example, if the intent is to find and join ad-hoc networks of unlicensed users, the digital television spectrum that is largely unused might be examined. Using location aware systems such as that suggested in [5] the system can estimate which channels are not in use and concentrate its search on those frequencies. If the user, however, needs to communicate with emergency services, the frequencies searched would differ.

### 3.1.2 Signal Identification and Isolation

Once the portion of the spectrum has been selected the specific signal within the spectrum must be identified and isolated from other signals. Depending upon the assigned use of the spectrum and the target signal which the user desires to recover, varying techniques can be used to identify the signal. In [80] for example, control information about a cellular network is used to determine which “slots” are open to use by an unlicensed user. This information can be used to find secondary users if the spectrum being examined is in a cellular band. The method presented in [81] of searching for receivers of the licensed band can be used in a similar fashion for television bands. If no receivers are receiving channel three for example, then it can be used without fear of interfering. Ultimately, the user will have to provide input as to what is being searched for to identify the signal of interest. The same techniques used to identify the signal of interest can then be used for an initial bandpass filtering. If the sampled signal is using the cellular band as in [80] for example, then a filter with the same bandwidth as the alloted cellular channel can be used. Other energy detection techniques [82, 83] can also be used to determine the signal bandwidth.

### 3.1.3 Spectral Fitting

In the literature, many feature extractors are presented to obtain the modulation parameters of a signal (see Section 2.1.4). These techniques however require \textit{a priori} knowledge
Figure 3.4: Ideal spectral shape for PSK/QAM and FSK modulation schemes.

about the signal. For this reason, the RapidRadio framework uses spectral fitting for feature extraction[84]. Spectral fitting tries to match the ideal shape of a signal in the frequency domain to the actual spectrum. The best fit will provide the estimated modulation parameters.

Each digital modulation family has several defining parameters with an infinite number of permutations. The spectral shape of signals using a given modulation family however can be characterized as a function of these parameters [84]. Figure 3.4 shows the ideal spectral shape for two common modulation schemes. The set of modulation parameters (β) for modulation family \( m \) can then be defined as:

\[
\beta_m = \{ p_1, p_2, \ldots, p_n \}
\]  

(3.1)

where \( n \) is the number of parameters (\( p_i \)) for modulation family \( m \). The ideal spectral shape of the signal is then stated as \( y_m(f, \beta_m) \). The actual spectrum of the signal is \( s(f) \). Optimization techniques, such as least squares can then be used to find the \( \beta_m \) that minimizes the difference between the two functions for all modulation families. This is expressed as [84]:

\[
\min_{\beta_m} \| y_m(f, \beta_m) - s(f) \|^2 \quad \forall m \in M
\]  

(3.2)

where \( M \) is the set of all supported modulation families. The \( m \) that provides the minimum error can then be considered to be the target modulation family. Artificial intelligence constructs, such as neural networks can be used to reduce the size of \( M \) to only a few possible modulation families [41, 42].

The estimated \( \beta_m (\hat{\beta}_m) \) obtained from the fitting is used to further process the signal. A more
accurate bandpass filter is developed to reduce side-channel interference with the bandwidth and center frequencies indicated in $\hat{\beta}_m$. Down-conversion to baseband and matched filtering are also performed. The resulting signal is then ready for synchronization.

3.1.4 Synchronization

Synchronization in the context of a receiver is the combination of two processes: carrier recovery and symbol synchronization. The former corrects any errors in $\hat{f}_c$ and allows the signal to be correctly mixed down to baseband. The latter attempts to find the optimal instant in which to sample the signal to recover data. Synchronization is a vital part of any digital modulation scheme as it allows for correct recovery of the data. Most algorithms used for synchronization are not only modulation-specific, but also constellation-specific. A carrier synchronizer used for B-PSK for example, is different from the one used for Q-PSK. Additionally many synchronizers have specific operating frequency ranges for which they work.

Although it is feasible to have synchronizers for every constellation in every modulation family across several candidate frequency ranges, it is extremely impractical. To maximize the utility of the RapidRadio framework, quasi-generic synchronizers circuits that work across large frequency ranges and for multiple constellations are needed. Parameterizable synchronizers which can be easily modified to work for a given constellations allows the framework to have only one or two blocks per modulation family. Quasi-generic synchronizers also allow for the testing of new constellations.

Because RapidRadio works with a relatively small sample of the signal, the synchronizers used must converge in a short period of time. Longer convergence times reduce the effective number of symbols present in the captured symbols. If the number of symbols is too small, then measurements based on statistical distributions may no longer apply. This can lead to an erroneous classification.
3.1.5 Metrics and Constellation Evaluation

After spectral fitting the modulation family has been identified but not the specific constellation. A constellation is a description of where the symbols transmitted should be found. Each symbol is described as a point in a 2-D plane where the x axis is the in-phase power (I) and the y axis is the quadrature power (Q). Figure 4.2 shows three sample constellations for the Phase Shift Keying modulation family. Constellation identification must be done after synchronization, but synchronization is constellation dependent. However, once a signal has been synchronized, it can be evaluated for correctness. An exhaustive search can be performed because the number of constellations commonly used is relatively small (i.e. less than 20). Given that $C_m$ is the set of constellations for modulation family $m$, then $x_c(t)$ is the signal, $x(t)$, synchronized with constellation $c$ where $c \in C_m$. $x_c(t)$ is then a set of symbols and constellation specific metrics are obtained. A rule-based system can then determine the most likely candidate constellation. The user can accept the decision of the framework or evaluate the metrics and chose a different constellation.

3.1.6 Radio Description File

The end result of the analysis phase of the RapidRadio framework is the creation of the RDF. The RDF is a platform independent, high-level description of the architecture of the receiver required to demodulate the analyzed signal. Because of the large number of parsing libraries available, XML is used for the RDF. XML also provides a structured format that is human-readable. No specific device architectural features or platform information is included in the RDF to ensure that it can be used to build a receiver on any platform. The RDF is stored in a database along with the $\hat{\beta}$ and other information, such as location and time of day. This allows the system to learn from the analysis of previous signals. RapidRadio could use a radio environment map, such as that proposed in [5]. This would allow it to associate a previous solution with a time and location. When a new signal is first encountered, the database of
previously classified signals can be consulted. If a similar signal has been classified before, then the solution in the database could be attempted first.

## 3.2 Radio Synthesis

The second phase of the RapidRadio framework is the platform specific implementation of the design detailed during the analysis phase. The synthesis process receives as input two XML files. The first of these is the RDF that was discussed above and is shown in Figure 3.3. The second input file is a platform description file which contains information about the target platform. The extent to which synthesis is performed depends on the resources available to the RapidRadio platform. A smaller platform such as a UAV, which lacks the resources to perform radio synthesis can perform signal analysis and transmit the resulting RDF to a platform with more resources. The AWAC performs radio synthesis and transmits the configuration to the UAV.

The use of lightweight run-time synthesis tools such as Wires on Demand [85, 14] would allow platforms to perform much of the synthesis on-board, only relegating to the base station the creation of new modules. A UAV that is building a radio but lacks some of the modules sends out a query to other nearby UAVs. After they have responded with the list of modules they possess, specific module requests to each can then be made. If the platform contains ample resources, such as a PC, then there is no limit to the system which can be built. Regardless of the platform, the radio synthesis flow consists of two major efforts: module instantiation and inter-module connection generation. These two steps are discussed below.

### 3.2.1 Module Instantiation

Module instantiation is the process of mapping the abstract description of the module in the RDF to an architecture-specific implementation. The first step in module instantiation
is verifying if the module needed already exists in the system. For modules such as the synchronizers this will probably be the case. These modules are not easily recreatable and will be frequently used, so the system is likely to have them. Other modules such as an Finite Impulse Response (FIR) filter are customized to a given signal and may not have been previously created. If a copy of the module exists, then the framework will verify that it meets all the requirements. Some requirements are applicable to all modules, such as the device vendor and type. Others are module specific such as the center frequency for an FIR filter.

If a module does not exist locally, then it can either be created or requested from an outside entity. A UAV for example, lacking the resources to create the module itself might could request a module from a base-station or some other vehicle within transmission range. A platform with more resources can automatically generate the required module. A database of generation instructions can be kept indicating the procedure to build the module. This process is shown in Figure 3.5

### 3.2.2 Inter-Module Connections

Automatically creating connections between modules is a complicated task that is often simplified by assuming all modules have the same interface. This can be seen in SoC generation tools such as the one presented in [51] as well as in the SCA which uses CORBA [25, 26, 24, 27]. With the increase in module re-use and the lack of standardization, future systems will need to interconnect modules with different interfaces. The RapidRadio framework can implement a strategy similar to that presented in [86, 74], where all a module’s ports are grouped into high level interfaces. Connections are then specified between interfaces. A database of rules is then used to define how to interconnect these interfaces. In some cases, glue logic must be inserted to account for different communications architectures. To connect a block-based architecture to a stream-based architecture, for example, a FIFO is
Figure 3.5: Module instantiation flow.
required to convert a transfer of a block containing $n$ inputs into $n$ transfers of one input. Using this strategy means that module interfaces need not be changed to be added to the framework, as long as the proper interconnection rules are provided.

3.2.3 Configuration Generation

Unfortunately FPGA bitstream generation is a process that mostly remains the domain of vendor tools. Using Wires on Demand the RapidRadio framework can assemble bitstreams from pre-built modules, but creating new modules is not possible. Stand-alone systems will need to offload bitstream generation to base-stations unless all modules are available locally. Many radios however are not stand-alone systems, but part of a larger system (i.e. UAV, plane, tank, etc...). A client-server type of architecture can be created to allow the radio to offload more resource-intensive applications such as bitstream generation to a more powerful part of the system. Although the discussion in this section has been in the context of FPGAs, it is important to note that most configurable platforms share the same limitation observed in the configuration generation process of FPGAs. Additionally, since the RDF is architecture independent, any configurable platform can be the target of the radio synthesis phase.
Chapter 4

Concept Prototype

This chapter presents a discussion of the prototype RapidRadio framework. First an overview of the phase shift keying modulation family is presented. This is followed by a discussion of the algorithms chosen for the analysis phase and the current implementation. Lastly, the receiver synthesis architecture is discussed.

4.1 Phase Shift Keying

Phase Shift Keying (PSK) is a commonly used digital modulation scheme where the phase offset of a reference signal, usually a sinusoid, is used to transmit data. Prior to transmission, the data to be transmitted is broken-up in \( n \)-bit patterns called symbols. Each symbol is assigned to one of \( 2^n \) phases. Because of the binary nature of the communications scheme, the number of symbols is always a power of two. The most basic form of PSK is Binary Phase Shift Keying (BPSK), which as the name suggests, uses two symbols.
4.1.1 Binary Phase Shift Keying

In BPSK a sine or a cosine wave is used as the reference signal (also known as the carrier wave). A phase difference of 180° is used to differentiate between symbols. When a cosine is used as the reference signal, the two symbols are defined as follows:

\[ s_1(t) = A \cos(2\pi f_c t) \] (4.1)

\[ s_2(t) = A \cos(2\pi f_c t + \pi) = -\cos(2\pi f_c t) \] (4.2)

where \( f_c \) is the frequency of the carrier wave and \( A \) is the amplitude. Notice that the value represented by a symbol is independent of the symbol and how it is transmitted. Although \( s_2 \) generally represents a zero the value it represents is implementation dependent. Because \( s_2 = -s_1 \) the transmitted signal can be expressed as:

\[ x(t) = m(t) \cos(2\pi f_c t) \] (4.3)

where \( m(t) \) is a signal which takes on the values of \( A \) or \(-A\). The symbol rate \( f_s \) denotes the frequency with which \( m(t) \) changes and is defined as:

\[ f_s = \frac{1}{T_s} \]

where \( T_s \) is the time required to transmit one symbol. Figure 4.1 shows a sample generated signal for a message consisting of alternating symbols.
4.1.2 M-ary Phase Shift Keying

M-ary PSK is a generic term to indicate a PSK modulation scheme with $M$ symbols. Equations 4.1 and 4.2 can then be generalized for the $i^{th}$ symbol as follows:

$$s_i(t) = A \cos \left( 2\pi f_c t + \left( i \frac{2\pi}{M} + \theta \right) \right) \tag{4.4}$$

where $\theta$ is a phase offset used to rotate the symbols and $0 \leq i \leq M$. Expanding Equation 4.4 gives us:

$$s_i(t) = A \cos(2\pi f_c t) \cos \left( i \frac{2\pi}{M} + \theta \right) - A \sin(2\pi f_c t) \sin \left( i \frac{2\pi}{M} + \theta \right) \tag{4.5}$$

Now, the in-phase component of the signal, $I_i(t)$, and the quadrature component of the signal, $Q(t)$ can be defined as:

$$I_i = A \cos \left( i \frac{2\pi}{M} + \theta \right) \tag{4.6}$$

$$Q_i = -A \sin \left( i \frac{2\pi}{M} + \theta \right) \tag{4.7}$$

Equation 4.5 can be restated as:

$$s_i(t) = \cos(2\pi f_c t) I_i + \sin(2\pi f_c t) Q_i \tag{4.8}$$

Symbols are often represented as a complex vector where $I$ corresponds to the real part of the vector and $Q$ corresponds to the imaginary part.

$$s_i = I_i + jQ_i \tag{4.9}$$

Note that substituting values of $M = 2$ and $\theta = 0$ in Equation 4.8 results in equations 4.1 and 4.2 for $i = 0$ and $i = 1$ respectively. Figure 4.2 shows the constellations for BPSK, Quadrature Shift Keying (QPSK) and 8-PSK. Because the PSK family of modulation schemes have constant amplitude, the magnitude of vector $s$ is the same for all symbols. This can be seen in Figure 4.2.
Figure 4.2: Common Phase Shift Keying constellations.
4.1.3 Raised Cosine

The Raised Cosine (RCOS) filter is commonly used in narrow-band digital communications schemes because it reduces Inter-Symbol Interference (ISI) and limits the bandwidth of the signal [87]. The transfer function, $H(f)$, of the filter is defined as follows:

$$H(f) = \begin{cases} 
1, & |f| \leq f_1 \\
\frac{1}{2} \left\{ 1 + \cos \left[ \frac{\pi(|f| - f_1)}{2f_\Delta} \right] \right\}, & f_1 \leq |f| \leq B \\
0, & |f| > B 
\end{cases} \quad (4.10)$$

where $B$ is the absolute bandwidth, $f_\Delta = B - f_0$ and $f_1 \triangleq f_0 - f_\Delta$. $f_0$ is the 6dB bandwidth of the filter. Figure 4.3 shows the frequency response of the RCOS filter and its defined parameters. The roll-off factor, $\alpha$, of the filter is defined as:

$$\alpha = \frac{f_\Delta}{f_0}$$

The impulse response of the filter is given by:

$$h(t) = \mathcal{F}^{-1}[H(f)] = 2f_0 \left( \frac{\sin 2\pi f_0 t}{2\pi f_0 t} \right) \left[ \frac{\cos 2\pi f_\Delta t}{1 - (4f_\Delta t)^2} \right] \quad (4.11)$$

It is shown in Figure 4.5 that the time response of the filter is a sinc function which has a zero every $T_s$. When the signal is perfectly sampled (i.e. $t = nT_s$), there is almost no ISI. From examining Figures 4.4 and 4.5, it can be seen that the value of $\alpha$ provides a trade-off between ISI and utilized bandwidth.

4.2 Analysis Phase

The analysis phase is implemented in Matlab with a Graphical User Interface (GUI) front-end designed to guide the user though the process of signal classification and receiver deployment. The process is mostly automated requiring minimal user input to select the signal to be
Figure 4.3: Raised Cosine filter definition.

Figure 4.4: Frequency response of Raised Cosine filter with $\alpha = \{0, 0.50, 1\}$.

Figure 4.5: Time response of Raised Cosine filter with $\alpha = \{0.25, 0.50, 0.75\}$. 
classified and to select the constellation to be used for the deployed receiver. The following sections describe the implementation, as well as the basic theory behind each of the key elements of the signal analysis phase. This discussion is framed from the user’s point of view. The GUIs are first presented and the underlying operations are then discussed.

4.2.1 Artificial Intelligence Engine

A staple of signal classification systems is their ability to adapt to the environment and to learn from past experiences. To this end, the C Language Integrated Production System (CLIPS) rule-based artificial intelligence engine has been integrated into the RapidRadio framework [88]. CLIPS was chosen as the intelligence engine due to its natural language syntax and because it is a well understood platform. A Java Native Interface (JNI) provided with the CLIPS package is used to allow the Matlab-based classification system to interface with the intelligence engine. Basic system facts are stored in CLIPS in order to keep the Matlab code as platform agnostic as possible.

4.2.2 Signal Selection and Isolation Interface

Upon initialization of the framework, the user is presented with the GUI shown in Figure 4.6. This interface allows the user to obtain a sample of the spectrum, isolate a small portion of the spectrum and obtain an estimate of the modulation parameters of the isolated signal. Once the user is satisfied with the results, pressing the “Next” button proceeds with the second half of the analysis phase.
Figure 4.6: RapidRadio GUI for signal selection and isolation.
4.2.3 System Initialization

System initialization is triggered when the “Acquire Signal” button shown in area one of Figure 4.6 is pressed by the user. The initialization process is shown in Figure 4.7. The CLIPS engine is initialized to eliminate any residual data from previous runs of the framework. A set of default facts stored in CLIPS are then used to initialize the framework.

First, the location of the constellation description files (XML) is obtained from CLIPS. The framework loads the constellations into memory and adds them to the CLIPS fact set. For each constellation the following data is obtained:

- **Constellation points:** The points of the constellation normalized to have unit power.
- **Amplitude profile:** A list of the different magnitudes that can be observed by the constellation’s symbols. The probability of each magnitude is also calculated by dividing
the number of symbols with the given magnitude by the total number of symbols.

- **Phase profile**: A list of all the different phase changes that can occur for the given constellation. The probability of each phase is calculated by dividing the number of transitions that have the given phase by the total number of valid transitions.

- **Transition matrix**: A list of all the valid symbol transitions and their probabilities.

- **Bits per symbol**: The number of bits transmitted per symbol.

The operating environment is then added to the CLIPS fact-set. Using pre-defined rules, CLIPS defines the probability of each constellation. The RapidRadio framework includes three pre-defined operating environments:

- **Unknown**: There is no knowledge about the environment. All constellations are assumed to be equiprobable.

- **Urban**: A high-noise environment is assumed. Constellations with less than four bits per symbol are assumed to be three times as probable as those with four or more bits.

- **Microwave**: A low-noise environment is assumed. Constellations with four or more bits per symbol are assumed to be four times as probable as those with three or less bits.

New environments can be easily added by modifying the CLIPS rule set. New rules are automatically used, because the rule-sets are always re-loaded during initialization.

### 4.2.4 Spectrum Sampling

Obtaining a sample of the spectrum is initiated by pressing the “Acquire Signal” button shown in area one of Figure 4.6. The bitstream used to sample the spectrum and the
Figure 4.8: 132k-point Discrete Fourier Transform (left) vs. Bartlett Periodogram with 64 windows (right).

Current sampling frequency of the bitstream is obtained from the CLIPS fact-set (this is shown in Figure 4.7). Currently a bitstream with a sampling frequency of 8 MHz and a buffer size of 132k 12-bit samples is used. The Matlab front-end establishes a TCP/IP link with a daemon running on the target embedded platform. Through this link the front-end loads the bitstream and extracts a sample of the signal. The signal is then converted to the frequency domain to allow examination of the spectrum.

Typically, when examining the spectrum of a signal, obtaining the DFT would suffice. The DFT, however, has a high variance, resulting in a noisy signal. To alleviate this, a periodogram using the Bartlett method [89] is used. In the Bartlett method, the signal is broken up into $L$ windows of equal size and the DFT of each window is calculated. The final periodogram ($P(f)$) is simply the average of all the DFTs. Figure 4.8 shows both the DFT and the Bartlett periodogram of a signal using 64 windows of 132k-point FFTs. Although using a large $L$ will considerably reduce noise, it reduces the resolution.
4.2.5 Signal Isolation

Area two of Figure 4.6 is used to show the periodogram of the captured signal. An $L = 4$ is used to allow the user to zoom into small areas without much loss of precision. The signal isolation panel shown in areas three of Figure 4.6 is then used to zoom into the desired signal. Sliders allow the user to change the center and width of the data displayed in area two. The value of the “Center” slider is used as the initial estimate of the carrier frequency ($\hat{f}_c$). Only the part of the base signal that is showing will be analyzed. This allows the user to select a portion of the spectrum to analyze. For optimal performance, the signal should occupy approximately 75% of the graph and be roughly centered. The remaining 25% must not have signal above the noise floor, otherwise subsequent steps may fail. Figure 4.9 shows a properly selected signal. Pressing the “Isolate Signal” button then invokes the modulation parameter estimation process.
4.2.6 Modulation Parameter Estimation

Throughout the analysis phase, only two assumptions about the signal are made. The first is that the signal is linearly modulated. Although this is a limiting factor, it is not an unreasonable assumption because linear modulations have widespread use. The second assumption made is that a root RCOS filter is used at the transmitter to limit bandwidth. This raised cosine filter is a very common filter because it limits the bandwidth of the signal and reduces ISI (see Section 4.1.3 for a description of the RCOS filter). Because the ideal shape of the signal is known, an estimate of the carrier frequency ($\hat{f}_c$), symbol rate ($\hat{f}_s$), roll-off factor ($\hat{\alpha}$), the top of the signal ($\hat{A}_{up}$) and the noise floor ($\hat{A}_{dn}$) can be obtained. Given the set of unknowns $\beta = \{f_c, f_s, \alpha, A_{up}, A_{dn}\}$ the spectral shape of the incoming signal, $y(f, \beta)$, can be defined as the following piece-wise function:

$$y(f, \beta) = \begin{cases} 
A_{dn} & f \in [0, f_1) \\
A_{dn} + \frac{A_{up} + A_{dn}}{2} \left[1 - \cos\left(\frac{f - f_1}{\alpha f_s} \pi\right)\right] & f \in [f_1, f_2) \\
A_{up} & f \in [f_2, f_3) \\
A_{dn} + \frac{A_{up} + A_{dn}}{2} \left[1 + \cos\left(\frac{f - f_2}{\alpha f_s} \pi\right)\right] & f \in [f_3, f_4) \\
A_{dn} & f \in [f_4, \infty) 
\end{cases}$$

(4.12)

where:

$$f_1 = f_c - \frac{1}{2}(1 + \alpha)f_s$$

(4.13)

$$f_2 = f_c - \frac{1}{2}(1 - \alpha)f_s$$

(4.14)

$$f_3 = f_c + \frac{1}{2}(1 - \alpha)f_s$$

(4.15)

$$f_4 = f_c + \frac{1}{2}(1 + \alpha)f_s$$

(4.16)

Figure 4.10 shows the expected spectral shape.

Finding the best $\hat{\beta}$ is then a non-linear Least Square optimization problem that can be stated as:

$$\min_{\beta} \|y(f, \beta) - P(f)\|^2 \quad f \in \{0, 1, 2, \ldots, N/2 - 1\}$$

(4.17)
where $N$ is the number of points in the Fast Fourier Transform (FFT). Equation 4.17 is then solved using a combination of the Newton step method and the trust region method. The Newton Step method uses linear approximations of $y(f, \beta)$, obtained from $\partial y(f, \beta) / \partial \beta$, to determine the modification to $\hat{\beta}$ that reduces the error. The difficulty with using the Newton method is that the calculated step may result in an increase in the error instead of a decrease. To prevent this mis-step the trust region method is used. The trust region method defines a maximum step size known as the trust region. If a calculated step results in an increase in the error, then the trust region is reduced thereby reducing the maximum step size. This process is discussed in detail in [84]. Figure 4.11 shows a sample spectral fitting of a signal.
4.2.6.1 Estimate Accuracy

Equation 4.17 tells us that the estimate of $\beta$ is affected by all values of $y(f, \beta)$ for $f$ between 0 and $N/2 - 1$, but we can see from Figure 4.10 that we are only interested in $y(f, \beta)$ for values $f$ between $f_1$ and $f_4$. Using Equations 4.13 and 4.16 the width of the area of interest ($w$) can be defined as:

$$w = f_4 - f_1 = f_c + \frac{1}{2}(1 + \alpha)f_s - \left( f_c - \frac{1}{2}(1 + \alpha)f_s \right)$$

$$= (1 + \alpha)f_s \quad (4.18)$$

Given that $N$ is evenly distributed between $-F_s/2$ and $F_s/2$, the spectral separation between points in the periodogram is $F_s/N$. The number of points in the periodogram that fall in our area of interest ($N_w$) can then be stated as:

$$N_w \approx \frac{w}{F_s/N} \quad (4.19)$$

To increasing the accuracy of the spectral fitting $N_w$ should be as large as possible. Because $w$ is fixed the $F_s/N$ ratio must be decreased to increase $N_w$. Although $N$ could be infinitely increased, the computational complexity would prove too high. A simpler solution is to reduce $F_s$ by bringing the signal down to baseband before performing the spectral fitting. Using a set of pre-defined downconverting bitstreams, a pseudo baseband signal is obtained. Downconverting bitstreams use a complex mixer and $\hat{f}_c$ to bring the signal down to baseband. Lowpass decimating filters are then used to reduce $F_s$.

The CLIPS fact-set is then searched to find the baseband bitstream with the smallest bandwidth that is greater than half the bandwidth requested by the user. Using the TCP/IP link discussed above, the bitstream name and $\hat{f}_c$ are provided to the target platform and a new pseudo-baseband signal is obtained, consisting of 32k consecutive complex samples. The Bartlett periodogram of the complex baseband signal is then obtained with $L = 16$. Because the bandwidth of the bitstream can be greater than the desired bandwidth the periodogram
is clipped to only reflect the bandwidth requested by the user. Using a baseband signal reduces $F_s$ which increases $N_w$ thereby increasing the accuracy of $\hat{\beta}$.

The initial estimates for the unknowns are obtained from the user’s specification of the signal to analyze. $\hat{A}_{up}$ is set to three quarters the maximum value of $P(f)$ in the specified range. $\hat{A}_{dn}$ is set to the minimum value of $P(f)$ in the specified range. $\hat{f}_c$ is set to zero because a baseband signal is assumed. $\hat{f}_s$ is set to half the width of the specified range. Finally, $\hat{\alpha}$ is set to 0.5.

### 4.2.7 Symbol Timing Recovery

Symbol timing recovery circuits attempt to determine the optimal moment in which to sample the incoming signal to extract the transmitted symbol. The importance of this can be seen in Figure 4.12 that shows a BPSK signal with four samples per symbol. Assuming that all samples are taking at time $4t + n$, the figure shows which samples would be used for $n = 0, 1, 2, 3$. Given that the signal has no noise and no carrier frequency error, the magnitude of the sampled signal should remain relatively constant. From looking at Figure 4.12 it can be seen that this latter restriction is only true for $n = 0$. Figure 4.13 shows the extracted symbols on the I/Q plane for the all values of $n$.

Many symbol timing recovery circuits, with varying levels of flexibility, have been proposed in the literature. In [90] an early-late synchronization algorithm is presented, capable of operating a various symbol rates, but a training sequence is required. An efficient phase independent synchronizer is presented in [91], but only phase modulated signals are supported.

To provide maximum flexibility the chosen timing recovery architecture should work with little to no change for all linear modulation schemes. The architecture must also be rotation agnostic because carrier recovery is done after symbol synchronization. Initially a Kalman tracking system with a Gardner-based timing error measurement was considered [92, 93]. The
timing error detector proposed by Gardner measures the error when a transition between symbols occurs. Symbol transitions are assumed to occur when there is a sign change in either in the in-phase or the quadrature component. This method of measuring transitions however, is only effective for BPSK and QPSK. For higher-order, constant amplitude constellations a more restrictive test was required based on the phase difference of the two symbols. An additional restriction of the Gardner-based architecture is that it can only measure errors between two symbols of similar magnitudes. Several tests were devised to ensure that the latter restriction was met, but such tests could not be easily implemented on an FPGA. Initial simulations also showed that the architecture did not function well under moderate to low noise levels.

As an alternative to the Gardner-based architecture, a modified passband timing recovery synchronizer with an oversampling rate of four is used [94, 95]. This architecture was chosen because it extracts the timing information from a spectral component in the signal, making it independent of the actual modulation scheme being used. The overall architecture of the symbol timing recovery module is shown in Figure 4.14. To extract the symbol timing from the received signal a spectral line generator is used. In the generator, the baseband signal
Figure 4.13: Plots of the obtained symbols for BPSK signal. (a)n=0 (b)n=1 (c)n=2 (d)n=3

Figure 4.14: Symbol timing recovery architecture
Figure 4.15: Spectral line generator

is filtered through two complex passband filters with center frequencies at $f = \pm \frac{1}{2T_s}$, where $T_s$ is the symbol time. The output of the filter at $f$ is then multiplied by the conjugate of the delayed version of the output of the filter at $-f$. The imaginary part of the complex multiplication is then filtered through a third passband filter. A center frequency of $\frac{1}{T_s}$ eliminates most of the noise, leaving a signal at the symbol rate. Finally, a limiter is used to eliminate residual amplitude modulation in the spectral line. Figure 4.15 shows the block diagram of the spectral line generator.

Figure 4.16 illustrates the frequency response of the first two filters overlaying the spectrum of the input signal (dashed line). The bandwidth of these filters dictate the circuits tolerance to frequency errors. If the nominal symbol rate is known, narrowband filters can be used to reduce the jitter of the recovered timing signal. For the RapidRadio architecture, a wider bandwidth was chosen to allow for symbol rate errors due to lack of knowledge of the nominal value.

A PLL is used to generate a local copy of the phase and frequency locked symbol clock. The PLL has a simple PI loop filter and an NCO. The value of the NCO’s accumulator is used to signal the correct sampling time. When the accumulator’s value has crossed a certain threshold, the system samples the incoming signal and produces a symbol. The ideal threshold ($\gamma_i$) is set to one quarter of the maximum value ($A_{mx}$) of the NCO’s ramp. Using a static threshold does not account for the fact that the number of samples per signal changes over time and may not be exactly four. The actual threshold value ($\gamma_a$) needs to be adjusted to match the number samples per symbol based on the increment value ($\lambda$) of the NCO.
and the phase response of the spectral line generator. Because the phase response of the spectral line generator is non-linear, a linear function is used to approximate its value in the neighborhood of the ideal threshold. Figure 4.17 shows the phase response of the spectral generator and the linear approximation. The threshold value can then be defined in terms of the ideal value, the NCO increment and the slope of the phase response:

$$\gamma_a = \gamma_i - 16 \left( \frac{\lambda}{A_{mx}} - \frac{1}{4} \right)$$  \hspace{1cm} (4.20)

The input signal should be sampled when the value of the accumulator $A(t)$ crosses the threshold value between two consecutive samples at times $t_{n-1}$ and $t_n$. There are three possible ways in which this condition can be meet. The first, shown in Figure 4.18(a) is when $A(t_{n-1}) < \gamma_a \leq A(t_n)$. Alternatively, if $\gamma_a$ is relatively low, the accumulator may have wrapped between the two samples. In this case a crossing is detected if $A(t_n) < A(t_{n-1})$ and $\gamma_a < A(t_n)$. This can be observed in Figure 4.18(b). A wrapping of the accumulator can also be observed if $\gamma_a$ is close to $A_{mx}$. From Figure 4.18(c) it can be seen that this can be
Figure 4.17: Phase response of the spectral line generator
detected if $\gamma_a < A(t_{n-1})$ and $A(t_n) < A(t_{n-1})$. When either of the three conditions is met a strobe is said to have occurred.

Given that $\gamma_a$ represents the perfect sampling time, the distance ($d$) between $\gamma_a$ and $A(t_{n-1})$ represents the symbol timing error. From looking at Figure 4.18 it can be seen that for cases one and three the distance is:

$$d = \gamma_a - A(t_{n-1}) \quad (4.21)$$

for case two it is:

$$d = A_{mx} - A(t_{n-1}) + \gamma_a \quad (4.22)$$

The value of $d$ represents the distance normalized to $A_{mx}$. To obtain the timing error ($\tau$), $d$ must be normalized to one sample time. The timing error is then expressed as:

$$\tau = \frac{d}{\lambda} \quad (4.23)$$

because $\lambda$ is the amount the NCO’s accumulator is increased at every sample.

Correcting for the timing error is accomplished by resampling the signal using a second order Lagrange polynomial. From (4.21) and (4.22) it can be observed that $\tau$ is measured in reference to the sample at time $t_{n-1}$. The interpolation is then performed over the sample set $\{t_{n-2}, t_{n-1}, t_n\}$. Assigning time indexes -1, 0 and 1 respectively to the samples, the
interpolation polynomial for symbol \( s_m \) is defined as:

\[
\ell_0 = \frac{\ddot{x}(t_n)+\ddot{x}(t_{n-2})}{2} - \ddot{x}(t_{n-1})
\]

(4.24)

\[
\ell_1 = \frac{\ddot{x}(t_n)-\ddot{x}(t_{n-2})}{2}
\]

(4.25)

\[
\ell_2 = \ddot{x}(t_{n-1})
\]

(4.26)

\[
s_m = \tau^2 \ell_0 + \tau \ell_1 + \ell_2
\]

(4.27)

### 4.2.8 Carrier Recover

The output of the timing recovery circuit is a stream of symbols which may be rotating due to carrier frequency error. An efficient derotation architecture is presented for 16QAM in [96] and [97]. The RapidRadio architecture expands on this work to increase the derotator’s flexibility and tolerance to frequency errors. The derotator architecture can be seen in Figure 4.19.
The incoming symbol \( (s(n)) \) is de-rotated using the predicted phase error \( (\theta_p(n)) \). A constellation-specific slicer is used to determine the closest constellation symbol \( (\hat{s}(n)) \). The slicer also produces the phase of \( \hat{s}(n) \) \( (\theta_{\hat{s}}(n)) \). The measured phase error \( (\theta_e(n)) \) is the difference between the phase of the \( s(n) \) \( (\theta_s(n)) \) and \( \theta_{\hat{s}}(n) \). The traditional PLL is replaced with a Kalman filter that tracks both the phase error and the frequency \( (f_d) \). The tracking capability of the Kalman filter allows the measurement of the absolute difference between the input phase and the phase of the symbols at the output of the slicer. The Kalman filter has two stages: the prediction stage and the update stage. In the former, the filter attempts to predict the value of the state variable \( (\hat{x}_{k|k-1}) \) based on all the previous values. In the latter stage, the predicted value is updated to reflect the measured value.

The prediction stage is mathematically defined as:

\[
\begin{align*}
\hat{x}_{k|k-1} &= F\hat{x}_{k-1|k-1} \\
P_{k|k-1} &= FP_{k-1|k-1}F^T + Q
\end{align*}
\]  

where \( F \) is the transition matrix, \( P \) is the error covariance of \( \hat{x} \) and \( Q \) is the process noise covariance. The update stage is defined by the following equations:

\[
\begin{align*}
\tilde{y}_k &= z_k - H\hat{x}_{k|k-1} \\
S_k &= HP_{k|k-1}H^T + R \\
K_k &= P_{k|k-1}H^T S_k^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k\tilde{y}_k \\
P_{k|k} &= (I - K_k H)P_{k|k-1}
\end{align*}
\]  

where \( z_k \) is the measured value, \( H \) is the measurement matrix, \( R \) is the measurement error covariance and \( K_k \) is the Kalman gain matrix. The parameters used for the filter in the derotator are shown in Table 4.1.

From Equations 4.31 and 4.32 it can be seen that \( K \) is a function of \( P \) and \( R \) and is not affected by \( z \). One of the properties of the Kalman filter is that \( P \), and therefore \( K \), converge
to a value over time if $R$ and $Q$ are constant. Fixing $K$ to a constant then reduces the filter to Equations 4.28, 4.30 and 4.33, resulting in a reduced complexity system.

The derotator architecture can be tailored to any modulation type by passing the constellation description into the slicer. Although every constellation has an optimal slicer architecture, for RapidRadio a generic approach to constructing the slicer was desired. The developed approach is capable of producing a correct slicer for any linear constellation based on an XML description file of the constellation. The framework supports hardware optimizations for two different constellation classes, deciding at implementation-time which one is most suitable for the constellation.

The first architecture is used for constellations that form a perfect grid, such as 16QAM and 64QAM. It partitions the I/Q plane into $N$ equally sized squares, where $N$ is the number of symbols in the constellation. Each square is centered around a constellation point. A square grid is ensured to form the ideal decision boundaries because the constellation itself is a grid. Figure 4.20(a) shows the traditional 64QAM constellation, where the crosses represent the

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{x}$</td>
<td>$\begin{bmatrix} \theta_p \ f_p \end{bmatrix}$</td>
</tr>
<tr>
<td>$F$</td>
<td>$\begin{bmatrix} 1 &amp; 1 \ \frac{1}{2} &amp; \frac{1}{2} \end{bmatrix}$</td>
</tr>
<tr>
<td>$Q$</td>
<td>$0.001 \cdot \begin{bmatrix} 0.25 &amp; 0.5 \ 0.5 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>$R$</td>
<td>$1$</td>
</tr>
<tr>
<td>$H$</td>
<td>$\begin{bmatrix} 1 \ 0 \end{bmatrix}$</td>
</tr>
<tr>
<td>$z_k$</td>
<td>$\theta_c(n)$</td>
</tr>
</tbody>
</table>

Table 4.1: Kalman filter implementation values
constellation points. From the figure it can be seen that partitioning the I and Q axes, into evenly spaced portions would provide ideal decision boundaries.

Figure 4.20(b) shows a variant of the 64QAM constellation [98]. This constellation reduces the average signal power, which permits increasing the Euclidean distance between the symbols while using the same power as the constellation shown in Figure 4.20(a). Notice, that a grid-based slicer is not an efficient architecture for the 64QAM variant. For constellations that cannot be optimally sliced with a grid, a squared Euclidian distance slicer is proposed. The distance architecture calculates the square of the distance between the received symbol and every point in the constellation. The constellation point with the lowest value is considered to be the correct symbol. The distance architecture can be computationally expensive for high-order constellations, but it will provide optimal slicing for any constellation. Additionally, pipelining the computations can reduce the resource requirements.

When constructing the derotator, the RapidRadio framework determines what slicer architecture is to be used and then creates the appropriate slicer. Although the distance-based slicer works for all constellations, the grid-based slicer is included because of the popularity
of grid constellations and its computational simplicity. An algorithm was devised to test the geometric shape of the constellation. The slicer architecture is then chosen based on the return value of the algorithm.

Part of the tailoring process is to adjust the magnitude of the symbols in the slicer based on the average signal power. Both slicer architectures assume that a gain control circuit ensures that the average power of the signal remains constant. From Figure 4.19 it can be seen that the slicer must also produce a measure of the symbol phase, \( \theta_s \). These values are hard-coded in a look-up table in the slicer because they are based on the ideal constellation points that do not change over time.

### 4.3 Hypothesis Fitness Evaluation

For each hypothesized constellation, the output of both the symbol synchronizer and the de-rotator is evaluated to determine the correctness of the demodulation. The evaluation is based on four metrics. The phase profile (pp) measures the change in phase between two consecutive symbols. The amplitude profile (ap) measures the magnitude of the received symbols. The symbol distribution (sd) measures how the symbols are distributed in the \( I/Q \) plane. The final metric is the symbol transition matrix (tm) which measures how often a transition between two specific symbols is observed.

All four metrics are calculated by comparing the theoretical probability distribution function (PDF) of the data with the empirical PDF. The empirical PDF is obtained by grouping the received data points in a histogram. Bin sizes for the histogram are calculated using Scott’s formula for optimal bin size [99]. The count for each bin is then divided by the total number of data points obtained. The theoretical PDFs are obtained from models adjusted for the level of noise observed. The two PDFs are compared using the Hellinger distance (H) [43, 44]. The Hellinger distance is a measure of how similar two PDFs are. It takes values in the range
of $(0, 1)$, where a zero indicates the PDFs are identical and a value of one indicates they are completely different. The metrics for all the hypothesis are combined using a Bayesian network to determine the likelihood of each constellation. The GUI used to present the data to the user and the Bayesian network are discussed below.

The hypothesis evaluation GUI presents the data used to obtain the metrics in graphical form for evaluation by the user. Figure 4.21 shows the GUI. The panel on the left marked with a one displays all the hypothesized constellations along with their scores. The hypothesis are ordered based on their likelihood, with the most likely constellation on the top. By selecting a constellation in this panel, the user can view the data used by the framework to formulate its decision. Navigating through the different hypothesis allows the user to make an informed decision on whether the framework chose the correct constellation or not. Areas marked two through five show the data used for the phase profile, the amplitude profile, the symbol distribution and the transition distribution respectively.

When the data is three-dimensional, as in the case for the symbol and transition distributions, only one data set is shown at a time. A switch, marked as six in Figure 4.21 is used to switch from showing the empirical data to showing the theoretical data, permitting the comparison of the two by the user. An additional button, marked as seven, changes how the symbol distribution data is presented. A contour view is provided for the symbol distribution because the viewpoint of the three dimensional mesh plot is fixed and may hide data. The contour view can also provide the user with a better understanding on how the symbols are clustered in the $I/Q$ plane.

Pressing the “Generate Current” button will start the process of receiver deployment for the constellation that is currently selected in the constellation panel of the GUI. First, the modulation parameters are refined to reflect all the known information. Then, an estimate of the average signal power is obtained. Using these values a Radio Description File is generated and receiver synthesis is started.
Figure 4.21: RapidRadio hypothesis evaluation Graphical User Interface.
4.3.1 Phase Profile

The phase profile examines the change in phase between consecutive received symbols. The theoretical Probability Density Functions (PDF) is obtained using the valid transitions for the hypothesized constellation. This does not account for errors due to signal noise however. Given that an estimate of the noise is obtained by the parameter estimator, the theoretical PDF can then be adjusted to reflect the phase variance due to the variance in the symbols. The PDF of the phase difference between two vectors perturbed by Gaussian noise is given as [100]:

\[ f(\psi) = \frac{1}{2\pi} \int_{0}^{\pi/2} \exp(-\rho(1 - \cos(\psi)\cos(\theta)))(1 + \rho + \rho \cos(\psi)\cos(\theta))\cos(\theta)d\theta \]

where \( \rho = 1/\sigma^2 \), \( \sigma^2 \) is the variance of the gaussian noise and \( \psi = \theta(t) - \theta(t - 1) \) with zero mean. Let \( \zeta \) equal the expected phase difference for a symbol transition and \( \psi_i \) be the phase difference with mean \( n \), then:

\[ f(\psi_\zeta) = f(\psi_0 - \zeta) \]

The theoretical PDF is then:

\[ f_{pp|c}(x) = \sum_{i=0}^{N-1} f(\psi_0 + \zeta_i) \]

where \( N \) is the total number of expected phase changes for a given constellation \( c \).

This metric is obtained prior to derotation because the derotator modifies the symbols phase according to the hypothesized constellation. The rotation, however, can be expressed as a constant phase error \( (\theta_e) \) and Equation 4.35 can be restated as:

\[ f_{pp|c}(x) = \sum_{i=0}^{N-1} f(\psi_0 + \zeta_i - \theta_e) \]

The value of \( \theta_e \) that results in the lowest Hellinger distance is used to calculate the value of the phase profile metric. The Hellinger distance for the phase profiles is then expressed as:

\[ H_{pp|c} = H(h_{pp}(x), f_{pp|c}(x)) \]
where $h_{pp}(x)$ is the empirical histogram. Figure 4.22 shows both the empirical phase profile as a histogram overlaid with the theoretical phase profile.

### 4.3.2 Amplitude Profile

The Amplitude profile examines the distribution of the magnitudes for the received symbols and compares it to the theoretical values. As with the phase profile, the measured magnitudes are grouped up into bins to form a histogram ($h_{ap}$). The theoretical PDF is obtained from the constellation description and is assumed to have a Rice distribution:

$$f(x, \nu) = \frac{x}{\sigma^2} \exp \left( \frac{x^2 + \nu^2}{2\sigma^2} \right) I_0 \left( \frac{x\nu}{\sigma^2} \right)$$
where \( \nu \) is the amplitude, \( \sigma^2 \) is the symbol variance and \( I_0 \) is a Bessel function with order 0. The theoretical PDF is then given as:

\[
\hat{f}_{ap|c}(c) = \sum_{i=0}^{N-1} p(\nu = \nu_i) f(x, \nu)
\]

where \( \nu_i \) is the \( i^{th} \) amplitude and \( N \) is the number of possible amplitudes for constellation \( c \).

The amplitude profile metric is the Hellinger distance between the actual distribution and the theoretical distribution and can be expressed as:

\[
H_{ap|c} = H(h_{ap}(x), \hat{f}_{ap|c}(x))
\]

Figure 4.23 shows the empirical and theoretical PDFs of the amplitude for an 8QAM signal.

4.3.2.1 Symbol Variance and Distribution

Symbol distribution examines the number of received symbols for each constellation point. Symbol variance is a measure of the distance between the received symbol and the theoretical
Figure 4.24: (a) Three dimensional plot of theoretical symbol distribution for QPSK (b) Contour plot of theoretical symbol distribution for QPSK (c) Three dimensional plot of empirical symbol distribution for QPSK (d) Contour plot of empirical symbol distribution for QPSK
location of the symbol. These two metrics are combined to form a two dimensional PDF of the received symbols. Assuming that in-phase and quadrature components of the symbol are independent random variables with Gaussian noise, the joint PDF can be expressed as:

\[ f_j(i, q) = \frac{1}{2\pi\sigma^2} \exp \left( \frac{(i - I_j)^2}{\sigma^2} \right) \]

where \( I_j \) and \( Q_j \) are the expected in-phase and quadrature values for the \( j^{th} \) symbol and \( \sigma^2 \) is variance of the noise. Assuming that the all symbols in a constellation are equally probable, the two dimensional theoretical PDF is then:

\[ f_{sd|c}(i, q) = \frac{1}{N} \sum_{i=0}^{N-1} f_i(i, q) \]

where \( N \) is the total number of symbols in constellation \( c \).

The empirical PDF is obtained using a two dimensional histogram \( h_{sd} \) to group the received symbols into bins. The Hellinger distance can be used to measure the difference between the empirical and theoretical PDFs and is expressed as:

\[ H_{sd|c} = H(h_{sd|c}(x, y), f_{ap|c}(x, y)) \]

Figures 4.24(a) and 4.24(b) show the theoretical PDFs for QPSK as a three dimensional plot and as a contour plot respectively. Figures 4.24(c) and 4.24(d) show the empirical PDFs.

### 4.3.3 Symbol Transitions

This metric compares the distribution of symbol transitions to the theoretical values. A \( M \times M \) matrix is populated with all the transitions observed, were the rows of the matrix represent the \( i^{th} \) and the columns represent the \( i^{th} + 1 \) symbol. The matrix is then divided by the total number of observed transitions to obtain the empirical PDF. The theoretical PDF is obtained from the constellation’s description file. For each symbol in the constellation,
the description file has a list or range of valid symbols a given symbol can transition to. As with other metrics, the Hellinger distance is used:

\[ H_{tm|c} = H(h_{tm|c}(x, y), f_{tm|c}(x, y)) \]

where \( h_{tm|c}(x, y) \) is the empirical transition matrix and \( f_{tm|c}(x, y) \) is the theoretical transition matrix.

### 4.3.4 Combining the Metrics

All of the metrics discussed above provide information on the fitness of a hypothesis, but cannot on their own identify the correct hypothesis. A mechanism for combining the metrics in a manner that results in a single value that can be used to rank the hypothesized constellations is needed. Additionally, the mechanism must allow for easily modifying the probability of occurrence for any constellation. Artificial intelligence blocks such as neural networks were considered as a method for combining the metrics, but because they have
to be trained, they lack the flexibility desired for the RapidRadio framework. A Bayesian network on the other hand does not require a priori training and has a natural mechanism for integrating the probability of occurrence of constellations.

The Bayesian networked used for signal classification in the RapidRadio framework is shown in Figure 4.26. Scoring is done in two stages. First the a priori probabilities of occurrence of all the constellations are pushed down to the processing node. At the same time the probability of each metric given a hypothesis is passed to the processing node. The new probability of each constellation is then calculated according to Equation 4.36 below:

$$P(h_i|pp, ap, sd, tm) = \frac{P(pp, ap, sd, tm|h_i)P(h_i)}{\sum_{i=0}^{N-1} [P(pp, ap, sd, tm|h_i)P(h_i)]}$$

(4.36)

Including the a priori probability of the hypothesized constellations in Equation 4.36 ensures that constellations that are known to be more likely have a higher probability of being chosen.

From looking at Figure 4.26 it can be seen that pp, ap, sd and tm are conditionally independent.
which allows the dividend of Equation 4.36 to be defined as:

\[ P(pp, ap, sd, tm|h_i) = P(pp|h_i)P(ap|h_i)P(sd|h_i)P(tm|h_i) \]

The probabilities for each metric are approximated as follows:

\[ P(pp|h) = 1 - H_{pp|h} \]
\[ P(ap|h) = 1 - H_{ap|h} \]
\[ P(sd|h) = 1 - H_{sd|h} \]
\[ P(tm|h) = 1 - H_{tm|h} \]

This approximation assigns higher probabilities to hypothesis with empirical PDFs that closely resemble the theoretical PDFs. Using the Hellinger distance for all metrics ensures that they are equally weighed. The result of the Bayesian network is a set of probabilities that indicate the likelihood of all constellations. Situational awareness and knowledge from past experiences can be inserted into the model by modifying the \textit{a priori} probabilities of each constellation. From Equation 4.36 it can be seen that the joint probability of the metrics given the hypothesized constellation is normalized by the sum of the joint probabilities of the metrics for all hypothesized constellations. This normalization allows the addition of new constellations with little to no change to the classification algorithm.

### 4.3.5 Parameter Refinement

Estimates for all modulation parameters are automatically obtained by the parameter estimator as part of the signal classification process. These values however do not contain the knowledge acquired during the classification process. Parameter refinement is the process of updating \( \hat{\beta} \) to reflect the newly acquired knowledge. Refinement is the last step before the Radio Description File is written because some of the information used depends on the constellation chosen.

The parameters updated during the refinement process are the symbol rate, the carrier frequency and the average signal power, amongst others. The symbol rate is obtained from
the symbol synchronizer. The FFT of the output of the Phase-Locked Loop (PLL) has a peak at the symbol rate. Examining the FFT and using the frequency value with the highest value in the FFT will then provide an accurate estimate of the symbol rate.

The carrier frequency is obtained from the initial estimate ($\hat{f}_c$) plus two correction factors ($c_1$ and $c_2$). The initial estimate of the carrier frequency $\hat{f}_c$ is obtained from the user input. This value however is a very rough estimate of the correct value, because it is obtained from the user’s interpretation of the periodogram. The signal is then corrected for carrier frequency errors in two stages. First, the parameter estimator obtains an estimate of what the carrier frequency error of the baseband signal is ($c_1$) and corrects it prior to hypothesis testing. The second frequency correction ($c_2$) is obtained from the derotator used for the synchronization of the selected constellation. From Equations 4.39 and 4.41 it can be seen that the derotator tracks the frequency error as well as the phase error. $c_2$ is then obtained by converting $f_{d|c}$ from a constant phase offset to a frequency as follows:

$$c_2 = \frac{f_{d|c}f_s}{2\pi}$$

The final center frequency is then defined as:

$$f_c = \hat{f}_c + c_1 + c_2$$

Using the estimated symbol rate and the synchronizer architecture, the CLIPS back-end then determines the required sampling rate ($F_{sr}$) at the input to the synchronizer. Based on $F_s$ and $F_{sr}$ the decimation factor ($\eta$) of the matched filter is determined as follows:

$$\max_{\eta} \left[ \frac{F_s}{\eta} \geq F_{sr} \right]$$

($4.37$)

$F_{sr}$, however is often not a factor of $F_s$, resulting in the need for a resampler. The CLIPS engine detects this condition and adds a resampler block to the RDF. The resampler used is based on the Lagrange interpolator discussed in Section 4.2.7 and can reduce the sampling rate to a maximum of one half on the input sampling rate. When the resampler is used, $F_{sr}$
needs to be duplicated to ensure that sampling frequency at the input of the synchronizer is at least twice $F_{sr}$. Equation 4.37 is then updated to reflect this as follows:

$$\max_\eta \left[ \frac{F_s}{\eta} \geq 2F_{sr} \right]$$

### 4.3.6 Radio Description File

The end result of the analysis phase of the RapidRadio framework is the creation of the Radio Description File. The RDF is a platform independent, high-level description of the architecture of the receiver required to demodulate the analyzed signal. Because of the large number of parsing libraries available, XML was chosen for the RDF. XML also provides a structured format that is human-readable. No specific device architectural features or platform information is included in the RDF to ensure that it can be used to build a receiver on any FPGA-based platform.

The following XML tags are defined for use in the RDF:

- **Module**: Specifies a hardware module to be included in the design. This tag has three attributes: Type, Subtype and Name. Type indicates the module type such as “Filter”. Subtype is used to differentiate between variants of the same module type. Name is a identifier for this module. The module’s name must be unique in the RDF.

- **Connection**: Specifies a high-level connection between two or more module interfaces.

- **Source**: Specifies the source of the connection. The source tag is only valid as a child of a Connection tag. This tag has two attributes: Interface and Module. The Module attribute indicates what module contains the source of this connection. The value specified must either match the Name attribute of a Module or must be set to “Global”. The Interface attribute indicates the name of the interface to be used as the source of the connection.
• **Sink**: Specifies the sink of the connection. The Sink tag is only valid as a child of a Connection tag. This tag shares the same attributes of the Source tag. Including more than one Sink tag under a Connections tag indicates a multi-sink connection.

• **Parameter**: Specifies the value of an input argument. A Parameter tag at the top level of the XML indicates a global parameter, such as the sampling frequency. If specified inside a Module tag, then it is providing an implementation specific parameter that is only used by the module. This tag has four possible attributes: Name, Type, Value and Text. Specifying Type=“static” indicates that the value provided is to be used for the parameter. If Type=“configurable” then the value of the parameter is obtained from some other parameter indicated by the Text tag.

A sample RDF can be seen in Appendix A.1.

### 4.4 Radio Deployment Phase

The radio deployment phase creates an FPGA-based receiver for the classified signal that produces a stream of symbols. Figure 4.27 shows the GUI used to inform the user on the progress of the build. Using a TCP/IP link the GUI starts the build on a server that hosts the vendor tools. The build process updates a log file with its progress. The GUI reads the contents of the log file on a time initiated basis and updates the display. When the build process has completed the new configuration bitstream is loaded into the target platform. A set of 81,000 non-contiguous symbols is then extracted from the platform and displayed on the GUI. A sample set of symbols is shown on the right hand side of Figure 4.27.
Figure 4.27: Radio deployment Graphical User Interface

Figure 4.28: Radio synthesis flow.
4.4.1 Radio Synthesis

A high-level description of the radio synthesis phase can be seen in Figure 4.28. The synthesis tool developed is written in C++ and builds on-top of other tools such as Makefiles, Xerces-C, Matlab and Xilinx’s Core Generator. This approach was chosen over using run-time synthesis tools such as Wires on Demand because of the desire to create new modules at synthesis-time. Additionally, since some resource intensive techniques are being used no artificial limits on resources available wanted to be imposed. The Wire on Demand framework, for example limits the area available for instantiating modules and assumes all modules are geographically isolated. The latter implies that unused resources lying inside the are allocated to a module cannot be used by another module. Lastly, the framework is intended to be platform independent, but most run-time synthesis tools are tied to a specific vendor. The following sections discuss the major pieces of the radio synthesis phase.

4.4.2 Platform Description File

A goal of the RapidRadio framework is to reduce the amount of FPGA knowledge necessary to create a system. There are many parameters, however, that are platform unique, such as ADC initialization, inter-component communications and output pin usage to name a few. To hide this information from the user the framework could be made platform specific, building all the knowledge about a given platform into the system. This approach, however, is not very attractive because it would make the framework hard to modify to target a different platform.

The RapidRadio framework solves this problem by assuming that a top-level design was previously created. This top-level design serves as a host to the receiver and takes care of all the initialization and communication infrastructure. The framework therefore produces a receiver module which accepts an input data stream and produces an output data stream. This approach requires a one-time setup when a new platform is chosen to serve as a target,
but the cost of designing and testing the top level design is relatively small when compared to the cost of building an entire new design every time.

Information about the top-level design used for building a receiver on a specific platform is included in the platform description file. The file defines the interfaces to the receiver module and the ports the interfaces map to. Additionally the file includes the name used for the newly created receiver module and points to a directory in the local filesystem which contains instruction on how to build the design. Build instructions are provided in the form of a Makefile. This format was chosen because its a universally understood tool that can encapsulate the vendor tool chain.

Like the RDF, the platform description is written in XML. The following tags are used to describe the platform:

- **Chip**: Used to define a device on the platform. Embedded tags are used to provide more details on the device type. Because the defining characteristics of the chip may be vendor specific a `Vendor` attribute is used in the Chip tag. As the name implies, this attribute identifies the vendor, such as Xilinx or Altera. The vendor type defines the expected embedded elements. Accordingly the `Device`, `Package`, `Family` and `Speedgrade` embedded tags are supported.

- **Clock**: Specifies the name of the clock signal port into the receiver module. The `Freq` attribute indicates the frequency of the signal in Hertz.

- **Reset**: Specifies the name of the reset signal port into the receiver module. The `Active` attribute is used to indicate if the reset is active high or active low.

- **Name**: Specifies the module name used to instantiate the receiver module in the base design.

- **Makefile**: Specifies the absolute path in the filesystem for the Makefile used to build the base system.
• **Interface**: Specifies an interface between the base system and the receiver module to be generated. The Interface element has three attributes:

  – **Name**: Specifies the Interfaces name. The name must be unique within the platform description file.

  – **Type**: Specify the Interface type. Although not currently used, this attribute allows for the interconnection of interfaces to be rule-based.

  – **Direction**: Specifies if the interface is an input or an output interface. Note that this does not imply that all ports within the interface have the same direction.

A platform description file must contain at least one Interface element.

• **Port**: Defines a port for the receiver module. All ports, with the exception of the clock and reset ports, must be part of an Interface. Port elements have the following attributes:

  – **Name**: Specifies the Port name. The name must be unique within the platform description file.

  – **Type**: Specify the Port type. Port types are used for communication synthesis. See Section 4.4.5 for details on how it is used.

  – **Direction**: Specifies if the Port is an input or an output interface.

A sample platform description file is shown in Appendix A.2.

### 4.4.3 Module Selection and Generation

The module database shown in Figure 4.28 does not contain the modules themselves, but module description files written in XML. The file can represent pre-built modules or a set of rules dictating how to create a module. Definitions of the interfaces for the module and a
list of all the ports associated with each interface are also contained. If the implementation
is not device specific or the module has not yet been implemented, the description file will
indicate that it can match any device. Otherwise the device type information is listed. Lastly
all module parameters are listed in the file. The module description file uses some of the
tags previously defined for the RDF and the platform description file. A sample module
description file is shown in Appendix A.3.

When a module is encountered in the RDF, XML files in the module database are searched
for a matching module. Matches are based on the modules type/subtype, parameters and
the target device. Parameters that are hard-coded into the modules implementation are
shown as having a specific value. If the database module does not have the same value as
that requested in the RDF then it is not considered a match. If the database module has a
value of “configurable” for the parameter, then this parameter is ignored during the match
process because the module can be configured to use the value provided in the RDF.

Similar to the platform description file, each module file contains a directory where a makefile
can be found. The makefile contains the sequence of steps required to obtain the HDL
description of the module and any object files required. The makefile receives as input a
temporary file created by the framework that contains the RDFs globals parameters, the
platform parameters and the module instantiation parameters. Figure 4.28 shows a sample
generation script for an FIR filter. A Matlab engine is used to create the coefficients based
on the input file. The input files (.coe and .xco) to Coregen are created and Coregen is
invoked to create the module. The resulting Verilog and object files are then moved to the
build directory. Note that for static modules which cannot be modified, the generation script
will result in a simple transfer of files from the repository to the build directory.
4.4.3.1 Indirect Parameterization

Indirect parameterization allows the value of one property to be set to the value of some other property. This provides RapidRadio the ability to adapt modules to the specific platform. Instead of fixing a modules port widths, for example, the widths can be set to the value of a specific parameter. Parameter values can be cascaded through the various layers of a radios definitions because there is no limit to the levels of indirection. This is shown in Figure 4.29. The width of the filters port is obtained from the modules O.Width parameter, which in turn gets its value from the RDFs R.Width parameter, and so on. The arrows in the figure show how the flow of how parameters are defined.
4.4.4 Bit Precision

The number of bits required to accurately represent a signal is not fixed. It depends on the type of signal being received, how much noise is expected and the expected bandwidth. The initial bit width of the signal is solely based on the precision of the ADC, which can change based on the implementation platform. The RapidRadio framework assumes that the bit width of the signal at the input to the demodulator is sufficient and attempts to keep the signal the same width across the entire demodulation process.

It is important to note that the framework does not concentrate on module creation, but on the assembly of systems based on pre-existing modules, some of which are parameterized. The parameters may allow the framework to modify the input and output width of the modules, but not necessarily the internal logic. Other modules on the other hand are fixed and the framework cannot modify the input and output precision. It is, therefore, the responsibility of the module designer to ensure that internal precision of the module can tolerate a wide operating range. The precision of the specific modules used in the current prototype is discussed in Section 4.5. Although module port widths are modified when possible, the goal of the framework is not to use optimal bus widths. Determining the optimal size of busses is an area of open research [101, 102], and is not within the scope of this work.

4.4.5 Inter-Module Connection Generation

A key element of the RDF is that connections are specified at a very high-level using quasi-generic interfaces. This is necessary because when the RDF is created the actual module implementation is unknown and the exact names and number of ports cannot be determined. At synthesis time, interfaces are expanded into a list of ports and a inter-connection strategy is needed. Matching on names is not satisfactory because that would impose to severe a restriction on the modules that can be used. In the RapidRadio framework each port is
assigned a type in the module description file. Type matching within a connection provides for greater flexibility. This approach is based on the work done at IBM for the Coral project [86]. Coral is a framework for system-on-chip design that amongst other things, automatically connects IP modules. Coral uses interface descriptions to determine how ports should be interconnected and inserts glue logic when necessary.

4.5 Receiver Architecture and Implementation

Receivers deployed by the RapidRadio framework share the general architecture shown in Figure 4.30. The signals sampling rate at each block is indicated by the blocks color. A complex mixer is used to bring the signal down to baseband. Decimating root-raised cosine filters are used for matched filtering and sampling rate reductions. A resampler circuit is used to recondition the signal, reducing the sampling rate to four times the symbol rate. Synchronization is then performed to produce a stream of symbols. The following sections describe in detail the FPGA implementation of these blocks.

4.5.1 Downconverter

Using an automatically generated multiplier block and a 14-bit Numerically Controlled Oscillator (NCO) the signal is brought down to baseband. The multiplier is created us-
ing Xilinx’s Core Generator. The output width of the multiplier \( w_o \) is inherited from the platform description file. The width of the multiplicands can be different because one is the output of the NCO which has a fixed output width of 14-bits and the other is the input data which has a width equal to the input width \( w_{in} \). The output width of the multiplier is then:

\[
    w_o = 14 + w_{in}
\]

The effective output of the multiplier is \( w_o - 1 \) because both inputs are signed. To avoid bit growth the output of the multiplier is truncated, to keep it the same size as the input. The Most Significant Bit (MSB) of the multiplier block is:

\[
    MSB = w_o - 2
\]

and the Least Significant Bit (LSB) is:

\[
    LSB = MSB - w_{in} + 1
\]

### 4.5.2 Matched Filtering

Matched filters are also created using the Core Generator. The roll-off factor is obtained from the RDF. The filter order is dependent on the number of samples per symbol at the filter’s input and on the roll-off factor. The roll-off factor is used to determine how many symbols the filter should contain to minimize ISI. The length of the filter in symbols \( l_s \) is defined as:

\[
    f_l = 4 \times \left\lceil \sqrt{\frac{2}{\alpha}} \right\rceil
\]

The number of samples per symbol at the input is:

\[
    s_s = \left\lceil \frac{F_s}{f_l} \right\rceil
\]

The filter order is then defined as:

\[
    \max (64, s_s \cdot f_l)
\]
The filter’s decimation factor ($\eta$) is a function of the filter’s input sampling rate ($F_s$) and the minimum required input sampling rate of the resampler ($F_{sr}$). $\eta$ is calculated by the CLIPS back-end as described in Section 4.3.5. Filter coefficients are obtained from Matlab using the sampling rate, symbol rate and roll-off factor. Coefficients are then scaled and stored into a “.coe” file.

A filter of order $N$ has $N + 1$ multiplications and $N$ additions. Assuming that the filter coefficients are the same size as the input data, the width of the output ($w_o$) is approximately:

$$w_o = 2w_{in} + \left\lceil \log_2 \left( \frac{N + 1}{\eta} \right) \right\rceil$$

where $w_{in}$ is the input width. As with the downconverter, the output of the filter is truncated to avoid bit-growth. Selecting which bits of the output will be used however is not an obvious choice. Selecting the $w_{in}$ MSB will guarantee that the output never overflows, but may result in underflow. To obtain an estimate of which bits to use, the scaled coefficients are used to process the sampled data obtained in the spectrum sampling phase (see Section 4.2.4). The magnitude of the filtered signal is then used to determine the MSB:

$$MSB = \lceil \log_2 (\max(x_s)) \rceil$$

where $x_s$ is the filtered signal. Adding a saturation circuit at the output of the filter ensure that the output of the filter is always valid.

### 4.5.3 Resampler

The RapidRadio framework assumes that it does not control the sampling rate of the input data stream. The synchronization circuit however, requires an input sampling rate of $4f_s$. To obtain the sampling frequency required by the synchronizer, a resampler circuit is used. The resampler uses 16-bit accumulator and a Lagrange interpolator to produce a copy of the input signal at a lower sampling rate. Aliasing is avoided because the matched filters eliminated higher frequency components.
The accumulator is incremented by a fixed value $\kappa$ for every input sample received. A new output sample is desired when the value of the accumulator is zero. The increment is calculated as follows:

$$\kappa = \frac{2^{16} f_{out}}{f_{in}}$$

where $f_{in}$ is the input sampling rate and $f_{out}$ is the output sampling rate. When the accumulator wraps, the interpolator uses the last three samples received and the value of the accumulator ($A(t_n)$) to create the new sample. The interpolation is then performed over the sample set $\{t_{n-2}, t_{n-1}, t_n\}$. Assigning time indexes -1, 0 and 1 respectively to the samples gives the interpolation polynomials in Equations 4.24 through 4.27. For the resampler the desired sampling instant ($\tau$) is the distance between the sample at time $t_{n-1}$ and $A_{mx}$ normalized to $\kappa$. $\tau$ is then expressed as:

$$\tau = \frac{A_{mx} - A(t_{n-1})}{\kappa}$$

### 4.5.4 Timing Recovery Implementation

The implemented architecture of the symbol timing recovery module is shown in Figure 4.31. The spectral line generator and one of the IIR filters are shown in Figures 4.32 and 4.33 respectively. Internally the IIR filter uses six bits for decimal representation in addition to
14 bits for integer representation. By only truncating to 14 bits at the output of the filter, the truncation error is reduced.

Notice that only the imaginary part of the complex multiplication is required for the spectral generation circuit. This permits the optimization of the multiplication to only require two multiplications and one addition. Given that the inputs to the multiplier are four 14 bit numbers, the output could require up to 29 bits of precision. Empirical results, however, showed that only 25 bits are required when the input signal is fully scaled to $\pm 2^{13}$. To maintain a signal of 14 bits, the output of the multiplier is rescaled by shifting it right 11 bits.
The output of the third filter is then passed through a limiter circuit that saturates at $\pm 725$. The saturation value was empirically determined to eliminate the amplitude modulation component of the output signal.

In the PLL, shown in Figure 4.34, the phase difference between the signal generated by the spectral generator and a cosine generated by a local NCO is calculated. A multiplier is used to measure the phase error. A PI filter is used for the PLL’s loop filter. Due to the small size of the loop filters coefficients, they cannot be properly represented using a reasonable number of bits. Obtaining representable values requires that the normalization factor ($\mu$), that converts the phase error into an NCO accumulator offset, be moved into the filter. An additional factor of $2^3$ is used to upscale the filter values because multiplying by $\mu$ does not result in sufficiently large values. Truncating the three least significant bits of the output of the filter provides the increment correction to the NCO. The NCO’s increment and accumulator are then used to calculate $\gamma_a$, $\tau$ and the strobe signal. Although (4.23) dictates that $d$ should be normalize to $\lambda$, this can be approximated by normalizing it to $A_{mx}/4$ which is the ideal value of $\lambda$. This is easily done by adjusting the binary point of $d$. An extra bit is added to the $\tau$ signal to allow for values greater than one, when $A(t)$ is very close to $\gamma_a$ and $\lambda$ is less than $A_{mx}/4$. Table 4.2 show the some of the values used for the timing recovery module.
### Table 4.2: Symbol synchronizer implementation values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop filter proportional constant ($\alpha$)</td>
<td>$2.44 \times 10^{-6}$</td>
</tr>
<tr>
<td>Loop filter integrating constant ($\beta$)</td>
<td>$3.02 \times 10^{-8}$</td>
</tr>
<tr>
<td>Phase error size</td>
<td>14 bits</td>
</tr>
<tr>
<td>NCO input size</td>
<td>12 bits</td>
</tr>
<tr>
<td>NCO accumulator size</td>
<td>28 bits</td>
</tr>
<tr>
<td>NCO ramp output size</td>
<td>12 bits</td>
</tr>
<tr>
<td>$\mu$</td>
<td>$2^{28}/2\pi$</td>
</tr>
<tr>
<td>$\tau$ size</td>
<td>11 bits</td>
</tr>
</tbody>
</table>

#### 4.5.5 Derotator Implementation

The derotator is implemented in two blocks. The first is comprised of the static portion of the architecture that is not dependent upon the constellation. This block, shown in Figure 4.35, was created in System Generator. The complex mixer derotates the input signal using a predicted phase error and a sin/cos look up table. The phase of the input symbol is calculated using Xilinx’s CORDIC atan core. The output of the atan block, which is in the range of $[-\pi,\pi]$ is normalized to a power of 2 in order to facilitates further operations, such as wrapping. The phase output of the slicer is then used to calculate the phase error of the input symbol. The measured phase error is in the equivalent range of $[-2\pi,2\pi]$ because it is calculated using a subtraction. The error is re-normalized to $[-\pi,\pi]$ by adding $2\pi$ if the value is less than $-\pi$ or adding $-2\pi$ if the value is greater than $\pi$. Table 4.3 shows the implementation parameters for the derotator.

Substituting the values in Table 4.1 into Equation 4.28 it can be seen that $\theta_p(n+1)$ and
Figure 4.35: Derotator FPGA implementation

\[ \theta_p(n+1) = \theta_p(n) + f_p(n) \]  
\[ f_p(n+1) = f_p(n) - \frac{f_p(n)}{2^9} \]

In the update stage of the filter, \( \theta_p \) and \( f_p \) are updated to reflect the new measured values as follows:

\[ \theta_p(n) = K_{1,1}(\theta_e(n) - \theta_p(n)) + \theta_p(n) \]  
\[ f_p(n) = K_{2,1}(\theta_e(n) - \theta_p(n)) + f_p(n) \]

As mentioned in Section 4.2.8 the Kalman gains are static and pre-calculated at design time. Their values are shown in Table 4.3. Figure 4.36 shows the FPGA implementation of both the prediction and the update stage of the Kalman filter.

Because the slicer block is constellation unique, it is generated at implementation time using a custom C++ HDL generator. A constellation name and the average signal power are provided as input. A Matlab engine is opened by the generator and the constellation description is loaded. All constellations are stored in XML files and contain a list of the constellation points. The magnitude of the constellation points has been normalized to obtain an average power of one. A Matlab script is then used to determine the slicer architecture to be used. Constellation points are scaled by the average power to ensure proper slicing.
(a) Prediction stage

(b) Update stage

Figure 4.36: Kalman filter FPGA implementation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_p$ size</td>
<td>14 bits</td>
</tr>
<tr>
<td>sin/cos size</td>
<td>12 bits</td>
</tr>
<tr>
<td>ATAN processing elements</td>
<td>9</td>
</tr>
<tr>
<td>Phase error scaling</td>
<td>$(2^{13} - 1)/\pi$</td>
</tr>
<tr>
<td>Kalman filter register size</td>
<td>14 bits</td>
</tr>
<tr>
<td>Kalman gain $K_{1,1}$</td>
<td>0.1702</td>
</tr>
<tr>
<td>Kalman gain $K_{2,1}$</td>
<td>0.01582</td>
</tr>
</tbody>
</table>

Table 4.3: Derotator implementation parameters
Figure 4.37: Grid-based slicer architecture.

The grid-based architecture is shown in Figure 4.37. Values $V_1$ through $V_n$ are the decision boundaries. The input values are compared with the boundaries and the correct symbol is chosen using two $n+1:1$ muxes, where $n$ is the number of boundaries. This architecture is fairly efficient as it only requires $2^n$ comparators and four look-up tables. All comparisons run in parallel, which provides a slicing latency of one clock cycle.

A distance-based slicer works for any constellation but is computationally expensive. For every point in the constellation two subtractions, two multiplications and one addition must be done. Additionally, $\log_2(M)$ comparisons are needed to determine the shortest distance, where $M$ is the number of points in the constellation. Ideally every constellation point would be processed in parallel, but this would not take advantage of the pipelining built into the multiplier blocks. The resource requirements would also be excessive. For 64QAM, for example, 128 multipliers would be required. A simple solution is to multiplex access to the multipliers.

The RapidRadio distance-based slicer uses a distance block for every four constellation points. Each block has two look up tables containing the I and the Q values for four
Figure 4.38: Distance block.
symbols. Calculating the distance between the input and each of the symbols in the table is accomplished using the circuit in Figure 4.38. Shifting the output of the subtractor right one bit keeps the signal 14 bits wide. The sum of the squares is then calculated \((d_i)\). Each distance is compared to the previous minimum \(d_{\text{min}}\). If it is lower then the minimum distance is updated to the current value. This process is shown on the left side of Figure 4.38. The output of the distance block is the smallest distance and the location and phase of the corresponding symbol (the latter is not shown in Figure 4.38). Multipliers have a pipeline depth of four, and the addition and subtraction circuits have a latency of one cycle. With the the comparison and control logic, the distance block has a total latency of 10 cycles.

Slicing constellations with more than four symbols is accomplished by using more than one distance unit. For a given constellation a total of \(U = M/4\) distance blocks are required. A set of cascading comparison units is used to merge the output of the distance blocks to obtain the constellation point. Each comparison unit divides the \(L\) inputs into \(L/2\) pairs. For each pair a relational operator is used to eliminate the largest distance. A total of \(\log_2(U)\) comparison units are necessary to obtain the results because each comparison unit reduces the number of candidate symbols by a factor of two. Given that each comparison units has a latency of one clock cycle, the distance-based slicer then has a total latency of \(10 + \log_2(U)\). Figure 4.39 shows the overall architecture of the slicer.
Figure 4.39: Distance-based slicer architecture
Chapter 5

Results

This chapter discusses the performance of the RapidRadio framework prototype. The signal analysis subsystem is first discussed. Its ability to classify unknown signals is validated through simulations. The implementation of the synchronization architecture is examined with over-the-air test. The framework’s platform independence is then demonstrated. A system wide test is discussed to demonstrate the integration of the analysis and synthesis phases. Finally, a user survey is that examines the required user knowledge-base required for operating the framework.

5.1 Signal Classification

The framework’s capacity to accurately identify the modulation scheme of an unknown signal was validated via simulations in Matlab. Signals modulated with one of six different modulation schemes were generated at various Signal to Noise Ratio (SNR) levels. The framework was then used to determine the modulation scheme and its conclusion was compared to the real value. The energy per symbol to noise ratio ($E_sN_0$) was varied from 0dB to 20dB. The constellations tested were BPSK, QPSK, 8PSK, 8QAM, 16QAM and 32QAM. For each
constellation 200 sample signals were generated at each $E_sN_0$ level. Each generated signal had 16k samples. A channel with white Gaussian noise is assumed. A carrier frequency error of 5 kHz was also introduced.

Two metrics are used to evaluate the performance of the classification subsystem. The first, is the number of times a signal was correctly identified. This metric however can be misleading, because it does not indicate how many times a constellation was incorrectly identified (false positives). Counting the number of false positives provides a measure of the confidence level for a given classification. If no false positives for BPSK are seen, for example, then classification of BPSK has a higher confidence level than if many false positives are seen.

Figures 5.1(a) and 5.1(b) show the rate of correct classification and of false positives respectively, as a function of $E_sN_0$. Notice that all signals are properly classified for signal levels greater than 12 dB. The rate of false positives for 8QAM, however, peaks at over 80%, indicating that for low signal levels most signals will be classified as 8QAM. This is partly explained by the fact that the metrics used to classify the signal are extracted from a synchronized signal and the synchronizer is incapable of establishing synchronization at low signal levels. This, however, does not explain the dip in classification probability for 8QAM shown in Figure 5.1(a).

Further examination showed that when 8QAM was misclassified it was classified as 16QAM. To understand why this happens the average value of the metrics were plotted (Figure 5.2). The comparison shows that the metrics are very similar at low SNRs. The dominant metrics are symbol distribution and transition distribution. Initially 8QAM has a higher probability because the difference in the transition distribution is higher than for the symbol distribution. As the SNR increases, however, the trend reverses itself, and the symbol distribution becomes the dominant metric, causing 16QAM to have the higher probability. At around 8 dB, the trend reverses itself again and the signal is then correctly classified as 8QAM.

An increased false positive rate results in a decreased confidence in the classification process.
Figure 5.1: Performance of signal classification algorithm with no victory margin

(a) Rate of correct classification

(b) Rate of false positives
Figure 5.2: Comparison of metrics for hypothesis of 8QAM and 16QAM, given an actual constellation of 8QAM.
Figure 5.3: Performance of signal classification algorithm with a 5% margin of ambiguity
(a) Rate of correct classification

(b) Rate of false positives

Figure 5.4: Performance of signal classification algorithm with a 10% margin of ambiguity
As stated above, the 8QAM constellation has a high false positive rates for $E_sN_0 < 5$. This would lead users to ignore a classification of 8QAM for low signal levels. High false positive rates are a side effect of forcing the system to always choose a constellation, even when there is no clear winner. Allowing the system to indicate when it cannot discern between constellations (results are ambiguous) reduces the rate of false positives. The most likely constellation must then beat the second most likely constellation by a margin large enough to avoid ambiguity. The required margin is then called the margin of ambiguity. Figures 5.3(a) and 5.3(b) show the rate of correct classification and false positives when a margin of ambiguity of 5% is required. Notice that introducing the margin of ambiguity eliminates many of the false positives, but also eliminates some of the correct classifications requiring higher signal levels for correct classification. Increasing the required margin of ambiguity to 10% further reduces the rate of false positives. This can be seen in Figure 5.4(b). The rate of correct classifications, however, is reduced further requiring higher signal levels for correct classification. This is shown in Figure 5.4(a).

The performance of the classification algorithm was tested for the 16QAM modulation with over-the-air signals. The percentage of correct classification, using a 10% margin of ambiguity, is compared to simulation results in Figure 5.5.

From Figures 5.1(b), 5.3(b) and 5.4(b) it can be observed that false positives are only seen for a subset of the constellations. It can also be seen that the rate of false positives is a function of $E_sN_0$. Using this information a variable margin of victory can then be used, assigning values depending on the signal level and the winning constellation. The margin of victory for BPSK, for example, could be set to near zero since no false positives for BPSK are observed. Higher values can be assigned to constellations with higher rates such as 8QAM and 16QAM. Using an intelligent agent, the margins can be adjusted over time to reflect the selections made by the user at run-time. This also allows for the development of margins for new constellations for which no pre-existing rate of false positives exists.

In comparing the signal classification algorithm used by the RapidRadio framework to con-
Figure 5.5: Comparison of the performance of the signal classifier for 16QAM under simulation and over the air.
<table>
<thead>
<tr>
<th>Technique</th>
<th>SNR Required</th>
<th>Non-Linear Mod.</th>
<th>High-Order Mod.</th>
<th>Integrated Adjustable Prob.</th>
<th>Integrated Synchronization</th>
<th>New Mod. at Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclostationarity</td>
<td>-3 to 0 dB</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cumulants</td>
<td>-5 to 5 dB</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RapidRadio</td>
<td>3 to 7 dB</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison between the RapidRadio classification architecture and other popular methods.
temporary work two techniques were examined: cyclostationarity [42, 41] and cumulants [103, 104]. A high-level comparison between classification architectures is shown in Table 5.1. Although the RapidRadio classification architecture underperforms in terms of the required signal level for classification, it does exceed in other areas. First, RapidRadio does not assume perfect synchronization. Because establishing synchronization is done as part of the classification architecture, a perfectly synchronized signal is not required. Cyclostationary-based approaches require that the sampling frequency be an exact multiple of the actual symbol rate (not to be confused with the nominal symbol rate). This can only be accomplished once perfect synchronization has been achieved. Synchronization, however, is difficult to obtain if the modulation scheme is unknown and the signal is below the noise floor. Second, RapidRadio does not require training, or the calculating of thresholds to permit the classification of new constellations. This permits the addition of new modulations to be considered at runtime. Third, RapidRadio allows modifying the a priori probabilities of all the constellation because it uses a Bayesian network.

5.2 Synchronization Architecture

Although simulation provided encouraging results, system validation with signals acquired over the air was desired. Over the air signals provide a more comprehensive test of the synchronization architecture because of the presence of effects such as multipath and fading. Transmitters were created for QPSK, 8QAM, 16QAM and 32QAM. Each transmitter used a carrier frequency of 2.05 GHz, a roll-off factor (α) of 0.65, a root-raised cosine pulse shaping filter and a symbol rate of 250 kHz. A carrier plus noise to noise ratio of 36 db was used. Table 5.2 indicates how this translates to $E_bN_0$ for each of the tested constellations.

The receiver uses an intermediate frequency of 70 MHz in the analog circuitry. A 14-bit ADC with a sampling clock frequency of 8 MHz is used to digitize the signal. Bandpass sampling allows for the reduction of system requirements and generates a copy of the desired signal at
Table 5.2: $E_bN_0$ of received signal for tested constellations.

<table>
<thead>
<tr>
<th>Constellation</th>
<th>$E_bN_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32QAM</td>
<td>19.01</td>
</tr>
<tr>
<td>16QAM</td>
<td>19.98</td>
</tr>
<tr>
<td>8QAM</td>
<td>21.22</td>
</tr>
<tr>
<td>QPSK</td>
<td>22.99</td>
</tr>
</tbody>
</table>

2 MHz. A complex down-converter with matched filters is used to bring the signal down to baseband with a 5 kHz carrier frequency error (4% of symbol rate). A decimation factor of two in the down-converter reduces the sampling rate to 4 MHz. A resampler further reduces the sampling rate to $\frac{4}{T_s}$. The resampler uses an 28-bit counter and the Lagrange interpolator discussed in Section 4.2.7 to produce a copy of the input signal at a lower sampling rate, with a maximum output frequency of one half of the input frequency. Changing the output frequency of the resampler allows testing various symbol frequency errors.

Figure 5.6 shows the output of the synchronizer for the various constellations with a timing error of 2% (5 kHz) and a two percent carrier frequency error. 81,000 symbols were obtained after allowing time for system stabilization. Notice how the symbols are concentrated around the ideal symbol locations. The variance of the symbols can be partially attributed to the fact that $d$ is normalized to $2^{10}$ and not to $\lambda$ (see Section 4.5.4). This simplification starts to break down as the difference between $\lambda$ and $A_{mx}4$ increases, causing the symbol synchronizer to inject noise. Figure 5.7 shows the resulting symbols when the timing error is increased to 4% (10 kHz). It can be seen that although the error injected by the synchronizer increased, only 32QAM shows obvious signs of errors.

Resource utilization for the synchronization architecture is shown in Table 5.3. The difference in resources utilized can be attributed to the difference in slicers. DSP and RAM utilization is constant because they are used by the static portion of the architecture. The DSP blocks are used by the interpolators in the symbol synchronizer and by the complex multiplier in
Figure 5.6: Synchronization of over the air signals with 2% symbol frequency and 2% carrier frequency error

<table>
<thead>
<tr>
<th>Constellation</th>
<th>DSP48</th>
<th>RAMB16</th>
<th>Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>32QAM</td>
<td>9 (14%)</td>
<td>4 (3%)</td>
<td>6029 (23%)</td>
</tr>
<tr>
<td>16QAM</td>
<td>9 (14%)</td>
<td>4 (3%)</td>
<td>3066 (12%)</td>
</tr>
<tr>
<td>8QAM</td>
<td>9 (14%)</td>
<td>4 (3%)</td>
<td>3730 (14%)</td>
</tr>
<tr>
<td>QPSK</td>
<td>9 (14%)</td>
<td>4 (3%)</td>
<td>3032 (11%)</td>
</tr>
</tbody>
</table>

Table 5.3: Resource utilization of synchronization architecture
Figure 5.7: Synchronization of over the air signals with 4% symbol frequency and 2% carrier frequency error.
the derotator. RAM blocks are used for the sine/cosine look-up tables. The highest resource utilization can be observed for 32QAM and 8QAM which use a distance based slicer.

5.3 System Synthesis

A key aspect of the RapidRadio framework is the ability to automatically assemble systems based on pre-built modules. The framework should have the capacity to target multiple platforms, independent of vendor. This is accomplished in two ways. First, a common build interface, in the form of a Makefile, is used for all vendors. This allows the system to use the same commands to build a system regardless of what the target platform is. Second, platform information, such as the clock speed, module name and interface, is placed in an XML file. Using an XML permits the framework to avoid hardcoding information about given platforms. Support for new platforms can be easily added by adding a new XML file.

The synthesis mechanism was verified by targeting two distinct platforms; the Harris SDR SIP which has a Xilinx FPGA and the USRP which has an Altera FPGA. A QPSK modulator was successfully built for each of the platforms. The Modulator consisted of a scrambler and a differential encoder. The scrambler converts the incoming bit sequence into a pseudo random sequence. The differential encoder converts the serial bitstream into a sequence of symbols. Symbols are then mapped to constellation points based on the previous symbols. Although this sample design does not create new modules, it does show that no FPGA vendor specific assumptions are built into the framework.

5.4 Receiver Deployment

Following independent verification of the pieces of the RapidRadio framework, the functionality of the combined system was examined. Five signals were generated and transmitted
Figure 5.8: Test platform for over-the-air experiments.

over the air at 2.05 GHz. The framework was used to classify the signal, determine the modulation parameters and synthesize the radio. The modulation schemes used for the tests were QPSK, 8QAM, 16QAM, 32QAM and RECT. RECT is a constellation invented solely for the purpose of testing the systems ability to recognize and deploy receivers for non-traditional constellations. Both the transmitter and the receiver were implemented in the Harris SDR SIP package which contains four Xilinx Virtex 4 XC4VLX60 FPGAs. Separate systems and clock sources were used for the transmitter and receiver to ensure that synchronization is not achieved simply because the same clock is used for both systems. A block diagram of the test platform is show in Figure 5.8. Although the exact impulse response of the channel is unknown, uniform noise and flat fading were observed. No in-band interference was present because the band used is not available for use by commercial devices.

For each signal the constellation chosen by the framework was validated and the receiver deployment phase was started. The synthesis phase was executed on A Linux machine,
using the standard vendor tools. A TCP/IP daemon running on the ARM processor in the development board was then used to load the newly created receiver bitstream onto the FPGA connected to the receiver RF chain. Figure 5.9 shows the extracted symbols for the test constellations. It can be observed that for all constellations synchronization was properly recovered because the extracted symbols are clustered appropriately.

Table 5.4 shows the modulation parameters for all test signals. The first row shows the nominal values used by the transmitter. The second row contains the the values obtained by the parameter estimator. These values are used by the classification algorithm to attempt synchronization. The last row in each section shows the results of the parameter refinement process. These are the values actually used to deploy the receiver. It can be observed that after parameter refinement the modulation parameters are very similar to the nominal values.

Table 5.5 shows the resource utilization for the deployed systems as well as the instantiation parameters for the matched filters. As expected the signals with the lowest data rate require high-order filters because of the high number of samples per symbol at the input sampling rate. The highest system utilization is observed by the constellations that require a distance based slicer (32QAM, 8QAM and RECT).

5.5 User Base Survey

The RapidRadio framework seeks to reduce the knowledge-base required for the classification of an unknown signal and the deployment of a receiver capable of receiving such a signal. To gauge the framework’s success in this endeavor, a group of 20 graduate students were asked to use the framework and fill out a survey (see Appendix B). Each student was presented with identical scenarios in which two signals were present in the spectrum. Using the framework, the student had to identify the signal of interest and classify it. After classification the framework deployed the transmitter for the selected signal and the user was shown a set of
Figure 5.9: Sample symbols extracted from the deployed receivers. Clustering of symbols indicates synchronization was properly achieved.
<table>
<thead>
<tr>
<th>Modulation</th>
<th>$E_sN_0$ (db)</th>
<th>Value Type</th>
<th>$\alpha$</th>
<th>$f_c$ (Hz)</th>
<th>$f_s$ (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32QAM</td>
<td>9.18</td>
<td>Nominal</td>
<td>0.65</td>
<td>2000000</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Estimated</td>
<td>0.54</td>
<td>1999311</td>
<td>104628</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refined</td>
<td>0.54</td>
<td>1999314</td>
<td>100010</td>
</tr>
<tr>
<td>16QAM</td>
<td>7.36</td>
<td>Nominal</td>
<td>0.65</td>
<td>2000000</td>
<td>250000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Estimated</td>
<td>0.64</td>
<td>1999162</td>
<td>251045</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refined</td>
<td>0.64</td>
<td>1999130</td>
<td>250039</td>
</tr>
<tr>
<td>8QAM</td>
<td>8.50</td>
<td>Nominal</td>
<td>0.75</td>
<td>2000000</td>
<td>350000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Estimated</td>
<td>0.74</td>
<td>2000384</td>
<td>351056</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refined</td>
<td>0.74</td>
<td>1999806</td>
<td>350046</td>
</tr>
<tr>
<td>QPSK</td>
<td>12.56</td>
<td>Nominal</td>
<td>0.65</td>
<td>2000000</td>
<td>250000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Estimated</td>
<td>0.68</td>
<td>1999308</td>
<td>246683</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refined</td>
<td>0.68</td>
<td>1999211</td>
<td>250044</td>
</tr>
<tr>
<td>RECT</td>
<td>10.75</td>
<td>Nominal</td>
<td>0.70</td>
<td>2000000</td>
<td>500000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Estimated</td>
<td>0.67</td>
<td>2003341</td>
<td>506648</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refined</td>
<td>0.67</td>
<td>2000914</td>
<td>500087</td>
</tr>
</tbody>
</table>

Table 5.4: Modulation parameters obtained from RapidRadio framework for test over-the-air signals. The first row shows the nominal values used by the transmitter. The second row shows the values estimated by the spectral fitting. The third row shows the values used for receiver deployment.
Table 5.5: Resource utilization for deployed receivers on a Xilinx Virtex 4 LX60 FPGA, using a sampling rate of 8MHz and an operating clock rate of 192MHz.

<table>
<thead>
<tr>
<th>Constellation</th>
<th>RRCOS Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Order</td>
</tr>
<tr>
<td>32QAM</td>
<td>608</td>
</tr>
<tr>
<td>16QAM</td>
<td>248</td>
</tr>
<tr>
<td>8QAM</td>
<td>176</td>
</tr>
<tr>
<td>QPSK</td>
<td>256</td>
</tr>
<tr>
<td>RECT</td>
<td>120</td>
</tr>
</tbody>
</table>

extracted symbols. The survey was used throughout the process to help determine the user’s knowledge in the area of communications and experience in the design and implementation of radios. Questions about the users interaction with the framework served to determine if the framework worked properly.

In total there were twenty participant in the survey with varying levels of communications knowledge. One of the participants, however, was not able to complete the survey due to his complete lack of knowledge in the area of communications. This person had received no training in communications and was unfamiliar with the concept of a Fourier transform. Although this person was not able to complete survey, his attempt demonstrated that a minimum of communications knowledge is required to operate the framework. The remainder of those surveyed had an undergraduate degree in Electrical and Computer Engineering and had taken at least a basic communications class as part of their undergraduate degree. Unless otherwise stated, the discussion that follows does not take into account the one person that was not able to complete the survey.
5.5.1 User Background and Experience

The first part of the survey was used to establish the users knowledge of basic concepts required to design or implement a radio receiver. Users were asked simple questions to measure their understanding of:

- What the power spectrum represents and how to identify the presence of a signal in a plot of the power spectrum.
- What the $I/Q$ plane represent.
- What a constellations is in the context of the $I/Q$ plane.
- What the concept of synchronization is in the context of a radio receiver and the difference between symbol synchronization and carrier recovery.

When shown a sample power spectrum plot, all of those surveyed correctly identified the presence of two signals. This indicates an understanding of what the Fourier transform is and how the presence of a signal is represented in a power spectrum plot. When asked if they knew what a constellation was 8 of those surveyed answered they did not know. When shown a constellation in the $I/Q$ plane and asked to identify how many symbols the constellation had, however, only three persons answered incorrectly. From looking at the constellation, nine of those surveyed were able to properly identify the modulation technique used.

To test the student’s ability to recognize when a signal has been properly synchronized, they were shown two sets of received symbols shown in Figure 5.10. The first set of symbols, seen in Figure 5.10(a), represent a 16QAM signal in which symbol timing has been properly recovered, but the carrier frequency has not. The second set of symbols shown in Figure 5.10(b) shows the same signal in which both symbol timing and carrier frequency have been recovered. The students were asked to identify which if any of the sets of symbols represent a signal with proper symbol timing recovery and only four persons identified both
sets as correct. The students were then asked to identify the set or sets in which frequency error had been corrected and twelve students correctly identified the second set.

Of those surveyed nine had previously designed or implemented a radio receiver, but only four had ever actually implemented a part of the radio on an FPGA. In most cases only downconversion and filtering were done on the FPGA, with only one person doing carrier recovery on the FPGA. In average less than less then 40% of the radio had been implemented on the FPGA. The time spent on FPGA design, however, was often greater than half of the receiver design time. Additionally six persons indicated that they would have wanted to implement more of the radio on the FPGA, but did not do it due to lack of knowledge or length of design time.

The results of these first set of questions show that approximately half of those surveyed have only the most basic communications knowledge. They were capable of identifying a signal in the spectrum, but did not clearly understand the difference between symbol timing recovery and carrier frequency recovery. Additionally most had never implemented a radio.
5.5.2 Framework Evaluation and Radio Deployment

The second set part of the survey provided a brief overview of the RapidRadio framework and guided the user through the process of classifying an unknown signal. For this part of the survey the users interacted with the framework and had to use it to deploy a receiver. The spectrum was populated with two signals. An 8QAM signal at 2 MHz and a QPSK signal at 1 MHz. All those surveyed were capable of correctly classifying a signal and deploying a functional receiver.

The results of the survey indicate that only 11 of those surveyed actually understood what the metrics were measuring, but a verbal debriefing after the survey showed that even those that did not understand the metrics could validate the results of the framework by comparing the empirical PDFs to the theoretical PDFs. Lastly most of those surveyed indicated that to reproduce the process of classification and deployment without the use of RapidRadio would take considerably longer.

Overall, the survey showed that only basic communications knowledge and little to no FPGA knowledge is required to operate the framework successfully. Out of the twenty persons surveyed all but one were capable of correctly classifying an unknown signal and deploying an FPGA-based receiver. The survey also showed that although the metrics were presented in a manner easy to understand, moderate to advanced communications knowledge is required to fully benefit from the data presented.
Chapter 6

Conclusions

With the emergence of new radio platforms that can change over time, the traditional process of radio design may no longer be suitable. It is possible, as is the case in signal intelligence, that the user may not know the parameters of the signal he or she may want to receive. Additionally, such parameters may be of little to no interest to the user, as long a functional receiver is produced. A new process is required that focuses on automating the process of discovering the type of receiver required and generates a functional receiver. In this work the RapidRadio framework for signal classification and receiver deployment is presented. RapidRadio is an analysis-based framework that allows the users to focus on the process of signal classification, abstracting away the underlying design and implementation of the receiver.

A new signal classification algorithm is presented that has comparable performance to those presented in contemporary literature. The proposed algorithm obtains an estimate of the modulation parameters from the spectral shape of the signal. Using the estimated parameters the framework constructs a synchronization architecture for each hypothesized constellation. The signal is then synchronized under each hypothesis and evaluated for correctness using four metric combined with a Bayesian network. A receiver is then created for the most likely
hypothesis.

The presented classification algorithm was shown to properly classify a wide range of signals under varying signal-to-noise ratios. It was also shown that it can be easily expanded to work with new constellations because it avoids artificial intelligence frameworks that require training. The algorithm is applicable to a wider range of signals, when compared to other in the contemporary literature, because it avoids assumptions such as perfect synchronization. The proposed algorithm also allows the user to change the \textit{a priori} probability of any constellation. By changing the base probabilities of the hypothesis, the user can inject knowledge of the environment into the classification algorithm.

Synchronization is a vital part of any receiver, yet it is often modulation specific. To allow maximum flexibility the RapidRadio framework uses a quasi-generic synchronization architecture. This architecture can be easily tailored to fit any linear modulation scheme. Using an XML description of the desired constellation the framework determines the proper modifications to the synchronizer and creates a custom hardware module. The architecture is also designed to work with a wide margin of error, to compensate for the fact that only estimates of the modulation parameters are available.

The hardware synthesis portion of the framework used to deploy receivers was validated using over-the-air signals. Receivers for five different signals were deployed with the RapidRadio framework. Initial test show that transmitted symbols could be properly received with little to no errors. Resource utilization was also shown to be moderately low. Lastly, receiver deployment was shown to be platform independent.

\subsection{Future Work}

The RapidRadio framework combined the process of signal classification and FPGA-based receiver deployment in an innovative manner. Further optimizations, however, can increase
the flexibility of the framework. The author suggests the following expansions to the framework:

- **Modulation support**: Expand the signal classification algorithm to work with other modulation schemes. Adding more signal models would allow the parameter estimator to identify the modulation parameters of more signal types. Increasing the number of synchronization architectures used would also expand the classification capabilities of the framework.

- **Receiver synthesis**: Identifying the common elements in the system architecture of the receiver can lead to shorter synthesis times. Partial reconfiguration can be used to build only that which is different for the receiver, considerably reducing build times. Receiver synthesis should also be expanded to address platforms with other configurable devices.

- **Expanded validation**: Although initial testing validated the functionality of the framework, an extended validation should be accomplished. The effects of channel fading, multi-path and co-channel interference on the classification algorithm should be investigated. All of these sources of noise can affect the spectral shape of the signal, thereby reducing the effectiveness of the parameter estimator. Additionally, synchronization may be affected, reducing the frameworks capable of correct classification.

- **Cognition**: Expanding on the CLIPS back-end to allow the framework to learn from previous experience. *A priori* constellation probabilities can be automatically updated to reflect incorrect classifications. Location awareness can allow the system to modify its behavior based on its location. Information about the spectrum being observed can also indicate the expected constellation types. Lastly, a signal selection algorithm can be used to automatically choose the signal to be classified.
Bibliography


[15] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, “Multi-processor system-level synthesis for multiple applications on platform FPGA,” Field Pro-


Appendix A

Sample XML Files
A.1 Radio Description File

```xml
<?xml version="1.0" encoding="utf-8"?>
<Receiver>
  <Parameter Name="SamplingFrequency" Value="8000000.00"/>
  <Module Name="IsoFilt" Subtype="Bandpass" Type="Filter">
    <Parameter Name="Center" Value="1000000.00"/>
    <Parameter Name="BW" Value="200000.00"/>
    <Parameter Name="TB" Value="8875.00"/>
    <Parameter Name="Order" Value="128.0"/>
  </Module>
  <Module Name="Mixer" Subtype="Complex" Type="Mixer">
    <Parameter Name="CarrierFrequency" Value="1000000.00"/>
  </Module>
  <Module Name="I_BasebandFilter" Subtype="RRCOS" Type="Filter">
    <Parameter Name="SymbolRate" Value="100000.00"/>
    <Parameter Name="Decimation" Value="20.00"/>
    <Parameter Name="Alpha" Value="0.75"/>
    <Parameter Name="Order" Value="128.0"/>
  </Module>
  <Module Name="Q_BasebandFilter" Subtype="RRCOS" Type="Filter">
    <Parameter Name="SymbolRate" Value="100000.00"/>
    <Parameter Name="Decimation" Value="20.00"/>
    <Parameter Name="Alpha" Value="0.75"/>
    <Parameter Name="Order" Value="128.0"/>
  </Module>
  <Connection><Source Interface="RecvIn" Module="Global"/></Source>
  <Sink Interface="dataIn" Module="IsoFilt"/></Connection>
  <Connection><Source Interface="dataOut" Module="IsoFilt"/></Source>
  <Sink Interface="dataIn" Module="Mixer"/></Connection>
  <Connection><Source Interface="I_Out" Module="Mixer"/></Source>
  <Sink Interface="dataIn" Module="I_BasebandFilter"/></Connection>
  <Connection><Source Interface="Q_Out" Module="Mixer"/></Source>
  <Sink Interface="dataIn" Module="Q_BasebandFilter"/></Connection>
  <Connection><Source Interface="dataOut" Module="I_BasebandFilter"/></Source>
  <Sink Interface="RecvOutI" Module="Global"/></Connection>
  <Connection><Source Interface="dataOut" Module="Q_BasebandFilter"/></Source>
  <Sink Interface="RecvOutQ" Module="Global"/></Connection>
</Receiver>
```
A.2 Hardware Description File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Platform>
  <Chip Vendor="Xilinx">
    <Device>xc4vlx60</Device>
    <Package>ff1148</Package>
    <Family>virtex4</Family>
    <Speedgrade>10</Speedgrade>
  </Chip>
  <Clock Freq="192000000">clk</Clock>
  <Reset Active="high">rst</Reset>
  <Name>receiver</Name>
  <Makefile>/home/jasuris/research/diss/code/HDL/base2</Makefile>
  <Interface Name="RecvIn" Type="XilSimple" Direction="input">
    <Port Name="DIN" Width="14" Type="data" Direction="input"/>
    <Port Name="ND" Width="1" Type="newData" Direction="input"/>
  </Interface>
  <Interface Name="RecvOutI" Type="DoubleFromFifo" Direction="output">
    <Port Name="I_data" Width="14" Type="data" Direction="output"/>
    <Port Name="Rdy" Width="1" Type="newData" Direction="output"/>
    <Port Name="Error" Width="1" Type="err" Direction="output"/>
  </Interface>
  <Interface Name="RecvOutQ" Type="DoubleFromFifo" Direction="output">
    <Port Name="Q_data" Width="14" Type="data" Direction="output"/>
    <Port Name="Qrdy" Width="1" Type="newData" Direction="output"/>
  </Interface>
</Platform>
```

A.3 Module Description File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Module Type="Filter" Subtype = "Bandpass" HDLName="firISO">
  <Chip Vendor="Xilinx">
    <Device>ANY</Device>
  </Chip>
  <Clock>CLK</Clock>
  <Reset Active="high">RESET</Reset>
  <Parameter Name="Center" Type="configurable"/>
  <Parameter Name="BW" Type="configurable"/>
  <Parameter Name="TB" Type="configurable"/>
  <Parameter Name="Order" Type="configurable"/>
  <Makefile>/home/jasuris/research/diss/code/HDL/modules/firISO</Makefile>
  <Interface Name="dataIn" Type="XilinxSimple" Direction="input">
    <Port Name="DIN" Width="14" Type="data" Direction="input"/>
    <Port Name="ND" Width="1" Type="newData" Direction="input"/>
  </Interface>
  <Interface Name="dataOut" Type="XilinxSimple" Direction="output">
    <Port Name="DOUT" Width="14" Type="data" Direction="output"/>
    <Port Name="RDY" Width="1" Type="newData" Direction="output"/>
  </Interface>
</Module>
```
Appendix B

User Survey
Communication Knowledge & Experience

1) Given the spectrum shown above, how many signals would you say there are?
- 0
- 1
- 2
- 3
- 4
- 5

2) Do you know what a constellation is?
- Yes
- No

The graphic below shows a set of received symbols. The x axis represents the magnitude of the in-phase component of the symbol and the y axis represents the magnitude of the quadrature component of the symbol.

3) How many symbols does the constellation displayed above have?
- 2
- 4
- 8
- 16

4) What kind of modulation is used for the constellation above
- Phase modulation
- Amplitude modulation
- Phase and Amplitude modulation
- Do not know

5) Of the symbols sets displayed below, for which ones would you say that symbol timing has been successfully recovered?
6) For the set of symbols displayed on the previous questions. For which set would you say the carrier frequency error has been corrected?
- a only
- b only
- a & b

7) Have you ever designed a radio receiver before? If so which parts
- No
- Downconversion
- Matched filtering
- Carrier recovery
- Symbol Synchronization
- Symbol Slicer
- RF front end
- ADC interface

8) Have you ever implemented a radio receiver before? If so which parts
- No
- Downconversion
- Matched filtering
- Carrier recovery
- Symbol Synchronization
- Symbol Slicer
- RF front end
- ADC interface
9) If you answered yes above, how long did it take? (including design, implementation and verification)
   - 1-2 weeks
   - 1-2 months
   - 3-4 months
   - > 4 months

10) Have you ever implemented part of a receiver on an FPGA?
   - Yes
   - No

11) Which parts if any did you implement on the FPGA?
   - Downconversion
   - Matched filtering
   - Carrier recovery
   - Symbol Synchronization
   - Symbol Slicer
   - RF front end
   - ADC interface

12) What percentage of the receiver was implemented on the FPGA
   - 0-20%
   - 20-40%
   - 40-60%
   - 60-80%
   - > 80%

13) What percentage of development time was spent on FPGA implementation
   - < 20%
   - 20-40%
   - 40-60%
   - 60-80%
   - > 80%

14) Would you have liked to implement more on the FPGA?
   - Yes
   - No

15) If yes? Then what stopped you?
   - FPGA was too small
   - Lack of FPGA knowledge
   - FPGA design is to lengthy

**RapidRadio Framework Evaluation**

Switch to the RapidRadio framework and the GUI should look as follows:

![RapidRadio GUI](image-url)

This GUI has 5 different areas:
1) The acquisition box, that allows the user to obtain a new sample of the spectrum.
2) A graphical representation of the FFT of the spectrum obtained from the receiver platform.
3) The signal selection box. This box is used to select the signal to be analyzed. Two sliders are used to
   zoom in to a specific signal. The center slider adjusts the center of the graph and the width slider
   adjusts the amount of spectrum displayed. After the spectrum has been limited to only contain the
   signal desired the user can press the "Isolate Signal" button. Pushing this button will then obtain a
   baseband version of the signal and perform parameter estimation.
4) Shows the spectrum of the signal at baseband and the fitted spectral shape.
5) Shows the estimated modulation parameters of the signal. The parameters displayed are the
   estimated carrier frequency error, the symbol rate, the roll-off factor (alpha) and the carrier plus noise
   to noise ratio. If the user is satisfied with the signal, pressing the "Next" button will then initiate the
   modulation classification process. If the results are not satisfactory, the process can be re-started by
   pressing the "Acquire signal" button again.
6) Press the "Acquire Signal" button. How many signals can you see in the RapidRadio framework?

   Pick a signal to analyze. Move the "Center" slider until the signal you want is in the center of the left
   graph. The use the width slider to reduce the area displayed, until the desired signal occupies about
   80% of the graph as so:

   Then press the "Isolate Signal" button.

17) What was the approximate center frequency of the signal you chose (from the leftmost
   graph of the first GUI)

18) How closely does the fitted curve match the periodogram of the signal
   - Not at all
   - A little
   - Similar
   - Very similar

   If the fitting was adequate press the "Next" button. If it was not, then press repeat the entire process
   ensuring that the complete signal was included before pressing the "Isolate Signal" button.

The GUI to display the results of the analysis is shown below:
This marked areas of the GUI are:

1) A list of all the tested constellations and their score. A higher score indicates a higher probability of being the correct constellation. Constellations are arranged in the order of likelihood. Choosing a constellation in this panel will cause all the metrics for that constellation to be displayed. By default the constellation with the highest probability is selected.

2) The phase profile. Displays the phase difference between symbols. The actual distribution for the rotating symbols is represented as the bars of the histogram. The ideal distribution, adjusted for the current SNR is displayed as the red line.

3) The amplitude profile. Displays the distribution of the magnitude (amplitude) of the symbols. The amplitude distribution of the symbols received is shown as a histogram. The red line shows what the distribution should be given the actual SNR.

4) The probability density function of the distribution of the symbols in the I/Q plane.

5) Radio buttons toggle the display in 4 from the empirical PDF (obtained from the received symbols) and the theoretical PDF obtained from the current SNR. The display can also be toggled from a 3 dimensional mesh to a 2 dimensional contour diagram.

6) The transition distribution. This graph plots the probability of a transition between any two points, based on the obtained data.

7) Button to generate the radio receiver.

**System Generation**

Select the constellation you think is correct and press the "Generate Current" button on the lower right hand side of the GUI. A new GUI should appear that looks like this:
The panel on the left hand side of the GUI gives the status of the build process. Please be patient the build will take about 10 minutes.

After the build has completed, if a return code of zero is indicated (rc=0) on the build panel, a set of received symbols will be displayed in the graph on the right hand side of the GUI. These are symbols received with the newly generated receiver, based on the classification process.

19) What constellation did you choose?

20) Did you choose the constellation suggested by the framework?
   - Yes
   - No

21) If you choose a constellation other than what was suggested by the framework, please explain why.

22) If an error was encountered, please note it below and consult with a RapidRadio representative.

23) Would you say that symbol timing was properly recovered for the data obtained from the newly generated system?
   - Yes
   - No

24) Do you see any clusters? If so, how many? If not, then what do you see?

25) Would you say that carrier frequency error was properly corrected for the data obtained from the newly generated system?
   - Yes
   - No

26) How well do you think the system constructed by the RapidRadio framework is working?
   - Not at all
   - Poorly
   - Satisfactorily
   - Well
   - Excellent

27) When classifying an unknown signal, how much do you think the RapidRadio framework helps?
   - Not at all
28) When implementing radios on the FPGA for an unknown signal, how much do you think the RapidRadio framework helps?
- Not at all
- A little
- Somewhat
- Considerably

29) Did you understand the metrics?
- Yes
- No

29a) How useful were the metrics displayed?
- Not at all
- A little
- Somewhat
- Considerably

29b) How, if at all, did the metrics affect your decision? 

30) How long did the process of signal classification and receiver deployment take you? 

32) If you were going to classify a signal without RapidRadio, what unit best describes the time frame? (Assume you have no a-priori knowledge of the signal)
- Minutes
- Hours
- Days
- Months
- Would not be able to 

33) If you were going to create a FPGA-based radio for the classified signal without RapidRadio, what unit best describes the time frame?
- Minutes
- Hours
- Days
- Months
- Would not be able to 

34) What would you change to make the framework more useful?

35) Do you think the framework correctly identified the signal?
- Yes
- No