RF FRONT END FOR SOFTWARE-DEFINED RADIO

By
Matt Vlasaty

Senior Thesis in Electrical Engineering
University of Illinois at Urbana-Champaign
Advisor: Prof. Jose Schutt-Aine

May 2021
Abstract

Software-defined radio (SDR) is the application of software-based signal processing to wireless communication systems. Integral to most SDR applications is the RF front end, which performs amplification and frequency conversion. The RF front end bridges the gap between baseband signals and carrier frequencies suitable for propagation. A printed circuit board design for an RF front end has been modified and fabricated for use in an SDR communication system. The design extends the operating frequency range of an existing SDR by converting to an intermediate frequency. Phase-shift keying modulation and orthogonal frequency-division multiplexing have been implemented in GNU Radio software and applied to the system. The design of the RF front end and its coplanar waveguide traces, and the system performance for several test cases, are presented.

Subject Keywords: software-defined radio; SDR; wireless communication; coplanar waveguide; GNU Radio; phase-shift keying; orthogonal frequency-division multiplexing
Acknowledgments

I would like to express my gratitude to Prof. Schutt-Aine for his guidance and for helping me to realize my vision. Through his suggestions and consistent attention, I was able to solve problems and tackle unfamiliar concepts. His interest in this work amplified my own. My understanding of and passion for wireless communication have grown immensely with him as both my professor and advisor.

I have endless appreciation for Thong Nguyen and his assistance. Throughout this project, he has taken every opportunity to help me find a direction that would make progress and that I felt would have benefit. This allowed me to pursue my goals with confidence, even during the times I stumbled. With Thong as both my teaching assistant and mentor, I have learned how to experiment without limit.

No matter what I set my mind to, I know that I will have the support of my family. They have always encouraged me to do my best, and I am glad that I can share this experience with them.

I am exceedingly lucky for the opportunity to dedicate myself to a topic that has intrigued me for so long, and I could not have asked for better circumstances.
Contents

1. Introduction to SDR ........................................................................................................................................... 1

2. System Overview ............................................................................................................................................... 2

   2.1 Resources .................................................................................................................................................. 3

3. Hardware Implementation ............................................................................................................................... 4

   3.1 Electrosense Design ................................................................................................................................ 4

      3.1.1 Components ...................................................................................................................................... 5

      3.1.2 Layout and Stackup .......................................................................................................................... 6

   3.2 Coplanar Waveguide Traces ....................................................................................................................... 8

      3.2.1 Test Board Measurement ............................................................................................................... 9

      3.2.2 Test Board Simulation .................................................................................................................. 10

      3.2.3 Via Placement Simulations ......................................................................................................... 11

      3.2.4 Trace Dimension Simulations ...................................................................................................... 14

      3.2.5 Final Design ................................................................................................................................... 18

   3.3 Electrosense Firmware .............................................................................................................................. 19

      3.3.1 Firmware Changes ........................................................................................................................... 20

      3.3.2 Firmware Installation ..................................................................................................................... 21

      3.3.3 Electrosense Usage ......................................................................................................................... 25

4. Software Implementation .................................................................................................................................... 28

   4.1 GNU Radio Flowgraphs ............................................................................................................................. 28

   4.2 PSK Implementation .................................................................................................................................. 30

      4.2.1 PSK Theory ....................................................................................................................................... 31

      4.2.2 PSK Flowgraph ................................................................................................................................ 32

      4.2.3 PSK Results .................................................................................................................................... 36

   4.3 OFDM Implementation ............................................................................................................................... 40

5. Conclusion ......................................................................................................................................................... 44

References ............................................................................................................................................................... 45
1. Introduction to SDR
Software-defined radio has emerged as a practical and versatile platform for wireless communication. By performing a majority of signal processing through software, SDR systems can be reconfigured based on the application. Although sampling rate and bandwidth are constrained by hardware, any number of digital techniques can be applied in an attempt to improve reliability or increase overall throughput. Before this processing can take place, an analog-to-digital converter (ADC) must sample the received signal. In most applications, an RF front end will both amplify the signal and convert it to baseband, which relaxes the requirements for ADC resolution and sampling rate. The RF front end and a digital-to-analog converter (DAC) can enable an SDR to transmit. It is also common for SDR systems to have programmable hardware, such as variable gain amplifiers that are software controlled.

In recent years, developments in both hardware and software have made SDR more affordable and accessible than ever before. The barrier to entry is no longer a large investment, as commercial SDRs are available for prices as low as 20 USD. Most of these inexpensive products follow the RTL-SDR design, which leverages a TV tuner to perform the functions of the RF front end. Although this design can only receive, SDRs that can transmit are available in the range of 100 USD. The HackRF One supports half-duplex operation up to 6 GHz, with a maximum sampling rate of 20 million samples per second (sps). HackRF is one example of several open-source projects, targeted for amateur use and education.

The availability of SDR hardware has encouraged the development of enabling software, used to monitor the radio spectrum and demodulate nearby broadcasts. While many of these programs are designed for ease of use, there are also frameworks that provide lower-level functionality. The set of signal processing tools packaged into GNU Radio form the basis of a complete communication system. In the graphical interface, known as GNU Radio Companion, these tools can be instantiated as blocks and connected together. Each block streams data to the next block in the signal chain, which begins at a data source and is terminated by a data sink. The interface is compatible with the mentioned hardware and operates in real-time.

The remainder of the thesis is organized as follows. Chapter 2 provides an overview of the implemented SDR communication system and communication techniques. Chapter 3 describes the hardware design process, and Chapter 4 describes the software capabilities. The RF front end successfully allows the system to communicate at carrier frequencies beyond the range of the receiving SDR.
2. System Overview

The goal of this research is to investigate the design of SDR systems and to realize their capabilities. For this purpose, an SDR communication system has been implemented, with a HackRF One functioning as a transmitter and an RTL-SDR functioning as a receiver. Both devices are operated using GNU Radio software, developed for this project. In addition, an external RF front end printed circuit board (PCB) is connected in the receiver chain to extend the frequency range of the RTL-SDR. The PCB design is the open-source Electrosense Frequency Extension Board, with a modified layout. Figure 2.1 shows a block diagram of the system, and Figure 2.2 shows photographs of the transmitter and receiver.

The transmitter PC streams data samples to the HackRF One over a USB connection. The device synthesizes baseband analog signals from these samples, at a rate of 2 Msps. Two data streams are transmitted in quadrature at the chosen carrier frequency. After propagating through the wireless channel, the signal is received by the antenna connected to the Electrosense board. There are three available antenna inputs, which each serve a unique frequency band. The receiver PC chooses which band is used. When the upper band is chosen, a signal with carrier frequency up to 6 GHz can be downconverted into the operating range of the RTL-SDR. This SDR performs another downconversion and performs sampling at the same rate as the transmitter. The samples streamed to the receiver PC contain the same information as those generated by the transmitter. In essence, the system performs digital communication, and the SDR hardware enables communication to occur over a wireless channel.

The choice of how to represent data and the task of recovering it after transmission are the responsibilities of the transmitter and receiver software. Two transmission methods have been studied and tested thoroughly: phase-shift keying (PSK) and orthogonal frequency-division multiplexing (OFDM).

![Figure 2.1: Block diagram of SDR system](image)
Using PSK, the data stream modulates only the phase of the transmitted RF signal. PSK has been implemented to transmit data as a continuous stream. Using OFDM, the data stream is parallelized, and each parallel stream is transmitted over a unique, orthogonal carrier frequency. The streams themselves use PSK modulation. OFDM has been implemented in a packet-based format. Each packet begins with a preamble for packet detection and a header containing metadata such as the packet length. The remainder of the packet contains the payload data. Each method requires synchronization techniques to acquire the optimal sample timing and to remove any frequency or phase offset. In some situations, it is necessary to correct for channel effects and hardware impairments. In the case of packet-based communication, the preamble and header are used to estimate timing offsets and channel response.

2.1 Resources
The design for the Electrosense Frequency Extension Board can be found at the Electrosense hardware GitHub repository [1]. The modifications that have been made starting from the V2 design are detailed in Chapter 3, and the design files are available at the GitHub repository for this project [2]. Many of the provided examples and tutorials for GNU Radio [3] were referred to during the development of the GNU Radio software that Chapter 4 describes.
3. Hardware Implementation

The HackRF One and RTL-SDR hardware were purchased for use in the system. The Electrosense Frequency Extension Board has been modified from its open-source design, fabricated, and assembled for use in the system. The reasons for these modifications and the process undertaken to arrive at a final design are described in this chapter.

3.1 Electrosense Design

The purpose of the Electrosense board is to extend the operating frequency range of the RTL-SDR receiver. The nominal range for the RTL-SDR is from 24 MHz to 1.766 GHz. The Electrosense board supports frequencies from DC up to 6 GHz, with each of its three inputs covering some portion of this range. The output is connected to one of the three input paths by an RF switch. The receiver PC chooses between the inputs. The “LOW” input is intended for frequencies up to 30 MHz. This path will upconvert signals using a fixed local oscillator (LO) frequency of 72 MHz. The “MID” input serves as a bypass path for signals that are already within the RTL-SDR range. This path only amplifies the signal and does not apply frequency conversion. The “HIGH” input serves frequencies above the RTL-SDR range and up to 6 GHz, and it is of the most interest for this reason. This path will downconvert signals to a fixed intermediate frequency (IF) of 1.576 GHz, using a programmable phased-locked loop (PLL) to generate the LO signal. Figure 3.1 shows a block diagram of the Electrosense board and downconversion path. Although only the RF paths are shown, many of the components are controlled by the board CPU.

![Figure 3.1: Block diagram of Electrosense board](image)
Although the open-source design is complete, it was not feasible to fabricate. The PCB design uses four copper layers, and the open-source stackup is not identical to the stackup available from the prospective manufacturer. In addition, several components on the board were not available from suppliers and had to be substituted. Most parts were replaced without modification to their PCB footprint.

The critical difference between the stackup from the original design and the stackup available from the manufacturer is the dielectric thickness between the top layer and the first internal layer. Since all of the RF traces lie on the top layer, the first internal plane is the ground plane reference for these traces. Since the dielectric thickness has changed, the trace width must also change to maintain a 50 Ω impedance. A test board was first fabricated to evaluate the manufacturing process and the ability to achieve the desired impedance. However, this board did not perform as expected. Consequently, the test board and several variations were simulated using Ansys HFSS. Cut-out sections of the Electrosense board were also simulated to verify the design before fabrication. Section 3.2 discusses these simulations and their results.

3.1.1 Components
The Electrosense CPU is an STM32 microcontroller. The receiver PC establishes serial communication with the board CPU over a USB connection. The board CPU firmware opens a shell that is visible to a serial terminal on the receiver PC, and this shell is used to send commands to the board. Section 3.3 describes the process of flashing the firmware and accessing the shell. The CPU sends control signals to the output RF switch to select an input path. It also provides the fixed 72 MHz LO signal for upconversion and controls the PLL chip used for downconversion.

The MAX2870 PLL acts as a frequency synthesizer, providing the LO signal for the downconversion path. It can operate in either integer-N mode or fractional-N mode to cover an output frequency range from 23.5 MHz to 6 GHz with high resolution. A 38.4 MHz temperature compensated crystal oscillator provides the PLL reference. When the CPU receives a request to use the downconversion path, the frequency must be specified so that the PLL can be tuned. The firmware tunes the PLL to a variable LO frequency, ensuring that the input signal will always be converted to a fixed 1.576 GHz IF. The status of the PLL can be read by the CPU, including an indicator for when it has locked to the desired frequency.

The output of the downconversion mixer stage is passed through a surface acoustic wave (SAW) band-pass filter, with its response centered at the 1.576 GHz IF and with a 2 MHz bandwidth. Out-of-band components are attenuated strongly by this filter. Depending on the input frequency, the CPU selects
between a low-pass filter and a high-pass filter to eliminate image frequencies at the input to the mixer. There are two fixed-gain amplifiers in the downconversion chain: an RF amplifier at the input and an IF amplifier before the output switch. The RF amplifier boosts the received signal level by about 20 dB, and the IF amplifier compensates for the mixer conversion loss and other passive losses.

The bypass path includes a single fixed-gain amplifier before connecting to the output switch. The upconversion path includes a mixing stage with RF, IF, and LO filters. The mixer can be disabled when the path is not in use.

Several components were replaced by alternates from the original design due to limited availability. Both the CPU clock and the PLL reference oscillator were replaced by parts with similar output characteristics and identical PCB footprint. The RF amplifier in the downconversion path was replaced by an equivalent part from the same manufacturer, and several inductors used for amplifier biasing were also replaced. The through-hole SMA connectors from the original design were replaced by surface-mount SMA connectors. This change eliminates the stubs created by the through-hole connectors.

3.1.2 Layout and Stackup
The area of the board is 70 mm by 100 mm (2.76 in by 3.94 in). This is slightly larger than the original design, to accommodate a through connection at the bottom of the board. Every component uses a surface-mount package and is mounted on top of the board, with the exception of the through-hole programming header for the CPU. The routing has been altered, but the component placement is unchanged. There are three SMA connectors on the left side of the board for each of the input paths, and one SMA connector on the right side of the board for the output. The USB connector is also on the left side of the board. Figure 3.2 shows both a graphic of the board layout, from Altium Designer, and a photograph of the physical board, which was fabricated and assembled by PCBWay.

The board has 4 copper layers and an overall thickness of 1.6 mm or 63 mils. The top and bottom external layers are used for signal traces, the top internal layer is a ground plane, and the bottom internal layer is a 3.3 V power plane. Figure 3.3 shows the detailed stackup of the board. Signal traces on both external layers are also surrounded by a ground fill. The ground fills provide a direct connection for ground pins. The vias used to connect each ground layer and to carry power and signal connections are drilled and plated through the entire board.
The ground fill has consequences on the structure and propagation characteristics of the signal traces. There is a ground plane underneath the trace, coplanar ground conductors on either side of the trace, and ground vias that connect the two layers. This structure is known as grounded coplanar waveguide (GCPW). The GCPW traces used in the RF signal paths must be designed for an impedance of 50 Ω. Figure 3.4 shows the final dimensions used for the 50 Ω traces. Section 3.2 describes the process of designing these traces, using both the measurement of a physical test board and the simulation of several variations of this test board.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Material</th>
<th>Type</th>
<th>Weight</th>
<th>Thickness</th>
<th>Dk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Top Overlay</td>
<td>Overlay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Top Solder</td>
<td>Solder</td>
<td>Solder Mask</td>
<td>0.01016mm</td>
<td></td>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
<td>Top Layer</td>
<td>Signal</td>
<td>Toz</td>
<td>0.03556mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Dielectric1</td>
<td>FR-4</td>
<td>Core</td>
<td>0.11mm</td>
<td>4.20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>InternalGND</td>
<td>Signal</td>
<td>Toz</td>
<td>0.036mm</td>
<td>3.96</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Dielectric5</td>
<td>FR-4</td>
<td>Propag</td>
<td>1.2mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>InternalV3</td>
<td>Signal</td>
<td>Toz</td>
<td>0.036mm</td>
<td>3.96</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Dielectric6</td>
<td>FR-4</td>
<td>Core</td>
<td>0.11mm</td>
<td>4.29</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Bottom Layer</td>
<td>Signal</td>
<td>Toz</td>
<td>0.03556mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Bottom Solder</td>
<td>Solder</td>
<td>Solder Mask</td>
<td>0.01016mm</td>
<td></td>
<td>3.5</td>
</tr>
<tr>
<td>11</td>
<td>Bottom Overlay</td>
<td>Overlay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.3: Electrosense board stackup
3.2 Coplanar Waveguide Traces
In order to verify the design of the CPW traces before the Electrosense board was fabricated, a test board was designed and manufactured. This board has both a direct single-ended line designed for 50 Ω impedance and a meandering differential line designed for 100 Ω differential impedance. The differential line is of interest due to the differential output of the MAX2870 PLL. However, this output is the only differential line on the Electrosense board, and it is relatively short. The single-ended line was examined in much more detail. Figure 3.5 shows the test board layout. The area is 63.5 mm by 31.8 mm (2.5 in by 1.25 in). The test board uses the same edge-mount SMA connectors as the Electrosense board.

The critical difference from the Electrosense board is that the test board has only 2 copper layers. The overall thickness remains 1.6 mm, meaning that the dielectric thickness between the signal trace and its return path is over 10 times larger. Figure 3.6 shows the exact stackup. Although the test board does not have the same stackup, it is useful to verify the tool that has been used to calculate trace dimensions for a specified impedance.
The results for several CPW impedance calculation tools were compared, and ultimately the dimensions given by the calculator within Altium Designer were used. Figure 3.7 summarizes these dimensions. The traces and connector footprints create several ground islands on the top layer, which have been connected to the bottom ground plane by single vias. There is also a via fence between the single-ended line and the differential line and between the single-ended line and the edge of the board. The via fences were placed to reduce crosstalk between the two lines.

3.2.1 Test Board Measurement
The S-parameters of the test board were measured from 9 kHz to 8.5 GHz using the Copper Mountain Technologies S5085 2-Port Vector Network Analyzer. None of the 4 ports for the differential line were terminated during the measurement. Figure 3.8 plots the magnitude of $S_{11}$ and $S_{21}$ in dB for this measurement. The response deviates significantly from what would be expected for a 50 Ω transmission line. There is a distinct dip in transmission just above 3 GHz, and the loss is quite large at higher frequencies. This line is practically unusable above about 2 GHz.
3.2.2 Test Board Simulation

The test board was simulated to prove that the layout itself is responsible for the unexpected behavior of the line. The layout that was created in Altium Designer and sent to the manufacturer was converted into an ODB++ file format that can be imported by Ansys HFSS to create a 3D Layout Design. All of the routing, via and component placement, and stackup information is preserved by this format, ensuring that the simulation setup is consistent with the original design files. Any discrepancies between the
measurement and simulation are caused by differences between the physical board and its intended design, error in the measurement, or inaccuracies in the simulation parameters. For example, the simulation is configured to use a constant value of 4.3 over all frequencies for the FR-4 dielectric constant. In reality, this value will deviate due to tolerance, dispersion, and dielectric loss.

Lumped ports are placed at the center pins of the two connectors. There are no terminations placed at the differential line connectors. The solution frequency is set to 9 GHz, and an interpolating frequency sweep from DC to 9 GHz is used.

Figure 3.9 shows the comparison between measurement and simulation. The simulation is clearly in agreement in predicting resonance around 3 GHz. The simulation predicts a higher resonant frequency, likely due to the discrepancy in dielectric constant. It seems that energy is leaking away from the line at resonance, so it is useful to plot the field distribution. Simulating the design at 2 GHz and at 3.4 GHz reveals how the field distribution changes near resonance. Figure 3.10 shows the magnitude of the electric field on the top surface of the dielectric at these frequencies. The scale units are dBV/m.

At 2 GHz, energy escapes at the connectors, where there are no ground vias. However, the coupled fields are 30 dB less than the fields underneath the single-ended line. At 3.4 GHz, the coupled fields have the same magnitude as those underneath the line. In addition, the fields are strongest at the boundary, but not directly underneath, the differential line.

3.2.3 Via Placement Simulations

With the measurement and simulation in agreement, it is now possible to simulate the design after performing modifications. The goal of modifying the design is to determine which parameters of the via
fence can be varied to eliminate resonance. The first set of simulations is used for a qualitative assessment of which parameter is most important.

Removing parts of the via fence and replacing them with single vias does not eliminate resonance. This result illustrates an important point: the observed resonance is caused by the ground planes. The potential on the top plane is only equal to the potential on the bottom plane where the two have been connected by a via. In areas where there is a lack of via stitching, the top and bottom patches form a waveguide-like structure that can resonate and draw energy away from the main trace. Removing the via fence only exacerbates this problem. Figure 3.11 and Figure 3.12 demonstrate two cases: the first with the top via fence removed and the second with the bottom via fence removed.

Figure 3.10: Distribution of electric field magnitude at 2 GHz (a) and 3.4 GHz (b)

Figure 3.11: Test board with top via fence removed
From the field distribution in Figure 3.10, it seems reasonable to place more vias around the connectors. These vias should reduce coupling into the ground patches and improve the response of the line. Figure 3.13 shows the simulation result for this change. The resonance around 3 GHz is not purely suppressed. Instead, the resonant frequency has increased, which is still an improvement compared to the original design.

Several other via parameters were investigated, such as via diameter, spacing between the via fence and the line, and spacing between vias. None of the initial changes made to these parameters were found to have any significant effect. However, an improvement was seen when several of these changes were
made in conjunction. The via diameter was reduced to 0.3 mm (12 mils), which is the smallest size that can be drilled by the manufacturer. This allows the via-to-via spacing to be reduced to 0.89 mm (35 mils). Spacing between the via fence and the line is also reduced. Figure 3.14 shows the result for these modifications. Resonance now appears above 5 GHz.

These initial simulations support the idea that the via fence is most effective when it is constructed like a wall. With many small vias packed closely together, the fence is nearly continuous. This observation seems to be in agreement with other published results [4].

### 3.2.4 Trace Dimension Simulations
In order to understand the effect of trace dimensions, a second set of simulations is performed. These simulations analyze what is essentially a cut-out of the single-ended line on the test board. This cut-out removes both the differential line and the connectors from the design, so that only the line, the vias, and the ground planes remain. Several modifications of the cut-out are simulated. One modification of the cut-out has the same trace dimensions as the Electrosense board. The CPW traces on the Electrosense board are much thinner because the dielectric thickness is significantly reduced compared to the test board. Figure 3.15 shows the geometry used for this set of simulations.

For every case, the parameters listed in Table 3.1 remain the same. Table 3.2 lists the parameters that are allowed to vary for the six cases. Case 1 represents the original dimensions of the test board, designed for 50 Ω impedance. Case 2 reduces the dielectric thickness H. Case 3 increases the spacing S between the trace and coplanar ground conductors. Case 4 increases via-to-line spacing VL and
increases spacing S even further. Case 5 uses the trace dimensions from the Electrosense board, which were calculated for the manufacturer stackup using Altium Designer. Case 6 increases the dielectric thickness H from Case 5.

Table 3.1: GCPW Fixed Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (mm)</th>
<th>Value (mils)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>40.0</td>
<td>1574.8</td>
</tr>
<tr>
<td>TW</td>
<td>12.0</td>
<td>472.4</td>
</tr>
<tr>
<td>VC</td>
<td>1.0</td>
<td>39.4</td>
</tr>
<tr>
<td>VR</td>
<td>0.5</td>
<td>19.7</td>
</tr>
<tr>
<td>T</td>
<td>0.0175</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 3.2: GCPW Parameters for Cases 1-6

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>H (mm)</th>
<th>W (mm)</th>
<th>S (mm)</th>
<th>VL (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Test board</td>
<td>1.508</td>
<td>1.234</td>
<td>0.254</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>Smaller H</td>
<td>0.254</td>
<td>1.234</td>
<td>0.254</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>Larger S</td>
<td>1.508</td>
<td>1.234</td>
<td>0.883</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>Larger S and larger VL</td>
<td>1.508</td>
<td>1.234</td>
<td>1.883</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>Electrosense</td>
<td>0.110</td>
<td>0.203</td>
<td>0.254</td>
<td>2.0</td>
</tr>
<tr>
<td>6</td>
<td>Larger H</td>
<td>0.762</td>
<td>0.203</td>
<td>0.254</td>
<td>2.0</td>
</tr>
</tbody>
</table>
Figure 3.16 shows the comparison between Case 1 and Case 2. Although the line in Case 2 is no longer a 50 Ω transmission line, the reduced dielectric thickness seems to improve the response at higher frequencies. The effect can best be seen on the Smith chart plot of S₁₁. The response of Case 2 closely resembles a transmission line with characteristic impedance less than the reference impedance. However, the response of Case 1 shows a strong inductive behavior, despite the line being designed for 50 Ω. In other words, the behavior of Case 2 is expected, while the behavior of Case 1 is not.

Figure 3.17 shows the comparison between Case 1 and Case 5. The line in the Case 5 design behaves like a 50 Ω transmission line, as expected.

One key difference between Case 1 and Case 2 and between Case 1 and Case 5 is the ratio of H to S. In GCPW, both the coplanar ground conductors and the lower ground plane provide references for the signal. When S is much smaller than H, most of the field couples to the coplanar ground conductors. This can be seen in Figure 3.10, where the field strength is much higher under the slots than it is under the trace itself. In Case 2 and Case 5, H is equal to or less than S. The purpose of Case 3 and Case 4 is to observe if the response of the test board can be made more predictable by increasing the slot spacing S, so that S is on the same order as H.
Figure 3.17: Comparison between Case 1 and Case 5

Figure 3.18 shows the comparison for Case 1, Case 3, and Case 4. Increasing $S$ causes the line impedance to increase slightly. The response also improves slightly because the dips in $S_{21}$ are not as significant for the increased slot spacing.

Figure 3.19 shows the comparison between Case 5 and Case 6. By increasing the dielectric thickness, the impedance of the line in Case 6 is now larger than 50 $\Omega$. However, the response is still similar to what would be expected.
The results for the second set of simulations indicate that the thin dielectric is much more suitable for GCPW than the thick dielectric. For the thin dielectric, fields are tightly coupled to the lower ground plane, and via height is reduced. Evidently, this effect makes the transmission line more reliable for any structure of ground vias surrounding it. The Electrosense board uses the dimensions from Case 5, which has the best performance.

Figure 3.20 shows the simulation result for a cut-out taken from the Electrosense board. This cut-out has the longest continuous CPW trace on the board. The trace connects two pads on the board. A lumped port is placed at each pad. There are several ground vias, but this placement is not the exact placement in the final design. The cut-out does exhibit some minor resonance. However, it is not nearly as severe as on the original test board. The length of the cut-out trace is about the same as the length of the trace in Case 5, and their responses are similar.

3.2.5 Final Design
Figure 3.2 shows the final layout for the Electrosense board. The dimensions from Case 5 are used for all of the traces that require 50 Ω impedance. There are no via fences directly surrounding any of the traces, but there is via stitching throughout the board. The combined results from Figure 3.17 and Figure 3.20 show that the effect of via placement is not nearly as significant as the effect of dielectric thickness. It is the reduced dielectric thickness of the four-layer board stackup that allows the Electrosense board to have expected performance.
The results presented in Section 3.2.3 and Section 3.2.4 were sufficient to finalize the Electrosense trace design. However, more simulations of the Electrosense board itself would have been valuable if time had permitted. This analysis would also be more complete with further comparisons between via fences and via stitching. Ultimately, via stitching was chosen because it was used in the original design and because the results did not suggest that via fences were necessary. The four-layer stackup proved to be much more suitable for GCPW, and results for the two-layer stackup were difficult to improve upon.

The results also suggest that implementing GCPW is far more precarious than implementing a structure like microstrip. Calculations exist to find the dimensions for a 50 Ω GCPW trace, given a desired slot spacing S. However, whether that trace will perform like a 50 Ω transmission line at high frequencies is a more complicated question. Further experimentation could be used to develop general procedures, but these procedures may not account for all aspects of a design in the way that a simulation does.

The bottom trace on the board, that is used as a through path, was originally placed so that it could be measured to evaluate the amount of crosstalk from the rest of the board. The trace was chosen to be stripline before the GCPW traces were evaluated. If the stripline trace had been replaced by GCPW, the measurement of this trace would have provided verification of the simulations and design process.

3.3 Electrosense Firmware

The firmware for the Electrosense board is available in the hardware repository [1] that also contains the layout files. Both source code and pre-built binaries are available. However, at the time of this
project, the firmware had not been updated for the newest version of the board. The changes that have been made to the firmware and the process for flashing it onto the CPU are described in this section.

The Electrosense firmware allows the receiver PC to tune the hardware to a desired carrier frequency. For example, to downconvert a signal with a carrier frequency of 2.4 GHz, the CPU must toggle the output switch to use the downconversion path. It must also tune the MAX2870 PLL to the LO frequency that will shift the signal to an intermediate frequency of 1.576 GHz.

The CPU runs a real-time operating system known as FreeRTOS. A shell processes user requests received over the serial port. The receiver PC sends these requests using a serial terminal client. Requests must belong to the list of valid commands. Section 3.3.3 explains and gives examples of the commands that will tune the board.

**3.3.1 Firmware Changes**
Two changes must be made to the firmware for this version of the board. The first change is to correct the GPIO pin assignments. The second change is made to ensure that the PLL receives the intended reference signal and is able to lock.

A previous version of the board had several extra switches and parts that were removed from this design. The STM32 required an I/O expander to control these additional components. Several pin assignments were changed with the removal of the I/O expander.

The STM32 supplies a 72 MHz clock signal over its MCO pin that is used as the LO for the upconversion path. This pin is also connected to the reference input of the MAX2870 PLL through resistor R11. Figure 3.21 shows the schematic sheet for the PLL. In case the crystal oscillator X2 is not working, the MCO pin can act as the reference signal. However, the MCO pin should not provide an output when the crystal oscillator is the intended reference. The firmware originally did not disable the MCO output, which prevented the PLL from locking over its entire frequency range. The MCO pin mode was reassigned so that it acts as an output during upconversion and as an input for any other case. This ensures that the pin is set to a high-impedance state when the PLL requires the crystal oscillator reference signal.

After making these changes, the firmware was rebuilt using GCC for Arm. The updated source code, a changelog, and updated binaries are available at [2].
3.3.2 Firmware Installation

The required software for firmware installation includes STM32 ST-LINK Utility [5] and dfu-util [6]. There are alternatives to ST-LINK Utility, such as STM32CubeProgrammer. The required hardware includes an STM32 programmer, such as the ST-LINK V2, and a Micro-USB cable.

When the firmware is built, two binary files are created. Each file must be loaded into the flash memory of the STM32. The first binary file is the bootloader, named “esense_bootloader.bin”. The second binary file is the firmware itself, named “esense.bin”. When both binaries have been loaded into the correct memory locations, the bootloader will run on startup and then pass program execution to the firmware.

The first step is to flash the bootloader, which requires use of the ST-LINK V2 hardware and ST-LINK Utility software. After it has been flashed, the bootloader will set the STM32 into Device Firmware Upgrade (DFU) mode, which allows the firmware to be flashed over the USB connection using dfu-util. As long as the bootloader is never manually erased, the STM32 can always be reset into DFU mode. The firmware can be upgraded or modified without ever having to reflash the bootloader.
Figure 3.22 shows the connection between the ST-LINK V2 and the Electrosense board. A ribbon cable connects the 5V, GND, SWDIO, and SWCLK pins on both the programmer and the board header. The connections on the ST-LINK V2 are labeled. Figure 3.23 shows the pinout of the board header. Connect the ST-LINK V2 to a USB port, open ST-LINK Utility on the PC, and select “Target > Connect”. The terminal output at the bottom of the window should display the message: “Connected via SWD”. Then, select “Target > Program and Verify”, and open the “esense_bootloader.bin” file. Figure 3.24 shows the menu that opens after this step. Ensure that the start address is set to 0x08000000 and select “Start”.

Figure 3.22: Electrosense board connected to ST-LINK V2 programmer

Figure 3.23: Programming header pinout
The terminal output should indicate that the bootloader has been successfully loaded into flash memory. The bootloader should force the red LED on the Electrosense board, labeled USB, to blink on and off. At this point, the ST-LINK V2 can be disconnected from both the PC and the Electrosense board.

The next step is to load the firmware using dfu-util, which is a command line utility. Connect a Micro-USB cable to the Electrosense board and to the PC. Open a terminal and change directory to locate the “esense.bin” file. Enter the command “dfu-util -l” to see a list of DFU-compatible devices. Figure 3.25 shows the list when the Electrosense board is connected. Find the entry named “TELEMIC E-Sense Upload to Flash 0x08002000”. This entry has the correct memory location for the firmware. Take note of the parameter named “alt”, which should have a value of 2. Assuming that this value is 2, enter the command “dfu-util -a 2 -D esense.bin -R”. Figure 3.26 shows the resulting terminal output. The USB connection to the Electrosense board resets because of the “-R” flag. Without disconnecting the USB cable, the board will reconnect to the PC. For the first few seconds, the red LED on the board will flash, indicating that the bootloader is active. Once the red LED remains on, the firmware is active. This bootloader process repeats whenever the board is reconnected. Firmware installation is now complete.
Figure 3.25: List of DFU-compatible devices

Figure 3.26: Terminal output for firmware installation
3.3.3 Electrosense Usage

With its firmware installed, the Electrosense board will process commands sent by the receiver PC. These commands are sent to the board CPU, over the USB connection, through a serial terminal on the PC. There are many ways to open a serial terminal for any operating system. In the following example, the Python serial module is used to open a serial terminal named “miniterm”. The process of connecting to the board also varies for the operating system and serial terminal being used. On Windows, the board is identified by its COM port. On Linux or MacOS, the board is identified by its device path. In the following example using Linux, the device path is “/dev/ttyACM0”. The baud rate used for serial communication is 115200, which is defined by the firmware.

In this example, the Python miniterm serial terminal connects to the board using the command “python -m serial.tools.miniterm /dev/ttyACM0 115200”. Upon connecting to the board, the Electrosense shell should become visible. Enter the command “help” to see a list of valid commands. Figure 3.27 shows this list. Entering a command such as “convert”, which requires additional arguments, will cause the shell to print these arguments. The “reboot loaderp” command will reset the board to DFU mode, requiring the firmware to be reflashed. The “syslog” command will print a list of the most recent events logged by the firmware. Figure 3.27 also shows the “syslog” output after startup, which reveals that PLL locking has failed. This is expected because the MCO pin mode is initially set to output.

To verify that the PLL can lock, enter the command: “max vcocache calibrate”. This will rerun the startup PLL calibration procedure, with the MCO pin mode set to input. Then, enter the command: “max vcocache”, which prints the VCO index for each frequency in the list. Figure 3.28 shows the VCO cache list before and after running the calibration command. The list entries are given a VCO index after calibration. Calibrating the VCO cache is optional. After calibration, the firmware will manually select an internal VCO when tuning to a new frequency, rather than doing so automatically. Manual selection saves the time that it takes for the PLL to choose which VCO can be tuned to the desired frequency.

![Image of Electrosense shell commands]

Figure 3.27: Electrosense shell commands
The “convert setup” command is used to tune the board, and the “convert status” command is used to view how the board is currently tuned. The status command prints a list of the available bands and the current input frequency and output frequency. The firmware assigns an ID to each band, and the asterisk indicates which band is in use. The “selected antenna” can be ignored. These are the firmware band definitions, which are related to the hardware in the following way. The band named “SW” corresponds to the upconversion path, which uses the input labeled “LOW” on the board. The bands beginning with “SHF” correspond to the downconversion path, which uses the input labeled “HIGH” on the board. The band named “Bypass” corresponds to the bypass path, which uses the input labeled “MID” on the board. Entering the status command is the best way to identify which input path is in use. For example, if the downconversion path is in use, the status command will print the asterisk next to one of the “SHF” bands. The CPU controls the output RF switch so that the signal from this path propagates to the output connector.

The syntax for the setup command is “convert setup [freqkHz] [forceBand]”. The “freqkHz” parameter is the input frequency in kHz that the board will be tuned for, and it is required. The “forceBand” parameter can optionally be set to one of the band IDs from the list printed by the status command. This is useful when the desired frequency is covered by more than one band. Upon entering the command, the board responds with the intermediate frequency that will appear at the output and the ID of the band that will be used. The response also indicates whether the conversion has inverted the spectrum of the signal, which occurs when the LO frequency is greater than the input frequency. This inversion can be undone in software in one of several ways, such as conjugating the time-domain samples.

Figure 3.29 shows an example where the status is printed before and after converting to an input frequency of 2.4 GHz. In this case, the Electrosense board performs downconversion. The “max status” command confirms that the PLL is locked and prints more status information than shown. In addition, the green LED on the board should be on to indicate that the PLL is locked. The yellow LED should be blinking on and off since the downconversion path is in use.
Figure 3.29: Frequency conversion command usage

<table>
<thead>
<tr>
<th>Band</th>
<th>ID</th>
<th>Name</th>
<th>FreqMin [kHz]</th>
<th>FreqMax [kHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SW</td>
<td>0</td>
<td>27000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SHF-L</td>
<td>1500000</td>
<td>3500000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SHF-M</td>
<td>3500000</td>
<td>6500000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SHF-H</td>
<td>6500000</td>
<td>13576000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SHF-Hc</td>
<td>18424000</td>
<td>13576000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Bypass</td>
<td>0</td>
<td>20000000</td>
<td></td>
</tr>
</tbody>
</table>

Input Frequency: 1500000 kHz
Output Frequency: 2400000 kHz
Spectral inversion: 0
Selected antenna: 0

<table>
<thead>
<tr>
<th>Band</th>
<th>ID</th>
<th>Name</th>
<th>FreqMin [kHz]</th>
<th>FreqMax [kHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SW</td>
<td>0</td>
<td>27000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SHF-L</td>
<td>1500000</td>
<td>3500000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SHF-M</td>
<td>3500000</td>
<td>6500000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SHF-H</td>
<td>6500000</td>
<td>13576000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SHF-Hc</td>
<td>18424000</td>
<td>13576000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Bypass</td>
<td>0</td>
<td>20000000</td>
<td></td>
</tr>
</tbody>
</table>

Input Frequency: 2400000 kHz
Output Frequency: 1576000 kHz
Spectral inversion: 1
Selected antenna: 0

PLL Locked
Divided VCO Frequency: 3976000 kHz
4. Software Implementation
To explore the capabilities of the SDR system, PSK and OFDM have been implemented in GNU Radio. In the case of PSK, data is streamed continuously from transmitter to receiver, and a known data sequence is transmitted for verification. The OFDM software transmits data in the form of packets, which contain training symbols, used for packet detection and synchronization, and metadata that identifies the length of a packet. For verification, the payload data for each packet contains a sequential index. An index not being received is an indication that a packet has been dropped. Synchronization of PSK symbols has been implemented using blind recovery algorithms such as the Costas loop, while OFDM synchronization uses data-aided methods. Although the PSK software is less practical, it is useful to determine how the quality of data received depends on different system parameters. The OFDM software is more practical because transmitting data in packets is more conducive to sending messages or files. However, it can be difficult to detect packets when data is impaired.

Chapter 4 describes the basic theory behind these communication techniques and how they have been implemented in GNU Radio. Section 4.1 describes the structure of flowgraphs created using the GNU Radio Companion tool. Section 4.2 provides both an explanation of the PSK flowgraph and the results obtained under various test conditions. Section 4.3 describes the OFDM flowgraph and the challenges faced in recovering packets without error.

4.1 GNU Radio Flowgraphs
Version 3.7.13.4 of GNU Radio Companion (GRC) is used for this project. Although there are newer versions available, this version has all of the necessary blocks. Most of the blocks used are native to GRC itself. In order to interface with the hardware, the gr-osmosdr library [7] has been added to the environment. This library adds SDR-compatible blocks to GRC, and it also requires that the proper device drivers have been installed.

GNU Radio Companion is used to create flowgraphs of interconnected blocks. Data flow begins at a source block and terminates at a sink block. The blocks that are connected between the source and sink have both data inputs and data outputs. After it has been created, a flowgraph is executed to perform its work. There is also the option to generate an equivalent Python script for a given flowgraph. The GNU Radio toolkit is written in Python, and GRC is used to instantiate the tools. Many blocks have bindings to make use of performance-driven code written in C++.
Figure 4.1 shows an example of a simple flowgraph, which is used to view the spectrum of data received by the RTL-SDR. With an antenna connected, the SDR amplifies, downconverts, and samples the received signal. The SDR transfers data to the PC over its USB connection. From the perspective of the flowgraph, the RTL-SDR produces samples and is instantiated as a source block. There are several settings within the block used to configure the hardware. In this example, the carrier frequency is set to 101.1 MHz, and the sampling rate is set to 2 Msps. The RF gain and IF gain settings can be increased to further amplify the signal before sampling. Data from the SDR source block is produced at the sampling rate and processed by the frequency sink block. This block performs a Fast Fourier Transform (FFT) on a window of 1024 samples and plots the resulting spectrum. Samples are stored in a buffer, and the plot is refreshed once the buffer is full. The bandwidth parameter, which is equal to the sampling rate, is used by the plot labels. The color of the source output is light blue to indicate that the block produces complex floating-point values. The real and imaginary part of these values are the I and Q data samples that range from -1 to 1.

Figure 4.2 shows the GUI window that opens upon executing the example flowgraph. The value of the FFT is plotted in dB scale, and the frequency axis extends over the 2 MHz bandwidth. This is a visualization of the data after it has been downconverted, so a signal at the center of the plot was received with a carrier frequency of 101.1 MHz. The two signals visible in the spectrum are local FM radio broadcasts. The power received from the station broadcasting at 101.1 MHz is greater than the power received from the station broadcasting at 100.3 MHz.
Any flowgraph created in GRC will have a similar structure. Each block operates on the samples within its input buffers and produces samples to fill its output buffers. The GNU Radio scheduler is responsible for managing these buffers and directing blocks to perform their work. The input and output buffers can have different sizes, which allows for blocks that alter the sample rate. Therefore, connections within the same flowgraph may not have the same sample rate. The stream tag is another important feature that enables the formation of packets. A stream tag is attached to a sample within a data stream to identify the start and length of a packet. Blocks that operate on packets require the use of stream tags.

To implement a transmitter, an SDR sink block must be instantiated. The device transmits the samples that appear at the input of its corresponding sink block. Within the transmitter flowgraph, there must also be a source block that produces data. This data can be random, follow a pre-defined sequence, or be taken from a file. There are also sink blocks to write to a file or to display data in a time-series format.

4.2 PSK Implementation
The flowgraph that implements phase-shift keying (PSK) modulation contains both the transmitter and receiver. Known data is transmitted to verify the operation of the receiver. Received data has been visualized for various test conditions to evaluate system performance. These conditions include transmit
power, antenna separation, receiver gain, and other hardware configurations. To test the frequency conversion performed by the Electrosense board, the transmitter and receiver can use different carrier frequencies.

4.2.1 PSK Theory
The data to be transmitted can be of any format or type but ultimately consists of bits. Modulation determines which properties of the transmitted analog signal represent these bits. In general, any combination of amplitude, frequency, or phase variation can encode digital data. The combinations of these properties are discrete, and they are referred to as symbols. Each symbol generally encodes a bit sequence of a common length. When only amplitude and phase are modulated, the set of all possible symbols can be represented by a constellation diagram.

Figure 4.3 shows the constellation diagram for PSK with 4 possible symbols and PSK with 8 possible symbols. Since only phase is being modulated and amplitude remains unchanged, the symbols form a circle in the IQ plane. Each symbol is a complex number that represents a unique bit sequence.

Equations (4.1) and (4.2) are the baseband analog I and Q signals, expressed in terms of the magnitude and angle of the symbol being transmitted. Equation (4.3) is the transmitted signal, with carrier frequency $\omega_c$, which has been quadrature modulated by the I and Q signals. Equation (4.4) applies a trigonometric identity to show that the transmitted signal is modulated with the same amplitude and phase as the symbol. Therefore, the constellation diagram is sufficient to expresses how each bit sequence modulates the carrier.

![Figure 4.3: PSK constellation diagrams](image-url)
\[ i(t) = A(t) \cos \theta(t) \] \hspace{1cm} (4.1)

\[ q(t) = A(t) \sin \theta(t) \] \hspace{1cm} (4.2)

\[ s(t) = i(t) \cos(\omega_c t) - q(t) \sin(\omega_c t) \] \hspace{1cm} (4.3)

\[ s(t) = A(t)[\cos \theta(t) \cos(\omega_c t) - \sin \theta(t) \sin(\omega_c t)] = A(t) \cos[\omega_c t + \theta(t)] \] \hspace{1cm} (4.4)

Symbols often consist of multiple samples. For example, if a symbol consists of four samples, the I and Q data streams produce the same symbol for four consecutive time instants. In this case, the symbol rate is equal to one quarter of the sampling rate. From the perspective of a flowgraph, this can be viewed as interpolation. Symbols are produced at the symbol rate and then interpolated by a factor equal to the number of samples per symbol. Therefore, bandwidth is reduced from the Nyquist frequency by the same factor. To further control bandwidth occupied by a signal, pulse shaping is applied. The same block that performs interpolation can also implement a finite impulse response (FIR) filter that changes the shape of the signal spectrum. The raised-cosine filter is a preferable choice for pulse shaping because the excess bandwidth can be directly controlled. The FIR implementation of a raised-cosine filter is defined by the excess bandwidth parameter and the number of taps. The other desirable property of the raised-cosine filter is that its impulse response goes to 0 at integer multiples of the symbol period. Therefore, at the optimal sampling point of each symbol, there is no intersymbol interference (ISI).

In practice, filtering is also applied at the receiver. By matching the impulse response of the filter to the time-inverted pulse shape, the effect is similar to a correlation operation. The matched filter produces a peak that maximizes signal-to-noise ratio (SNR). In order to ensure that there is no ISI, the overall frequency response of the transmit filter and receive filter cascade should have a raised-cosine response. Since the overall frequency response is the product of the two, each filter should use the square root of the raised-cosine frequency response. This is known as a root-raised-cosine filter.

**4.2.2 PSK Flowgraph**

Figure 4.4 shows the transmitter portion of the PSK flowgraph. The source block produces a pre-defined sequence of bytes that can be identified as a staircase waveform in a time-domain plot. Only the two least significant bits are mapped into a QPSK symbol, and the rest of the bits are unused. The symbols are passed through an interpolating root-raised-cosine filter before being transmitted by the HackRF One. The associated variables, including the filter definition, are not shown.
Differential encoding is applied to the bit sequence before modulation. The effect is that bit sequences are actually encoded as relative transitions between symbols rather than as absolute symbols. If bits are encoded directly as symbols, a 90-degree rotation of the entire constellation diagram will cause every symbol to be decoded incorrectly. With differential encoding, this constant phase offset has no effect on the decoding process. The symbol table must ensure that the transitions between symbols are invariant to rotation. Figure 4.5 shows the symbol table used by the flowgraph, for which rotation has no effect. For each bit sequence, there is an integer number of counterclockwise transitions between symbols. The number of transitions is preserved even if the entire diagram is rotated. Differential encoding is one method of resolving the issue of phase ambiguity when performing blind phase synchronization.

Figure 4.5: Differential QPSK with rotational symmetry
Figure 4.6 shows the receiver portion of the flowgraph. The receiver applies synchronization, decodes symbols, and plots the demodulated bit sequence. Due to inherent differences between the transmitter and receiver hardware, the received data has sampling clock, sampling time, frequency, and phase offsets relative to the transmitted data. The purpose of synchronization is to remove these offsets. The constellation sink named “Before sync” plots data before synchronization is applied. The incoming data is passed through a decimating FIR filter before plotting. This receive filter is matched to the transmit filter and performs decimation by the same factor.

The first step in the synchronization process is to remove the sampling clock and sampling time offsets. This should recover the optimal sampling time, where the filters do not introduce any ISI. The constellation sink named “After clock sync” plots the result. The next step is to remove the frequency and phase offsets. This should recover the phase of the transmitted symbols and allow for demodulation. The constellation sink named “After phase sync” plots the result. Both the clock and phase synchronization methods are blind, meaning that they do not use any knowledge of the signal to apply correction. The flowgraph also applies automatic gain control so that the average magnitude of symbols after synchronization is equal to the magnitude of the transmitted symbols. This facilitates the comparison between received symbols and transmitted symbols, but it generally does not improve the ability to correctly decode symbols.
The symbol sync block performs clock synchronization and applies a decimating matched filter. The Costas loop block applies frequency and phase synchronization. Both methods are blind and employ a control loop with negative feedback to estimate and remove the corresponding offsets. The input samples are passed through an error detector, and the offsets are tracked by a loop filter with proportional and integral paths. The clock tracking loop filter estimates the instantaneous and average symbol timing offset, and the phase tracking loop filter estimates the phase offset and frequency offset. Over time, the loop should act to minimize error and obtain accurate estimates. The appropriate time or phase shift is applied to the output signal.

The symbol sync block allows for configuration of the loop filter parameters, the timing error detector, and the interpolating resampler. For this configuration, the interpolating resampler consists of a pair of filter banks. There are 32 filters in the first bank, each applying a different fractional delay. The 32 filters in the second bank differentiate the signal and apply the same fractional delays. Both banks apply the matched filter. The goal of the control loop is to choose the filter with the fractional delay that will minimize error. The error detector uses the pair of filters with the same fractional delay from each bank.

The Costas loop can only be used to synchronize PSK signals. The error detector is chosen for the symbol set being used, which is QPSK in this case. Equation (4.5) is the expression for the error detector, where \( y \) is the input signal. When the real part of a symbol has the same magnitude as the imaginary part, the error is 0. When the Costas loop has accurate phase and frequency offset estimates, the error is minimized. The opposite of the phase offset estimate is applied to the output. The Costas loop cannot distinguish any 90-degree phase rotations that still remain in the signal. This is the cause of the phase ambiguity that differential encoding resolves.

\[
e(n) = \text{sign}(\text{Re}(y(n))) \text{Im}(y(n)) - \text{sign}(\text{Im}(y(n))) \text{Re}(y(n))
\]  

(4.5)

Figure 4.7 shows the output of each constellation sink in the flowgraph. In the first diagram, there is a lack of synchronization. Samples are occasionally taken in between the optimal sampling points of adjacent symbols, and they appear at the center of the constellation diagram. With symbol timing recovered, the diagram has the appearance of a ring. With phase correction applied, the original constellations are clear. Although the symbols are properly distinguished, the constellation diagram alone cannot prove that errors have not occurred. It is possible for an incorrect number of symbol transitions to occur, which will cause an error when decoded. Figure 4.8 shows an example where the transmitted staircase waveform has been recovered without error.
The ability of the system to recover PSK symbols without error depends on many factors. The combination of transmit power, antenna separation and orientation, and receiver gain determines whether the receiver can detect the transmitted signal. These factors also influence the received SNR. Hardware impairments and the response of the wireless channel can also introduce ISI and reduce SNR. An important metric that encapsulates most of the impairments on system performance is error vector magnitude (EVM). EVM quantifies how the distribution of received symbols deviates from the transmitted symbols, which are used as reference points. Equation (4.6) gives the expression for the root-mean-square value of EVM for N symbols.

\[
EVM = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (I - I_{ref})^2 + (Q - Q_{ref})^2} \times 100\%
\] (4.6)
When EVM is low, constellations are closely distributed around the reference symbols and are able to be distinguished. When EVM is high, the constellations around each point can overlap. One cause of deviation from the reference symbols is noise, and deviation is more severe when SNR is low. Higher SNR is generally associated with lower EVM. The goal of varying system parameters is to maximize SNR and minimize EVM. This will result in the fewest number of errors.

System performance was tested for various hardware configurations, including two different RTL-SDR products. The RTL-SDR Blog design and the Nooelec NESDR SMArTee XTR SDR were used. Figure 4.9 shows a picture of the two designs and lists their nominal frequency ranges. While both designs use the RTL2832U for sampling and USB connectivity, they do not use the same RF front end. The RTL-SDR Blog design uses the R820T-2 tuner chip, and the Nooelec design uses the E4000 tuner chip. Although both designs should be able to receive the 1.576 GHz IF output of the Electrosense board, the R820T-2 seems to strongly attenuate signals above 1.5 GHz. This cut-off was observed by comparing received signal power directly above and below 1.5 GHz. The E4000 tuner does not have this problem and is able to receive well above 2 GHz. However, the tuner degrades signal quality as carrier frequency increases.
Figure 4.10 demonstrates that the system experiences a clear increase in EVM for higher carrier frequencies when using the Nooelec design. At 1.576 GHz, there are many symbols that lie in between two reference points and will be decoded incorrectly. The orange points on the diagrams represent the reference symbols. The carrier frequency is the only parameter that is varied between the two cases. The case with 200 MHz carrier frequency uses the bypass path of the Electrosense board, while the case with 1.576 GHz carrier frequency uses the downconversion path of the Electrosense board. The same increase in EVM can be seen for a 1.576 GHz carrier frequency using the bypass path. This indicates that the board itself is not the cause of this performance degradation. Symbol data from GNU Radio was written to a file and imported into Python. EVM is calculated within Python.

To improve performance, an increase in SNR should cause a decrease in EVM. The SNR will increase for higher transmit power, higher receiver gain, and reduced antenna separation. Figure 4.11 shows the constellation diagrams for three different SNR values. The carrier frequency is 1 GHz, and the receiver SDR is the Nooelec design. The spectrum plots of the received samples before synchronization are used to estimate SNR. When SNR increases from 20 dB to 27 dB, there is a 5% reduction in EVM. When the SNR increases from 27 dB to 40 dB, there is only a 0.3% reduction in EVM. Each plot contains only 1024 symbols, so a 0.3% reduction in EVM is rather insignificant. This example is an exception to the idea that higher SNR is associated with lower EVM. When the SNR exceeds 27 dB, it is no longer the limiting factor on performance. Further increases in SNR have almost no effect on the distribution of constellations.

Figure 4.12 compares the constellation diagrams for the Nooelec and RTL-SDR Blog designs for the same SNR value of 40 dB and carrier frequency of 1 GHz. This comparison clearly shows that the impairments caused by the Nooelec design are far more severe than those caused by the RTL-SDR Blog design.

![Figure 4.10: EVM comparison for 200 MHz and 1.576 GHz carrier frequencies using Nooelec SDR](image)
Figure 4.1: EVM comparison for RTL-SDR designs

Figure 4.11: EVM comparison for increasing SNR using Nooelec design

Figure 4.12: EVM comparison for RTL-SDR designs
Hardware correction algorithms and equalization could be implemented to remove the deterministic impairments. However, it seems that the Nooelec design experiences errors due to phase noise. The LO internal to the E4000 tuner introduces phase noise that is random and not removed by the synchronization process. For improved system performance, an SDR whose LO produces less phase noise could be used.

4.3 OFDM Implementation

Systems that employ orthogonal frequency-division multiplexing transmit data over multiple carrier frequencies. Each carrier frequency is referred to as a subcarrier, and each subcarrier is modulated individually. For a given bandwidth, the number of subcarriers determines the frequency spacing. Figure 4.13 depicts the spectrum of an OFDM signal, with subcarrier represented by different colors. This graphic shows 5 subcarriers. The bandwidth is equal to the frequency spacing multiplied by the number of subcarriers. All other subcarriers have a null where one subcarrier is at its maximum value. The transmitted OFDM signal is centered at the carrier frequency. Before sampling, the signal is converted to baseband by the RF front end. When the signal is processed by a discrete Fourier transform, there is no intercarrier interference if the signal is properly centered. If there is a frequency shift, the transform bins will occur at points on the spectrum that are in between subcarrier peaks. The absence of a frequency shift is necessary to maintain orthogonality and prevent interference between subcarriers.

![OFDM spectrum with 5 subcarriers](image)

*Figure 4.13: OFDM spectrum with 5 subcarriers*
OFDM grants several advantages. For the same bandwidth, an OFDM signal is far less susceptible to narrowband interference or fading compared to a single-carrier signal. The system can allocate more transmit power to, change the modulation type of, or even refrain from using subcarriers experiencing low SNR. In addition, the symbol duration of each OFDM subcarrier is much longer than that of a single-carrier signal with the same data rate. The duration of ISI introduced by the channel is a much smaller fraction of the subcarrier symbol duration, and its effect can be eliminated by a guard interval.

GNU Radio offers many specialized blocks to implement OFDM. Figure 4.14 shows the foundation for an OFDM system, which was used for simulation. The number of subcarriers is set to 64. The input data stream of modulated symbols is parallelized into 64 streams, which are used as the frequency bins for an inverse FFT. The IFFT output samples are then recombined into a single stream. The output of the inverse transform is the time-domain representation of a signal with the desired frequency spectrum. The time-domain signal is a superposition of modulated, orthogonal subcarriers. Figure 4.14 also shows that a forward FFT disassembles the OFDM signal into the original stream of symbols. Although this flowgraph illustrates the basic elements of an OFDM system, it is not very practical. Synchronization is needed to remove any frequency offset and prevent intercarrier interference. In addition, the receiver must correctly sort each group of 64 samples. Data cannot be recovered if samples are placed into incorrect bin locations before performing the forward FFT.

The hardware implementation flowgraph transmits packets. Since the start of a packet is defined, the receiver is able to identify the first sample of the OFDM signal. The packet structure facilitates the placement of samples into the correct bins. In addition, a preamble and header are appended to the data. The preamble is used for packet detection, allowing the first sample to be identified. The header identifies how many groups of 64 samples, or OFDM symbols, are contained in the packet. Figure 4.15
shows the OFDM transmitter, which constructs each packet, modulates the input data, and performs an inverse FFT. The output samples are scaled into the range that the transmitter can produce. The file source block streams characters from a text file into the flowgraph.

Figure 4.16 shows the OFDM receiver, which consists of several stages. The first stage is packet detection, which uses a correlation metric to identify the preamble sync words sent out by the transmitter. Since the sync words are known to the receiver, they are also used to estimate and remove frequency offset. It is crucial that the offset is removed before demodulating the payload. The header metadata is demodulated, and the sync words are used for channel estimation. Equalization is applied to both the header and payload data using the channel estimate. A cyclic prefix is appended to the payload data to act as a guard interval, while preserving orthogonality.
The results for OFDM follow a similar trend as the PSK results. For conditions that yield high SNR and low EVM, packets are detected and recovered reliably. For higher EVM values, packets are lost. A cyclic redundancy check (CRC) in the header ensures that there are no errors in the header metadata. Most importantly, this ensures that the packet length parameter is received correctly. Packets are lost when they are not detected or when the header CRC fails. Figure 4.17 shows an example of packet loss. The payload text sent in each of these packets contains a number, and one of the numbers is missing from the sequence. If every number in the sequence was unique, the rate of packet loss could be determined.

The advantage of the OFDM flowgraph over the PSK flowgraph is its ability to transmit data in packets, which facilitates the transfer of messages and files. However, the flowgraph acts only as a proof of concept and does not take advantage of the flexibility that OFDM offers.

As the demand for higher data rates continues, single devices occupy larger bandwidths. The ability to divide these large bandwidths among many subcarriers is crucial to maintaining throughput in noisy environments. Systems and networks are able to adapt subcarrier allocation to the environment and for varying channel usage. Multiple access methods allow for users to share bandwidth more efficiently. Multiple-input, multiple-output (MIMO) systems can achieve higher spectral efficiency. OFDM presents many opportunities for optimization, which is what makes it such a worthy candidate for data transmission in the ongoing development of wireless communication.

Figure 4.17: Output text file with packet loss
5. Conclusion
The transmitter and receiver communicate reliably for certain hardware configurations and unreliably for others. The Electrosense board fulfills its role as an RF front end, allowing the system to operate at carrier frequencies up to 6 GHz. The software presented in Chapter 4 represents a starting point for the two techniques of interest. To improve reliability, both techniques would benefit from improved equalization and hardware correction. The system could also be improved by replacing the RTL-SDR with an SDR that does not have strong attenuation or phase noise at the intermediate frequency of the Electrosense board.
References

[1] Electrosense hardware GitHub repository. Available at: https://github.com/electrosense/hardware

[2] Project GitHub repository. Available at: https://github.com/markmv2/sdr


