Spectrum Awareness Testbed

by

Andrea Walker

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 11, 2021

Keywords: Cognitive radio, spectrum sensing, compressive sensing

Copyright 2021 by Andrea Walker

Approved by

Mark L. Adams, Chair, Associate Professor of Electrical and Computer Engineering
Shiwen Mao, Ginn Distinguished Professor of Electrical and Computer Engineering
Stanley Reeves, Professor of Electrical and Computer Engineering

Abstract

This thesis presents a flexible and scalable spectrum awareness testbed targeting a wideband frequency range. Due to the static frequency allocation scheme, spectrum scarcity has become a problem in communications. Opportunistic spectrum access will allow secondary users to take advantage of empty portions of the spectrum to increase the efficiency of spectrum use. To allow that to happen, opportunistic users must be able to identify, characterize, and geolocate nearby transmitters.

The spectrum awareness testbed is capable of recovering the approximate carrier frequencies of an input transmission. It operates in the 5G Frequency Range 1 (5G FR1) and is currently configured to sense one transmission. The testbed uses the Modulated Wideband Converter (MWC) as a sub-Nyquist sampling scheme to acquire the input signal in hardware. The input is split into multiple channels. Each channel is then mixed with a periodic waveform, lowpass filtered, and sampled at a low rate for digital processing. The periodic waveform defines a relationship between the low-rate samples in each channel and the support of the input signal, which is recovered through a compressed sensing (CS) technique.

To verify operation of the testbed, the MWC system was simulated in Matlab. For the parameters selected for the hardware implementation, the simulation achieved a successful support recovery rate greater than 90% for SNR values larger than 5 dB. The MWC system was constructed in hardware and tested using a Hardware-in-the-Loop (HWIL) setup. Multiple carrier frequencies and signal bandwidths of 10 MHz and 80 MHz were evaluated. The largest successful percentage of support recovery for a signal with an 80 MHz was 43.28% for the carrier frequency 2.0 GHz. For a signal with 10 MHz bandwidth, the largest successful percentage of support recovery was 36.86% for the carrier frequency 2.0 GHz.

The performance of the testbed MWC hardware implementation did not meet the performance seen in simulation. This is likely attributed to low input signal power levels, the frequency range of the chosen mixer, and analog component inaccuracies. Possible solutions

are suggested to improve performance. Overall, the hardware implementation functions as a proof of concept for a wideband spectrum awareness testbed.

Acknowledgments

I would like to express my gratitude to my advisor and committee chair, Dr. Mark Adams, for the opportunity to work in the STORM Lab and his guidance and advice throughout my time as a graduate student.

I would like to thank my committee members Dr. Stanley Reeves and Dr. Shiwen Mao for their support.

I am thankful for my peers and friends in the STORM Lab for their support and encouragement throughout my time as a graduate student.

Finally, I am tremendously blessed by the support of my family for their unwavering love and support throughout my life.

This work was supported in part by the National Science Foundation under Grant CNS-1822055. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the foundation.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

5G FR1	5G Frequency Range 1
ADC	Analog-to-Digital Converter
AFE	Analog-Front-End
AWG	Arbitrary Waveform Generator
AWGN	Additive White Gaussian Noise
COTS	Commercial-off-the-Shelf
CR	Cognitive Radio
CS	Compressed Sensing
CSV	Comma Separated Value
CTF	Continuous-to-Finite
DFT	Discrete Fourier Transform
DTFT	Discrete-Time Fourier Transform
FFT	Fast Fourier Transform
FMC	FPGA Mezzanine Card
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface

HSDC Pro	High-Speed Data Converter Pro
HWIL	Hardware-in-the-Loop
LO	Local Oscillator
LPF	Lowpass Filter
MWC	Modulated Wideband Converter
NF	Noise Figure
OMP	Orthogonal Matching Pursuit
RF	Radio Frequency
RIP	Restricted Isometry Property
SFP	Soft Front Panel
SNR	Signal to Noise Ratio
SS	Spectrum Sensing
TI	Texas Instruments

Chapter 1

Introduction

Spectrum scarcity impedes practical implementations of emerging wireless multimedia applications that require a larger portion of the frequency spectrum. This increased demand for frequency spectrum has brought about a spectrum shortage. The fixed frequency allocation scheme that has been used for decades was considered optimal because it avoided interference between active wireless users; however, an increased number of wireless users in recent years has introduced a spectrum scarcity problem [1].

Much research in recent years has been put into cognitive radio (CR) technologies to allow spectrum sharing through opportunistic spectrum access [1]. Studies have shown that much of the time, the primary user of the assigned spectrum is either not using it or using it infrequently [2]. This situation creates temporal and spatial spectrum holes as illustrated in Figure 1.1. A temporal spectrum hole occurs when, for a given frequency band, the band is not occupied for some period of time. A spatial spectrum hole occurs when, for a given time, frequency bands across the spectrum may not be in use. Due to the presence of spectrum holes, it has been concluded that the spectrum scarcity problem is caused by inefficient allocation rather than physical shortage of spectrum [1]. Spectrum Sensing (SS) allows opportunistic users to take advantage of these spectrum holes in both time and frequency to use the spectrum more efficiently. To do this, it is necessary to be able to identify, characterize, and geolocate nearby transmitters without the use of $a\ priori$ information.

The goal of this work is to design, construct, and evaluate a spectrum awareness testbed to demonstrate the feasibility of spectrum sensing in a wideband regime. It targets the 5G FR1 range (400 Mhz-7.125 GHz) and does not rely on $a\ priori$ information. Specifically, this means that the carrier frequencies are unknown. The proposed system is designed to detect one

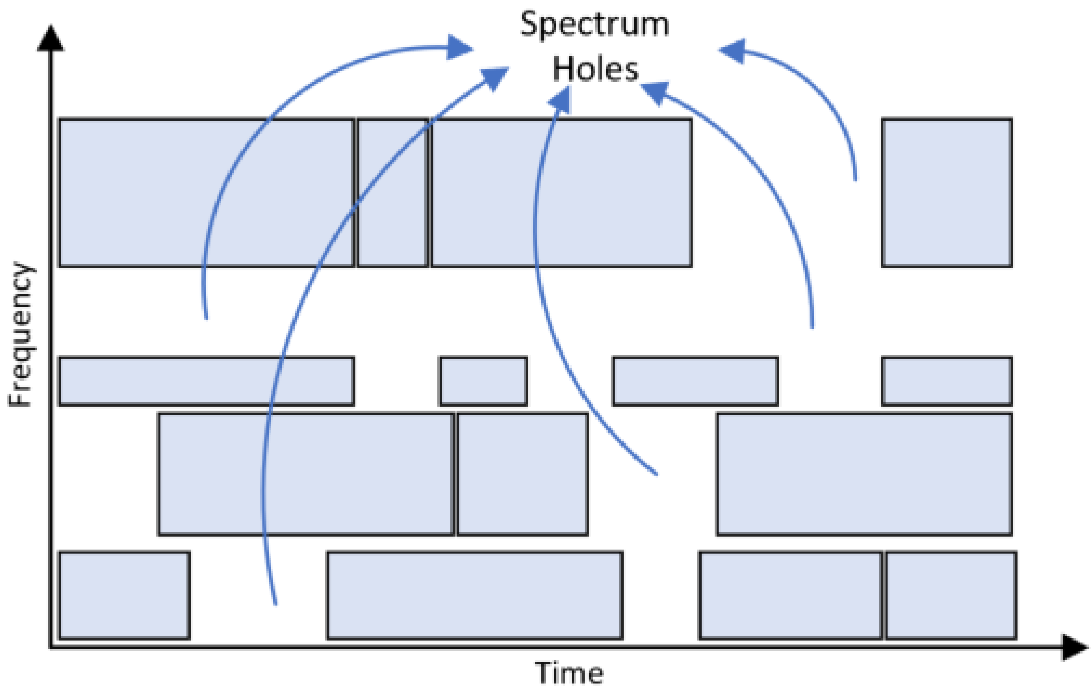


Figure 1.1: An example of spectrum holes in both time and frequency.

transmission with a maximum bandwidth of 80 MHz at a time. However, one of the biggest advantages of the solution lies in its scalability. Future iterations will be able to improve and expand the performance of the system by adjusting the operating parameters and hardware.

Chapter 2

Background

Spectrum sensing techniques can be assigned to two broad categories: narrowband and wideband spectrum sensing techniques. The difference between the two is in the range of frequencies each strategy can sense. Narrowband spectrum sensing techniques are limited to a single frequency band, whereas wideband spectrum sensing techniques are useful over a wider range of frequencies encompassing multiple frequency bands. In the case of wideband spectrum sensing, compressed sensing (CS) provides a means of achieving sub-Nyquist sampling for systems targeting a large Nyquist frequency [2]. Although wideband spectrum sensing techniques are more useful for the frequencies targeted by this project, it is worth reviewing the concept of spectrum sensing with an overview of narrowband spectrum sensing.

2.1 Narrowband Spectrum Sensing and Its Limitations

Some of the most common narrowband spectrum sensing techniques are energy detection, cyclostationary feature detection, and matched filtering [2]. They generally use a binary hypothesis model with

$$x(t) = \begin{cases} n(t) & 0 < t \leq T \quad H_0 \\ h * s(t) + n(t) & 0 < t \leq T \quad H_1 \end{cases} \tag{2.1}$$

In the binary hypothesis model, $x(t)$ is the received signal during observation window T, $n(t)$ is the additive white Gaussian Noise (AWGN) of the channel, $s(t)$ is the transmitted signal, and h is the channel gain. Hypothesis H_0 indicates that the frequency band is unoccupied, and hypothesis H_1 indicates the that the frequency band is occupied. Generally speaking, the spectrum sensing decision is made based on some threshold value that is calculated from the probability of detection p_d and the probability of false alarm p_f [1].

Narrowband spectrum sensing techniques are not directly applicable to a wider frequency range because they make a single decision for the entire range of spectrum under examination. Some wideband techniques divide the spectrum into narrower bands and perform narrowband spectrum sensing either sequentially or in parallel, but these approaches are costly in time and hardware, respectively [2]. Thus, narrowband spectrum techniques are extremely limited when it comes to examining a wider portion of the spectrum. Many narrowband techniques also commonly rely on some form of $a\ priori$ information, which was undesirable for this project.

2.2 Wideband Spectrum Sensing and Its Challenges

Clearly, narrowband spectrum sensing techniques alone are not sufficient for a testbed targeting a wider range of frequencies. Thus, a different approach is necessary to accomplish wideband spectrum sensing. Wideband spectrum sensing techniques reside in two categories: Nyquist wideband sensing and sub-Nyquist wideband sensing [3].

Nyquist wideband sensing performs spectrum sensing on digital signals taken at or above the Nyquist rate [3]. An example of this is the Multiband Joint Detection technique, illustrated in Figure 2.1. The wideband signal $x(t)$ is sampled by a high sampling rate Analog-to-Digital Converter (ADC) at the Nyquist rate, and then the Fast Fourier Transform (FFT) is taken of the digital signal. The wideband spectrum $X(f)$ is divided into a series of narrowband spectra for which spectrum holes were detected using the binary hypothesis test [2].

However, for the testbed to detect signals with carrier frequencies as large as 7.125 GHz, it would need to be capable of sampling at the Nyquist rate 14.25 GHz. There are few commercial off-the-shelf ADCs capable of sampling at such a high rate, and those that exist on the market are prohibitively expensive for the relatively inexpensive solution presented in this work. Another issue to consider is the analog input bandwidth of an ADC. It is not uncommon for the analog input bandwidth of high-rate ADCs to be much less than that of the sampling rate.

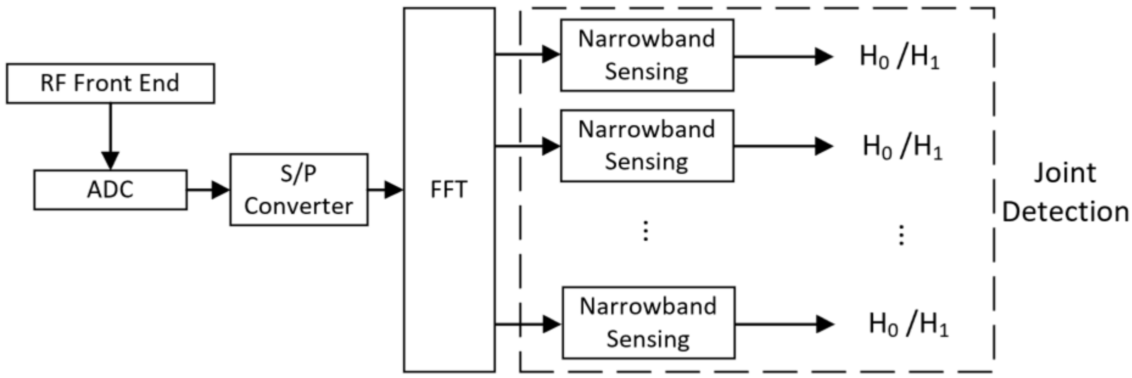


Figure 2.1: Illustration of Multiband Joint Detection

For these reasons, it is infeasible to sample at the Nyquist rate, so it is necessary to consider sub-Nyquist wideband sensing techniques.

Sub-Nyquist wideband sensing techniques perform spectrum sensing using a sampling rate that is lower than the Nyquist rate. From the Nyquist-Shannon sampling theorem, it is known that in order to sample and reconstruct a band-limited signal, the sampling rate must be at least twice the bandlimit. Otherwise, aliasing may occur upon reconstruction and destroy information present in the original signal [4]. Aliasing is generally undesirable, but the field of compressed sensing has opened avenues to recover signals at sub-Nyquist rates given some mild constraints on the signal.

2.2.1 Brief Overview of Compressed Sensing

Compressed sensing theory states that it is possible to accurately sample and recover certain signals at less than the Nyquist rate by solving underdetermined linear systems. To ensure accurate recovery, two conditions are necessary. The first condition is sparsity, which means that the signal must be sparse in some domain. The second condition is incoherence, which maintains that a signal must be spread out in the domain it is acquired in and sparse in some other domain [5]. It was shown in Chapter 1 that inefficient allocation of frequency bands allows for temporal and spatial spectrum holes that indicate sub-optimal use of the spectrum. In other words, it is reasonable to assume that the signals at a receiver are sparse in the frequency domain. It is also evident that an acquired signal will have a much denser representation in the

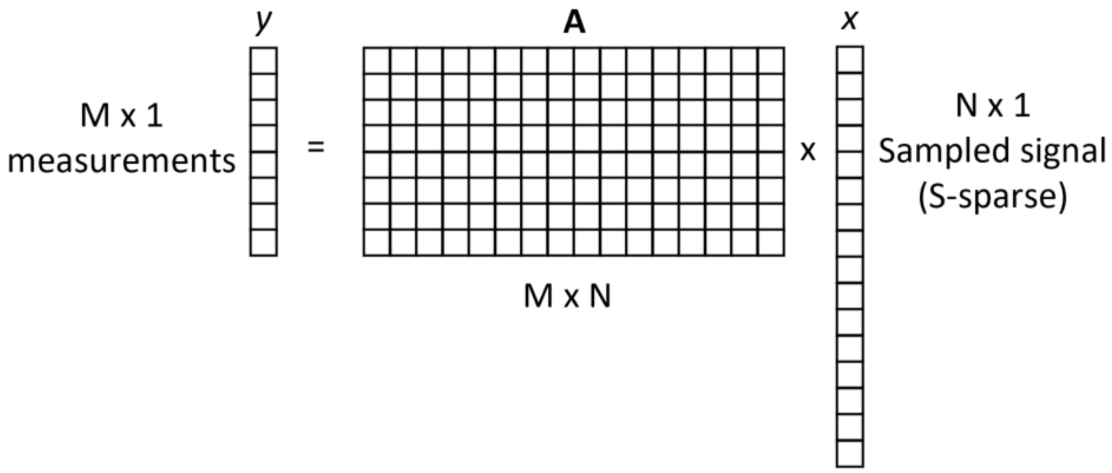


Figure 2.2: Illustration of Multiband Joint Detection

time domain in which it is observed. Therefore, any signals acquired at the receiver satisfy the two conditions necessary for correct recovery at sub-Nyquist rates.

The underdetermined system of linear equations $y = \mathbf{A}x$ in Figure 2.2 represents the compressed sensing problem where y is the vector of measurements taken at a sub-Nyquist rate, x is the desired signal that is sparse in some domain, and the matrix $\mathbf{A}$ represents a relationship that ties the sub-Nyquist measurements to the original sparse signal. For an underdetermined system of linear equations, there are infinitely many solutions [5]. Some popular methods of solving the compressed sensing problem are basis pursuit and orthogonal matching pursuit (OMP). These methods find the sparsest solution to the underdetermined system of linear equations [2]. For the testbed, OMP is used to recover the unknown carrier frequencies detected at a receiver.

2.2.2 Brief Overview of sub-Nyquist Sampling Methods

Compressed sensing provides a means of leveraging the sparsity of a signal is order to sample it at less than the Nyquist rate. However, the question still remains of how an input transmission can be acquired by hardware such that there is a relationship between the sub-Nyquist samples and the original sparse signal. The Modulated Wideband Converter (MWC) was used in this

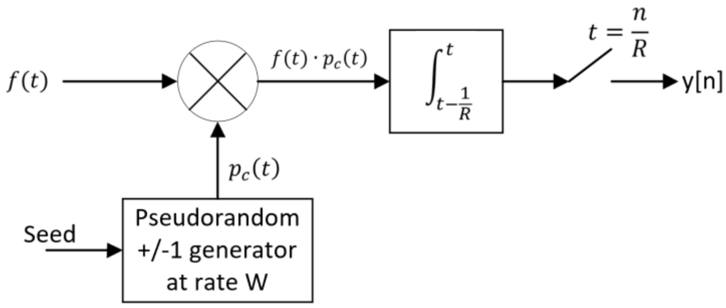


Figure 2.3: Top-level view of the Random Demodulator.

project as a hardware implementation of sub-Nyquist sampling. This section briefly describes some other hardware solutions to the sub-Nyquist sampling problem and their issues.

The Random Demodulator shown in Figure 2.3 is one such hardware solution. It multiplies the input signal $f(t)$ by a sign waveform generated from a pseudorandom sign generator [6]. The pseudorandom sign generator alternates at rate W, and the output of the mixer is integrated and sampled at rate R [6]. The signal modulator consists of multitone functions of the form in (2.2) with the finite set of tones in (2.3) [6].

$$f(t) = \sum_{\omega \in \Omega} a_\omega e^{-j2\pi\omega t}, t \in [0, 1) \tag{2.2}$$

$$\Omega \subset \{0, \pm, \pm 2, ..., \pm(W/2 - 1), W/2\} \tag{2.3}$$

It can be shown that $f(t)$ is recovered from the sequence $y[n]$ by the linear system

$$\mathbf{y} = \mathbf{\Phi A} \tag{2.4}$$

for an $R \times W$ matrix $\mathbf{\Phi}$ [6]. However, it turns out that the matrix $\mathbf{\Phi}$ is extremely large and generally has millions of rows and columns [7]. Computationally, the size of $\mathbf{\Phi}$ is limiting and makes it challenging to implement in practice from a digital signal processing perspective. Also, implementation of the integrator can present challenges as well [7].

Multi-coset sampling is another potential hardware solution for the sub-Nyquist sampling problem. Consider an input signal $x(t)$ that is sampled at the Nyquist rate, and whose samples

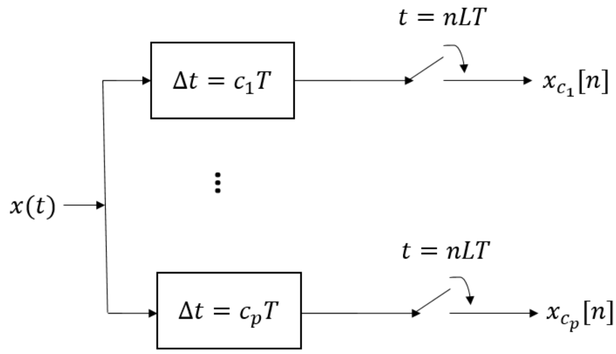


Figure 2.4: Overview of multi-coset sampling.

are defined by the Nyquist grid $x(nT)$. Multi-coset sampling chooses certain samples from the Nyquist grid in a periodic and nonuniform fashion [8]. The Nyquist grid is divided into blocks of L consecutive samples. The sampling pattern is defined by the set $C = \{c_i\}_i^p = 1$ where $0 \leq c_1 < ... < c_p \leq L - 1$. Here, p denotes the number of cosets [8]. The ith coset, or sampling sequence, is defined by (2.5).

$$x_{c_i}[n] = x(nLT + c_i T) \tag{2.5}$$

The average sampling rate of the multi-coset system is p/LT. This is smaller than the Nyquist rate because $p < L$ [8]. Figure 2.4 illustrates a multi-coset sampling system. Essentially, each channel samples at a rate $1/LT$ but has an offset in the ith channel of $c_i T$ from time $t = 0$. Unfortunately, there are two significant difficulties with this approach. First, it is very difficult to achieve accurate time delays between the ADC on the order of the Nyquist rate, which is on the order of several GHz. It has been noted that inaccuracies in the delays significantly affect the recovery of the signal [7].

The second issue arises from analog bandwidth at the input of the ADC. An ADC samples at some rate r samples/second. The analog bandwidth determines the maximal frequency b that the ADC can process, and any spectral content above this limit is distorted [7]. For the multi-coset system, although the sampling rate for each ADC in the system has been reduced to well below the Nyquist rate, the input signal could have spectral content that is well above

b [7]. Because of this phenomenon, it would be necessary to use ADCs that have an analog bandwidth b that is on the order of the Nyquist rate. This is an atypical use of an ADC, so for commercial off-the-shelf devices, it would be very difficult to find an ADC that matches the specifications of the system. Furthermore, the sampling rate and analog bandwidth of COTS ADCs are usually similar, so to purchase an ADC with the appropriate analog bandwidth would mean that the sampling rate is well over specifications [7]. Conversely, an ADC with the appropriate sampling rate and analog bandwidth could be designed at the cost of great complexity, which was well beyond the scope of this project. Either way, the cost of the ADC for multi-coset sampling would be prohibitive.

Chapter 3

The Modulated Wideband Converter (MWC)

The Modulated Wideband Converter (MWC) is a hardware solution to the sub-Nyquist sampling problem for applications in wideband spectrum sensing. It is capable of recovering the unknown carrier frequencies of input transmissions with relatively low hardware complexity and sampling rate. The MWC was developed as an improvement over the multi-coset system described in Chapter 2.2.2. Specifically, it reduces the analog bandwidth at the input of the ADC and uses synchronized sampling in all channels to avoid implementing small time offsets in the ADC [7].

3.1 Background

The MWC considers a multiband model with multiple transmissions of bandwidth B at carrier frequencies f_i within some range $\mathcal{F} = [-f_{NYQ}/2, f_{NYQ}/2]$, as shown in Figure 3.1. For an input signal $x(t)$ within this multiband model, the MWC processes the input signal as shown in Figure 3.2. The input signal $x(t)$ is initially split into m channels. In the ith channel, $x(t)$ is mixed with a periodic waveform $p_i(t)$ that has period T_p [7]. In this version of the MWC, the periodic waveforms in each channel are repeated M-length random sequences where

$$p_i(t) = \alpha_{ik}, \quad k\frac{T_p}{M} \leq t \leq (k+1)\frac{T_p}{M}, \quad 0 \leq k \leq M-1 \tag{3.1}$$

and each $\alpha_{ik} \in \{+1, -1\}$ is drawn randomly. Mixing with the periodic waveform aliases the input signal to create different linear combinations of the input signal in each channel. The corresponding frequency $f_p = 1/T_p$ determines the rate of aliasing. Each channel is then put

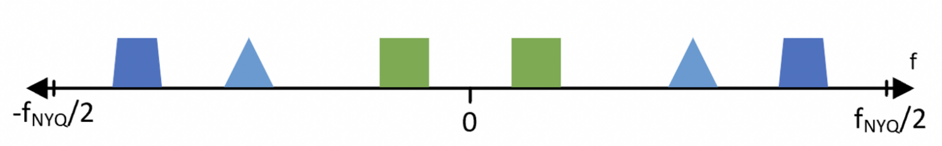


Figure 3.1: The multiband model with transmissions at different carrier frequencies.

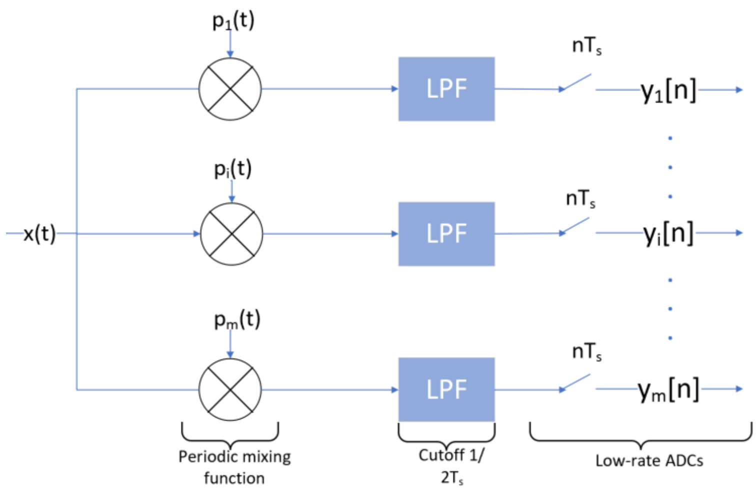


Figure 3.2: Overview of the MWC.

through a lowpass filter with cutoff $\frac{1}{2T_s}$ and sampled at the low rate $F_s = \frac{1}{T_s}$ where $F_s << F_{NYQ}$. The digital sequences $y_i[n]$ are related to the support set of the original input signal $x(t)$ through the coefficients of the mixing functions $p_i(t)$ [7]. This relationship is described in further detail in the following section.

3.2 Mathematical Analysis

This section derives the relationship between the unknown carrier frequencies of the input signal $x(t)$ and the low-rate digital sequences $y_i[n]$. It is first useful to define the frequency of the mixing functions f_p and the sampling rate f_s in (3.2) [7].

$$\begin{aligned} f_p &= \frac{1}{T_p}, \quad \mathcal{F}_p = [-\frac{f_p}{2}, +\frac{f_p}{2}] \\ f_s &= \frac{1}{T_s}, \quad \mathcal{F}_s = [-\frac{f_s}{2}, +\frac{f_s}{2}] \end{aligned} \tag{3.2}$$

The mixing function $p_i(t)$ in the ith channel is periodic, so it has the Fourier expansion in (3.3).

$$p_i(t) = \sum_{l=-\infty}^{\infty} c_{il} e^{j\frac{2\pi}{T_p}lt} \tag{3.3}$$

where the Fourier coefficients are calculated by (3.4) [7].

$$c_{il} = \frac{1}{T_p} \int_0^{T_p} p_i(t) e^{-j\frac{2\pi}{T_p}lt}\, dt \tag{3.4}$$

Mixing is represented as the analog multiplication $\tilde{x}_i(t) = x(t)p_i(t)$ and is calculated in the frequency domain by (3.5).

$$\begin{aligned} \tilde{X}_i(f) &= \int_{-\infty}^{\infty} \tilde{x}_i(t) e^{-j2\pi ft}\, dt \\ &= \int_{-\infty}^{\infty} x(t) (\sum_{l=-\infty}^{\infty} c_{il} e^{j\frac{2\pi}{T_p}lt}) e^{-j2\pi ft}\, dt \\ &= \sum_{l=-\infty}^{\infty} c_{il} \int_{-\infty}^{\infty} x(t) e^{-j2\pi(f-\frac{l}{T_p})t}\, dt \\ &= \sum_{l=-\infty}^{\infty} c_{il} X(f - lf_p) \end{aligned} \tag{3.5}$$

From (3.5), the input to the lowpass filter $h(t)$ is a linear combination of shifted copies of $X(f)$, where the shifts are integer multiples of f_p. Without taking sparsity into account, the sum in (3.5) has no more than $\lceil f_{NYQ}/f_p \rceil$ nonzero terms because $X(f) = 0$ for $f \notin \mathcal{F}$ [7].

The ideal frequency response of the lowpass filter is shown in Figure 3.3. The purpose of the lowpass filter is to limit frequencies in the sampled sequences $y_i[n]$ to the interval $\mathcal{F}_s$. Because of this constraint, the discrete-time Fourier transform (DTFT) of the ith sequence $y_i[n]$ is defined by (3.6).

$$Y_i(e^{j2\pi fT_s}) = \sum_{n=-\infty}^{\infty} y_i[n] e^{-j2\pi fnT_s}, f \in \mathcal{F}_s, \tag{3.6}$$

and it follows that the DTFT of the sampled sequences $y_i[n]$ in (3.6) is equal to the analog equation of (3.5) on the interval $f \in \mathcal{F}_s$, as shown in (3.7) [7].

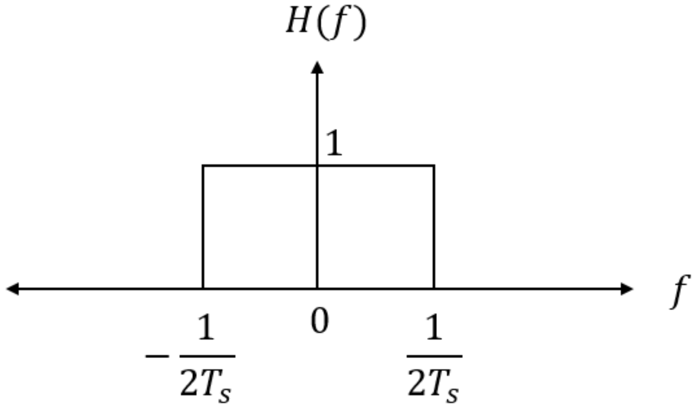


Figure 3.3: Ideal frequency response of the lowpass filter $H(f)$.

$$\sum_{n=-\infty}^{\infty} y_i[n]e^{-j2\pi f n T_s} = \sum_{l=-L_0}^{L_0} c_{il}X(f - lf_p), f \in \mathcal{F}_s \tag{3.7}$$

The change on the limits of the sum in (3.7) occurs because of the sampling interval $\mathcal{F}_s$. Here, L_0 is the smallest integer that can be found so the sum in (3.7) contains all nonzero contributions of $X(f)$ over the interval $\mathcal{F}_s$ [7]. L_0 can be found using (3.8).

$$L_0 = \lceil \frac{f_{NYQ} + f_s}{2f_p} \rceil - 1 \tag{3.8}$$

Equation (3.7) defines the relationship between the low-rate sampled sequences $y_i[n]$ and the support set of the input signal $x(t)$ [7]. It can be rewritten in matrix-vector form as

$$\mathbf{y}(f) = \mathbf{A}\mathbf{z}(f), f \in \mathcal{F}_s. \tag{3.9}$$

In (3.9), the vector $\mathbf{y}(f)$ has length m and ith element $y_i(f) = Y_i(e^{j2\pi f T_s})$ [7]. The unknown support set $\mathbf{z}(f) = [z_1(f), ..., z_L(f)]^T$ has length

$$L = 2L_0 + 1 \tag{3.10}$$

with

$$z_i(f) = X(f + (i - L_0 - 1)f_p), 1 \leq i \leq L, f \in \mathcal{F}_s$$
$$= \begin{bmatrix} X(f - L_0 f_p) \\ X(f - (L_0 - 1)f_p) \\ \dots \\ X(f + (L_0 - 1)f_p \\ X(f + L_0 f_p) \end{bmatrix}. \tag{3.11}$$

The matrix $\mathbf{A}$ has dimensions $m \times L$ and coefficients c_{il}

$$\mathbf{A}_{il} = c_{i,-l} = c_{il}^* \tag{3.12}$$

The reversed order of coefficients $c_{i,-l}$ in (3.12) is caused by the order of $z_i(f)$ in (3.7). Each $z_i(f)$ is a frequency slice of $X(f)$ with width f_p [7]. Determining the nonzero $z_i(f)$ is satisfactory to recover the support set of $x(t)$. In this system, the parameters f_p, f_s, and M can be chosen for different sensing environments [7]. The following list outlines the assumptions used in the testbed [7]:

1. $f_s \geq f_p \geq B$
2. $M \geq M_{min}$, where M_{min} is defined by $M_{min} = 2\lceil \frac{f_{NYQ}}{2f_p} + \frac{1}{2} \rceil - 1$
3. $m \geq 4N$ for blind reconstruction

It should be noted that an additional factor of two can be saved for Item 3, but at the expense of additional digital processing [8]. That method is not used in this design but could be useful in a future version. Item 1 ensures that $\mathbf{z}(f)$ has at most N nonzeros, where N is the maximum number of active transmissions [7].

3.2.1 Calculating the Matrix $\mathbf{A}$

Equation (3.12) demonstrates that each mixing function $p_i(t)$ contributes one row in the matrix $\mathbf{A}$. Each $p_i(t)$ should possess sufficient uniqueness to have linearly independent rows in $\mathbf{A}$ [7].

Recall the coefficients c_{il} of the random periodic waveform in (3.4), and let them be expanded in (3.13).

$$\begin{aligned} c_{il} &= \frac{1}{T_p}\int_0^{T_p/M} p_i(t)e^{-j\frac{2\pi}{T_p}lt}\,dt \\ &= \frac{1}{T_p}\int_0^{T_p/M}\sum_{k=0}^{M-1}\alpha_{ik}e^{-j\frac{2\pi}{T_p}l(t+k\frac{T_p}{M})}\,dt \\ &= \frac{1}{T_p}\sum_{k=0}^{M-1}\alpha_{ik}e^{-j\frac{2\pi}{M}lk}\int_0^{T_p/M} e^{-j\frac{2\pi}{T_p}lt}\,dt \end{aligned} \tag{3.13}$$

Evaluating the integral in (3.13) and letting $\theta = e^{-j\frac{2\pi}{M}}$ produces

$$\begin{aligned} d_l &= \frac{1}{T_p}\int_0^{T_p/M} e^{-j\frac{2\pi}{T_p}lt}\,dt \\ &= \frac{-1}{j2\pi l}[e^{-j\frac{2\pi l}{M}} - 1] \\ &= \frac{-1}{j2\pi l}[\theta^l - 1] \\ &= \frac{1-\theta^l}{j2\pi l} = \begin{cases} \frac{1}{M} & l = 0 \\ \frac{1-\theta^l}{j2\pi l} & l \neq 0 \end{cases} \end{aligned} . \tag{3.14}$$

With the knowledge of (3.14), (3.13) is rewritten in (3.15).

$$c_{il} = d_l \sum_{k=0}^{M-1} \alpha_{il}\theta^{lk} \tag{3.15}$$

Next, define $\bar{\mathbf{F}}$ as the $M \times M$ discrete Fourier transform (DFT) matrix with ith column

$$\bar{\mathbf{F}}_i = [\theta^{0i}, \theta^{1i}, ..., \theta^{(M-1)i}]^T, 0 \leq i \leq M-1, \tag{3.16}$$

and then choose $\mathbf{F}$ as a column subset of $\bar{\mathbf{F}}$ that is ordered to reflect the enumeration of $\mathbf{A}$ in (3.12).

$$\mathbf{F} = [\bar{\mathbf{F}}_{L_0}, ..., \bar{\mathbf{F}}_{-L_0}] \tag{3.17}$$

Finally, letting $\mathbf{S}$ be an $m \times M$ with $\mathbf{S}_{ik} = \alpha_{ik}$ and $\mathbf{D}$ be an $L \times L$ matrix where $\mathbf{D} = diag(d_{L_0}, ..., d_{-L_0})$ allows (3.9) to be rewritten in (3.18) and expanded upon in (3.19) [7].

$$y(f) = \mathbf{SFDz}(f), f \in \mathcal{F}_s \tag{3.18}$$

$$\begin{bmatrix} Y_1(e^{j2\pi f T_s}) \\ \vdots \\ Y_m(e^{j2\pi f T_s}) \end{bmatrix} = \begin{bmatrix} \alpha_{1,0} & \cdots & \alpha_{1,M-1} \\ \vdots & \ddots & \vdots \\ \alpha_{m,0} & \cdots & \alpha_{m,M-1} \end{bmatrix} \begin{bmatrix} | & \cdots & | & \cdots & | \\ \bar{\mathbf{F}}_{L_0} & \cdots & \bar{\mathbf{F}}_0 & \cdots & \bar{\mathbf{F}}_{-L_0} \\ | & \cdots & | & \cdots & | \end{bmatrix} \begin{bmatrix} d_{L_0} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & d_{-L_0} \end{bmatrix} \begin{bmatrix} X(f - L_0 f_p) \\ \vdots \\ X(f) \\ \vdots \\ X(f + L_0 f_p) \end{bmatrix} \tag{3.19}$$

3.3 Reducing Physical Channels

One of the necessary conditions for successful support reconstruction using the MWC is that the number of physical channels must fulfill $m \geq 4N$, where N is the number of possible transmissions (including both the positive and negative frequency axes) [8]. For example, Figure 3.1 has $N = 6$. For that signal, the required number of physical channels would be $m \geq 24$. It is easy to see that the number of physical channels can rapidly grow, and each additional channel impacts the cost and complexity of the hardware implementation. Therefore, it is desirable to reduce the number of physical channels.

It is possible to sample at a larger rate f_s in order to reduce the number of physical channels [7]. This objective is achieved by setting $f_s = qf_p$ for an odd integer $q = 2q' + 1$. In this scenario, the number of physical channels is reduced to $m \geq \lceil \frac{4N}{q} \rceil$, but at the expense of more complex digital signal processing [8][7]. To see how this is possible, consider the channel i in (3.7) for any $f \in \mathcal{F}_p$

$$
\begin{aligned}
y_i(f + kf_p) &= \sum_{l=-\infty}^{\infty} c_{il} X(f + kf_p - lf_p) \\
&= \sum_{l=-L_0-k}^{L_0-k} c_{i,(l+k)} X(f - lf_p). \\
&= \sum_{l=-L_0}^{L_0} c_{i,(l+k)} X(f - lf_p)
\end{aligned}
\tag{3.20}
$$

In (3.20), it is important to note that $-q' \leq k \leq q'$. The first line of (3.20) occurs from changing the variable from f to $f + kf_p$. The result is that the system now provides q equations for each physical channel in (3.9). For a single channel with $f \in \mathcal{F}_p$, (3.21) demonstrates how the additional virtual channels added to the system [7].

$$
\begin{bmatrix} Y_i(f - q'f_p) \\ \vdots \\ Y_i(f) \\ \vdots \\ Y_i(f + q'f_p) \end{bmatrix}
=
\begin{bmatrix}
c_{i,L_0-q'} & & & \cdots & & & c_{i,-L_0-q'} \\
\vdots & \ddots & & & & & \vdots \\
c_{i,L_0} & \cdots & c_1 & c_{i,0} & c_{i,-1} & \cdots & c_{i,-L_0} \\
\vdots & & & & & \ddots & \vdots \\
c_{i,L_0+q'} & & & \cdots & & & c_{i,-L_0+q'}
\end{bmatrix}
\begin{bmatrix} | \\ \mathbf{z}(f) \\ | \end{bmatrix}
\tag{3.21}
$$

From (3.21), it is evident that the additional rows are shifts of the original coefficients. In order to construct the new matrix $\bar{\mathbf{A}}$, first calculate the matrix $\mathbf{A}$ for the m physical channels. Then perform the shifts described in (3.21) for each physical channel. The expanded matrix $\bar{\mathbf{A}}$ now has mq rows [7].

The left-hand side of (3.21) must account for the shifts in $\bar{\mathbf{A}}$ as well. In other words, the sequences must be time-modulated, lowpass filtered, and downsampled to the interval $f \in \mathcal{F}_p$ before reconstruction [7]. The frequency shifts $y_i(f + kf_p)$ for $-q' \leq k \leq q'$ are accomplished by time modulation. Each sequence is then truncated by a digital lowpass filter $h_D[n]$ that has a cutoff frequency of $f_s/2q$, which is equivalent to a cutoff frequency of $fp/2$. Finally, the filtered sequences are decimated by the factor q. Equation (3.22) demonstrates how the additional sequences are derived by time-modulation and lowpass filtering [7].

$$\begin{aligned}\tilde{y}_{i,k}[\tilde{n}] &= (y_i[n]e^{-j2\pi k f_p n T_s}) * h_D[n]|_{n=\tilde{n}q} \\ &= (y_i[n]e^{-j\frac{2\pi}{q}kn}) * h_D[n]|_{n=\tilde{n}q}\end{aligned} \tag{3.22}$$

3.4 Support Reconstruction

This section describes the process of recovering the carrier frequencies from the original input signal, via the compressed sensing problem in Equation (3.9). Since the spectrum range $[\frac{-f_{NYQ}}{2}, \frac{f_{NYQ}}{2}]$ is divided into L slices of width f_p, then each slice represents one band that is either occupied or unoccupied. Recovering the occupied slices amounts to recovering the support of $\mathbf{z}(f)$. This is sufficient for estimating the carrier frequencies of the original signal $x(t)$. The support of a vector is the set of nonzero elements of the vector [7].

Section 3.2.1 demonstrates how to calculate the matrix $\mathbf{A}$ for the compressed sensing problem. To guarantee recovery of the support of $x(t)$, the matrix $\mathbf{A}$ must have spark equal to $N + 1$, where the spark of a matrix is the smallest number N such that there exists a set of N columns in $\mathbf{A}$ that are linearly dependent [9]. Finding the spark of a matrix is NP-hard and generally considered to be too computationally expensive to calculate [9]. The Restricted Isometry Property (RIP) says that the matrix $\mathbf{A}$ will have the RIP if there is some $0 \leq \delta_K < 1$ such that

$$(1 - \delta_K)||\mathbf{u}^2 \leq ||\mathbf{A}\mathbf{u}||^2 \leq (1 + \delta_K)||\mathbf{u}||^2 \tag{3.23}$$

for every N-sparse vector $\mathbf{u}$ [10]. Although the RIP is also computationally challenging to verify for matrices, it has been shown that random sign matrices of the form in (3.1) with size $m \times L$ will satisfy the RIP for N-sparse vectors with high probability as long as $m \geq CN \log(M/N)$ for a positive constant C [10]. This property justifies the choice of random sign matrices for the mixing functions $p_i(t)$.

To calculate the support of the vector $\mathbf{z}(f)$, the Continuous to Finite (CTF) block in Figure 3.4 is applied to the measurement vectors $y_i[n]$ [9]. It uses a spectrum blind reconstruction

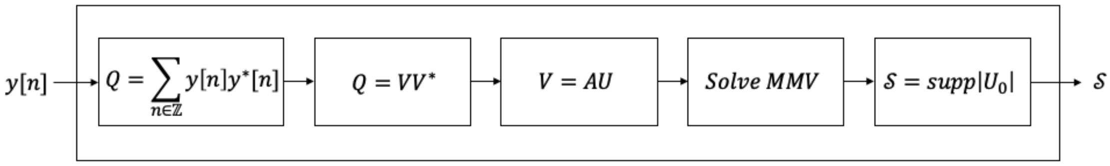


Figure 3.4: The Continuous to Finite Block (CTF).

Algorithm 1 SBR4

Require: Input: $y[n]$
Compute $\mathbf{Q} = y_i[n]y_i[n]'$
Decompose $\mathbf{Q}$ into frame $\mathbf{V}$ using eigendecomposition
Solve $\mathbf{V} = \mathbf{AU}$ by Orthogonal Matching Pursuit (OMP) for sparsest $\mathbf{U}$
return $\mathbf{U}$, the sparsest solution (the support of $\mathbf{z}(f)$)

algorithm, SBR4, to recover the support for $x(t)$ [8]. SBR4 is presented in Algorithm 1. $\mathbf{Q}$ is calculated by

$$\mathbf{Q} = \int_{f \in \mathcal{F}_s} \mathbf{y}(f)\mathbf{y}^H(f)df = \sum_{n=-\infty}^{\infty} \mathbf{y}[n]\mathbf{y}^T[n] \tag{3.24}$$

for the vector of samples $\mathbf{y}[n] = [y_1[n], ..., y_m[n]]^T$. Using the matrix $\mathbf{Q}$ found in (3.24), a frame $\mathbf{V}$ is found using an eigendecomposition, where $\mathbf{V}$ is chosen to be the matrix of eigenvalues corresponding to the nonzero eigenvalues or the eigenvalues that are above some threshold [9].

Algorithm 2 OMP

Require: Input: $\mathbf{A}, \mathbf{V}$
Initialize residual$= \mathbf{V}$
while more iterations **do**
  $\mathbf{b} \leftarrow ||\mathbf{A}' *$residual$||_2$
  $BestPos = max(\mathbf{b})$
  $\mathbf{U} \leftarrow \mathbf{U} \bigcup (BestPos)$
  $x \leftarrow A_U A_U^{\dagger} \mathbf{V}$
  residual$= \mathbf{V} - \mathbf{x}$
end while
return $\mathbf{U}$, the sparsest solution (the support)

The goal of $\mathbf{V} = \mathbf{AU}$ is to find the matrix $\mathbf{U}$ with the largest number of zero rows that matches $\mathbf{V}$ [9]. Once the frame $\mathbf{V}$ has been calculated, Orthogonal Matching Pursuit (OMP), a greedy recovery algorithm, is used to find the support of $x(t)$. OMP is outlined in Algorithm

2 [9]. Initially, the residual is set equal to $\mathbf{V}$. In each iteration, the support is incremented by finding the column of $\mathbf{A}$ that is most correlated to the signal residual. Then, a partial estimate of the signal is calculated and subtracted from the original $\mathbf{V}$ to update the residual [9]. The algorithm iterates $N/2$ times, the max number of possible transmissions that is assumed.

Chapter 4

Testbed Design

This chapter outlines the software simulation and hardware design for the Spectrum Awareness Testbed implemented using the MWC. First, the general parameters used for the MWC are described and justified. What follows is a description of the simulation used to verify the choice of parameters and overall operation of the MWC. A description of the analog front-end (AFE) and discussion of the specific hardware choices closes out the chapter.

4.1 System Parameters

	Parameter Set 1	Parameter Set 2
Number of transmissions N	4	2
Nyquist Frequency f_{NYQ}	14.25 GHz	14.355 GHz
Maximum bandwidth B	100 MHz	80 MHz
Expansion Factor q	7	7
Mixing function frequency f_p (MHz)	104.01 MHz	87 MHz
Sampling rate f_s (MHz)	728.10 MHz	609 MHz
LPF Cutoff Frequency $f_s/2$	364.05 MHz	304.5 MHz
Number of sign-alternations per period M	137	165
Number of analog channels m	3	2
Number of virtual channels mq	21	14

Table 4.1: MWC parameters.

Parameter Set 1 in Table 4.1 shows the original plan for the MWC operating parameters. This design would be able to support up to two simultaneous transmissions ($N = 4$) with bandwidth no greater than $B = 100$ MHz and carrier frequencies up to 7.125 GHz, which is the upper bound on 5G Frequency Range 1 (FR1). For the original parameters, the mixing function frequency f_p was chosen to be slightly larger than the maximum bandwidth in order to avoid edge effects [7]. Then, the sampling frequency was calculated by the multiplication $f_s = qf_p = 728.10$ MHz. It follows that the lowpass filter cutoff frequency should be $364.05MHz$. The value for M is calculated using $f_{NYQ}/f_p = 137$, and the number of physical channels is $m = \lceil \frac{4N}{q} \rceil = 3$.

However, due to hardware inaccuracies in the lowpass filter, the parameters had to be modified late into the project. Although the lowpass filter had the correct cutoff frequency in simulation, in practice the cutoff frequency did not match the designed value. The hardware issues are described in greater detail in Section 4.3.1. Due to time constraints, the decision was made to modify the MWC parameters instead of prolonged hardware troubleshooting.

Parameter Set 2 of Table 4.1 shows the modified MWC operating conditions. In this version, the MWC can sense a single transmission ($N = 2$) with bandwidth no greater than $B = 80$ MHz and carrier frequencies up to 7.125 GHz. The maximum carrier frequency, and subsequently the Nyquist frequency $f_{NYQ} = 14.355$ GHz, were increased simply to produce integers for the mixing function frequency f_p and sampling frequency f_s. Having integer numbers for those frequencies simplifies the generation of both the sampling clock and the mixing functions in hardware. Despite the increase, the MWC will only be tested with signals with carrier frequencies in 5G FR1 from 410 MHz to 7.125 GHz.

Because of the reduced lowpass filter cutoff frequency, Parameter Set 2 had to take this pre-determined value into account to calculate the rest of the MWC parameters. The rest of the parameters were derived by taking this constraint into account. The LPF cutoff frequency of 300 MHz implies that the sampling rate must be approximately 600 MHz. In practice, $f_s = 609$ MHz was chosen because it is divisible by the expansion factor $q = 7$. It follows that the mixing function frequency is $f_p = f_s/q = 87$ MHz. The number of sign-alternations per period M is then $M = fNYQ/f_p = 165$. Also, the number of physical channels was decreased to reduce hardware complexity. For this implementation, the number of physical channels is $m = \lceil \frac{4*2}{7} \rceil = 2$.

4.2 Software Design

Before performing hardware tests, the MWC was simulated using Matlab. The simulation encompasses the system from input signal generation to support recovery. Appendix A shows the Matlab script used to perform the simulation. The latter parts of the simulation are reused for the digital signal processing and support recovery of the data recorded from the hardware implementation of the MWC. For all of the simulations, the input to the MWC takes the form $x(t) + n(t)$, where $x(t)$ is a multiband signal and $n(t)$ is Additive White Gaussian Noise (AWGN). The multiband signal takes the form in (4.1).

$$x(t) = \sum_{i=1}^{N/2} \sqrt{E_i B} sinc(B(t - \tau_i)) cos(2\pi f_i (t - \tau_i)), \tag{4.1}$$

where E_i are the energy coefficients, B is the bandwidth, τ_i are the time offsets, and f_i are the carrier frequencies [7].

4.2.1 Simulation Description

This section demonstrates all of the steps in the MWC Matlab simulation in order to illustrate the mathematical concepts described in Chapter 3. A high-level block diagram of the simulation is given in Figure 4.1. For this example, the modified MWC parameters in Table 4.1 are simulated with $E_i = 1$, $\tau_i = 0.5$, SNR $=$ 20 dB and $f_i = 3.5$ GHz. To begin, the signal described by (4.1) is generated. Its time-domain and frequency-domain representation is presented in Figure 4.2.

Next, the support set of the input signal is calculated. Recall that the spectrum is divided into f_p-wide slices, and the slice edges are defined by the range $[1, L]$. Figure 4.3 shows an illustration of this concept by using the carrier frequency $f_i = 3.5$ GHz as an example. Note that the support for a single carrier frequency is defined as the lowest index that is nearest to the left edge and the lowest index that is nearest to the right edge of the signal. Appendix B provides a Matlab function to calculate the support set for a carrier frequency given f_p, L_0, and B. It follows that the support for $f_i = 3.5$ GHz that includes both positive and negative frequencies is the set in (4.2).

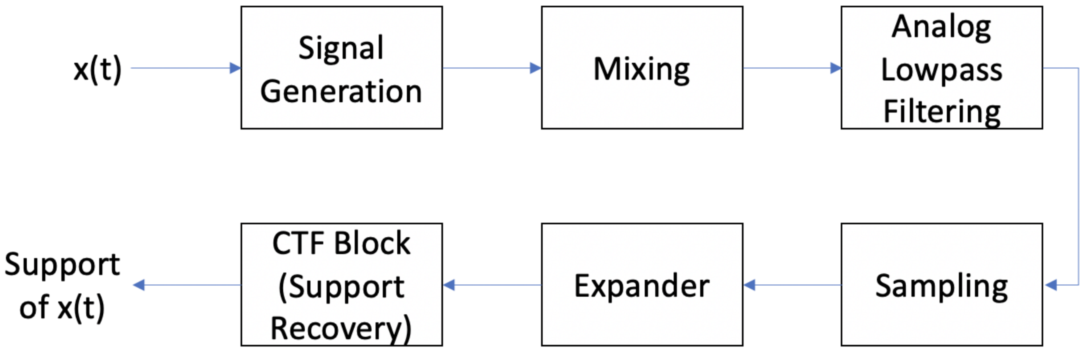
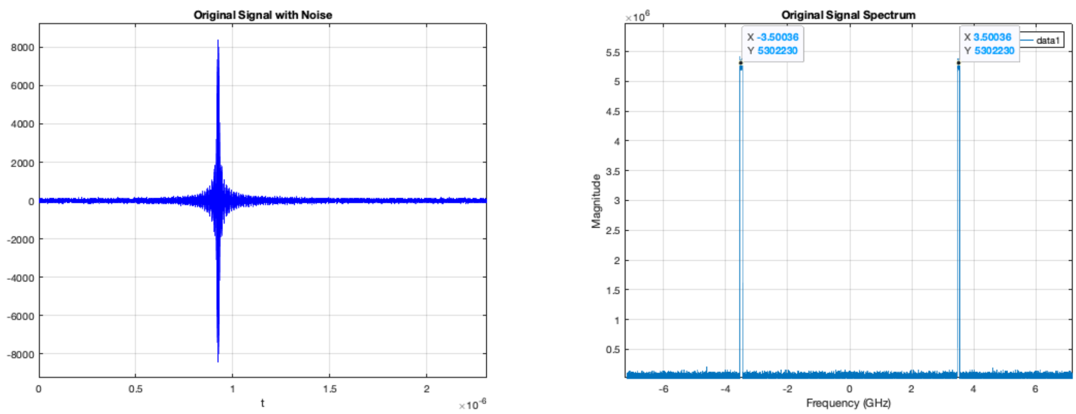


Figure 4.1: Incorrect image of the simulation block diagram...



(a) The original signal and noise $x(t) + n(t)$.

(b) Spectrum of the original signal $x(t)$.

Figure 4.2: The original signal.

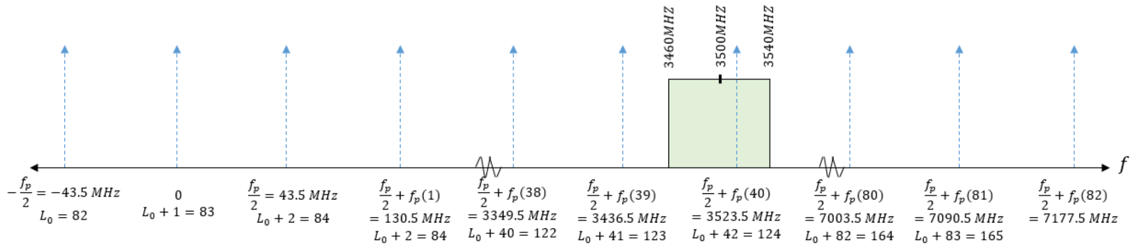


Figure 4.3: Illustration of the spectrum divided into f_p-wide slices and the placement of an input signal with $f_i = 3.5$ GHz.

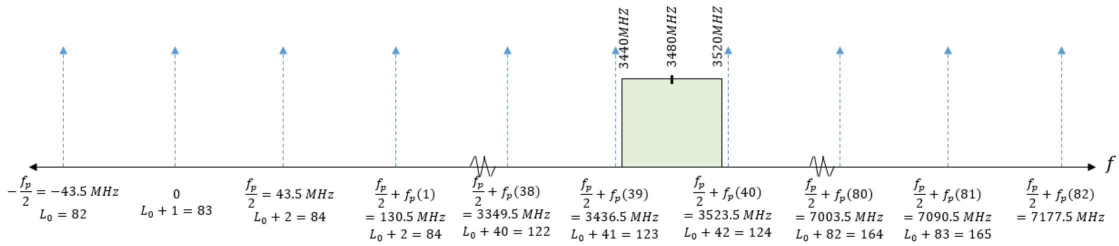


Figure 4.4: Illustration of the spectrum divided into f_p-wide slices and the placement of an input signal with $f_i = 3.48$ GHz.

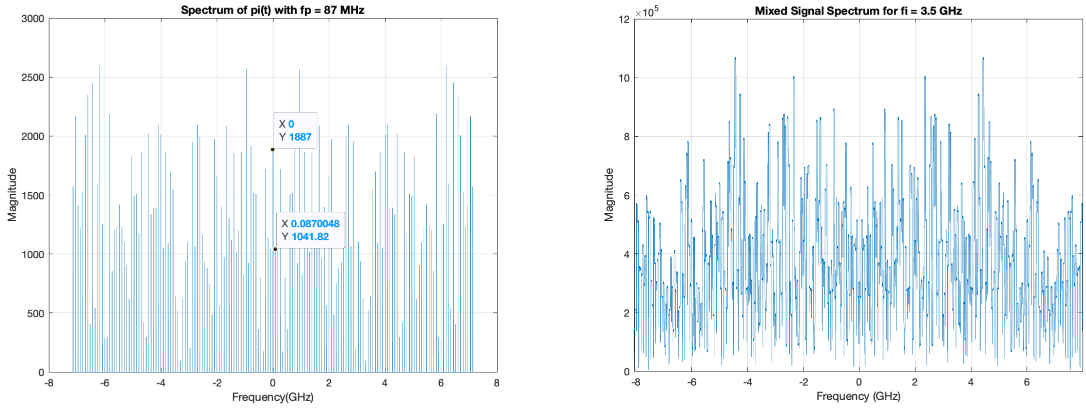
$$Supp(f_i = 3.5GHz) = [42, 43, 123, 124] \tag{4.2}$$

The indices associated with the negative frequencies are found by subtracting the indices associated with the positive frequencies from $L + 1$.

As an aside, it is important to note the edge case that occurs when a carrier frequency falls within a single f_p-wide slice. Figure 4.4 shows an example of this case for $f_i = 3.48$ GHz. The lowest index that is nearest to both the left and right edges of the signal is the same, and the support is given in (4.3).

$$Supp(f_i = 3.48GHz) = [43, 123] \tag{4.3}$$

The periodic waveforms $p_i(t)$ are generated randomly to have length M. Figure 4.5a shows the spectrum of a single $p_i(t)$, where the frequency spikes occur on integer multiples of $f_p = 87$ MHz. Next, the input signal is mixed with each periodic waveform $p_i(t)$. In Matlab, the mixing is represented by pointwise multiplication. Figure 4.5b shows the spectrum of the signal after mixing.

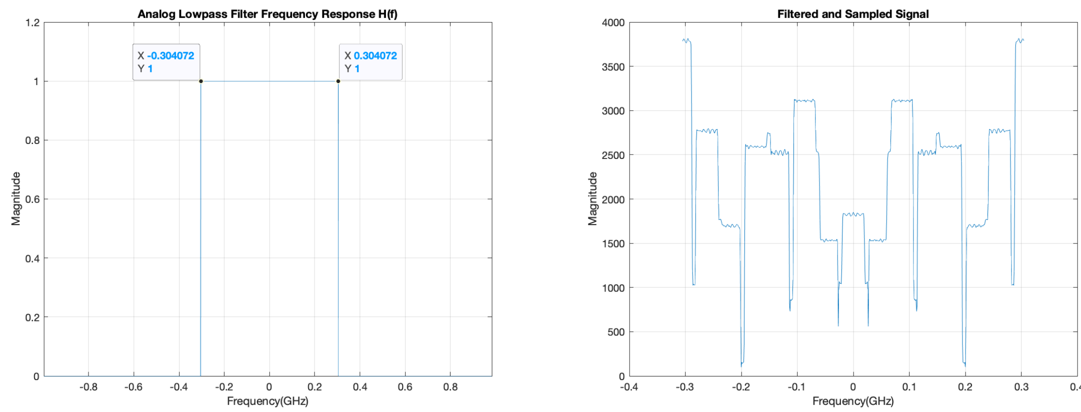


(a) Spectrum of $p_i(t)$

(b) Spectrum of the mixture $x(t)p_i(t)$.

Figure 4.5: Mixing.

After mixing, the signal is lowpass filtered and sampled. The lowpass filter has a cutoff frequency $f_s/2$, which can be seen from the frequency response in Figure 4.6a. Applying the filter to the mixed signal produces Figure 4.6b. Note that the signal is now bounded to the range $[-\mathcal{F}_s/2, \mathcal{F}_s/2]$, and the downsampling of the filtered signal produces the digital sequences $y_i[n]$. The downsampling operation concludes the portions of the simulations that imitate the stages in the analog domain. The rest of the simulation also encompasses the digital signal processing that is performed on the actual sampled sequences $y_i[n]$.



(a) Frequency response $H(f)$ of the lowpass filter.

(b) Spectrum of $y_i[n]$, the sampled sequence.

Figure 4.6: Lowpass filter frequency response and spectrum of $y_i[n]$.

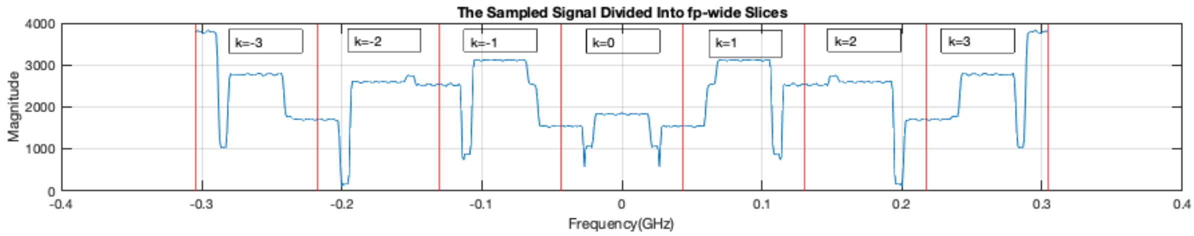
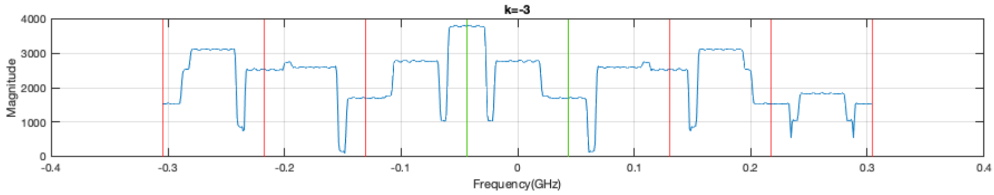


Figure 4.7: A sampled sequence divided into f_p-wide slices.

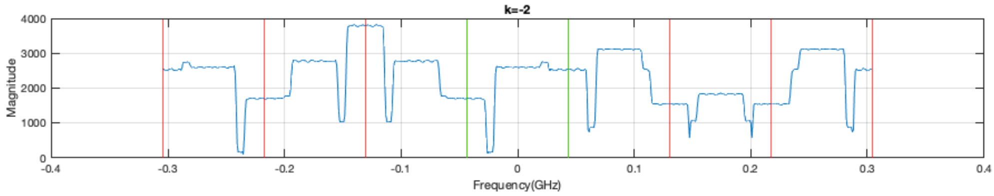
The next step is to expand the number of physical channels $m = 2$ into $mq = 14$ virtual channels via the method described in Section 3.3. Figure 4.7 shows a sampled sequence bandlimited to $[-\mathcal{F}_s/2, \mathcal{F}_s/2]$ and divided into f_p-wide slices. Each slice represents the data for one of $q = 7$ virtual channels. Each sampled sequence $y_i[n]$ is split into q virtual channels and time-modulated by $e^{-j\frac{2\pi}{q}kn}$ for $-3 \le k \le 3$, which is equivalent to shifting the spectrum. Each modulation shifts one slice of the spectrum in Figure 4.7 to baseband. Figure 4.8 demonstrates the shifting operation. When $k = 0$ in Figure 4.8d, no shift occurs. For negative values of k, the signal is right-shifted to the slice centered at the origin, and for positive values of k, the signal is left-shifted to the slice at the origin.

After time-modulation, each virtual channel is lowpass filtered and downsampled by the factor q. Figure 4.9 shows the frequency response of the digital filter with cutoff frequency $f_p/2$. Filtering and decimating each virtual channel preserves the f_p-wide slice between the green lines in Figure 4.8. Figure 4.10 shows all seven virtual channels for one physical channel. Each virtual channel has only 101 samples.

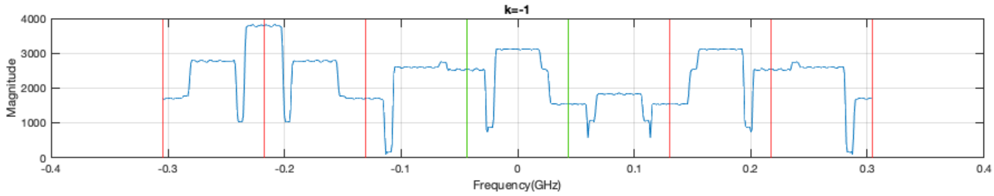
The final step of the expander is to calculate the expanded matrix $\mathbf{A}$ by performing shifts for $-3 \le k \le 3$ as shown in (4.4).



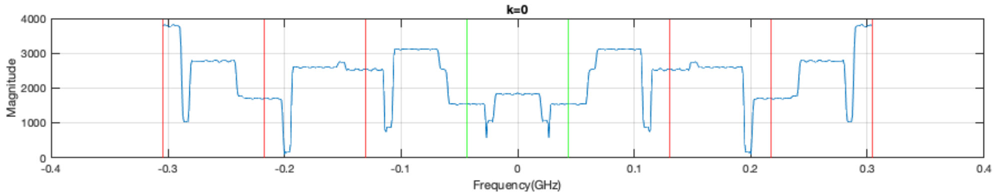
(a) The sampled sequence modulated by $k = -3$.



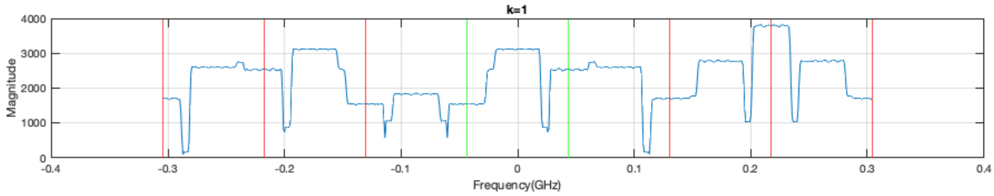
(b) The sampled sequence modulated by $k = -2$.



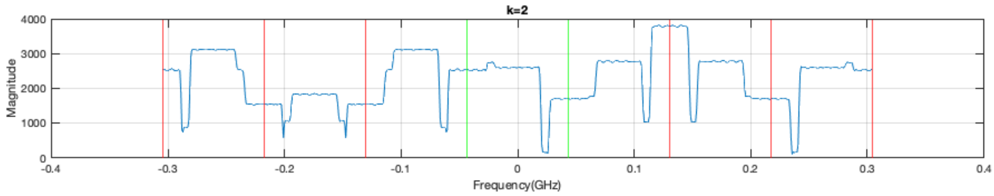
(c) The sampled sequence modulated by $k = -1$.



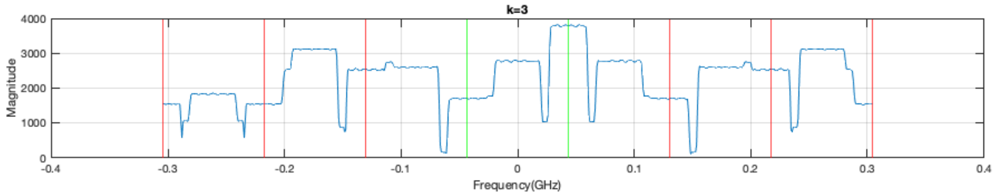
(d) The sampled sequence modulated by $k = 0$.



(e) The sampled sequence modulated by $k = 1$.



(f) The sampled sequence modulated by $k = 2$.



(g) The sampled sequence modulated by $k = 3$.

Figure 4.8: Frequency shifting of a sequence $y_i[n]$.

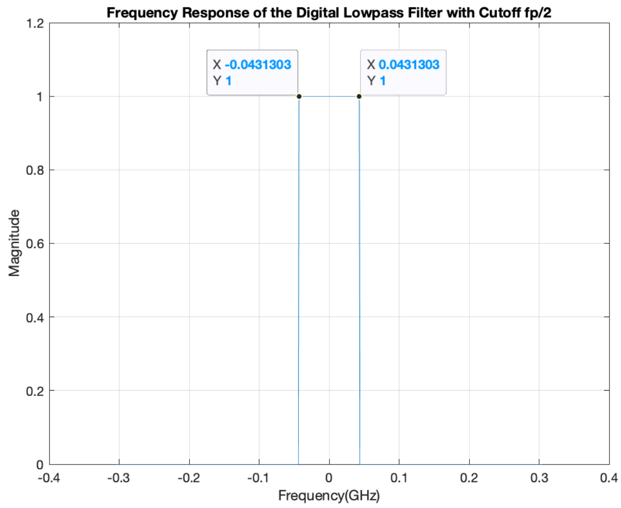
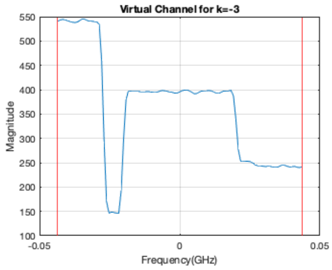


Figure 4.9: Frequency response of the digital lowpass filter.

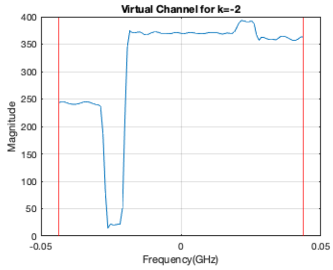
$$\mathbf{A}_{exp} = \begin{bmatrix} c_{i,79} & \cdots & c_{i,-2} & c_{i,-3} & c_{i,-4} & \cdots & c_{i,80} \\ c_{i,80} & \cdots & c_{i,-1} & c_{i,-2} & c_{i,-3} & \cdots & c_{i,81} \\ c_{i,81} & \cdots & c_{i,0} & c_{i,-1} & c_{i,-2} & \cdots & c_{i,82} \\ c_{i,82} & \cdots & c_{i,1} & c_{i,0} & c_{i,-1} & \cdots & c_{i,-82} \\ c_{i,-82} & \cdots & c_{i,2} & c_{i,1} & c_{i,0} & \cdots & c_{i,-81} \\ c_{i,-81} & \cdots & c_{i,3} & c_{i,2} & c_{i,1} & \cdots & c_{i,-80} \\ c_{i,-80} & \cdots & c_{i,4} & c_{i,3} & c_{i,2} & \cdots & c_{i,-79} \end{bmatrix} \tag{4.4}$$

Finally, the Continuous-to-Finite (CTF) block applies the SBR4 and OMP algorithms to the 14 virtual channels and expanded matrix $\mathbf{A}_{exp}$ to find the sparsest solution for the support set. Appendices C and D show the SBR4 and OMP algorithms implemented in Matlab. For this example, the support is successfully recovered as

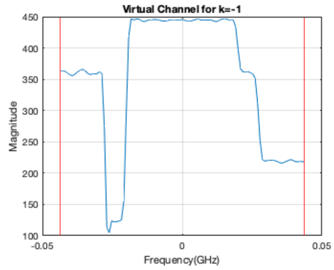
$$RecoveredSupp = \{42, 43, 123, 124\} \tag{4.5}$$



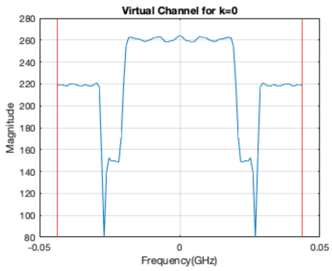
(a) The virtual sequence for $k = -3$.



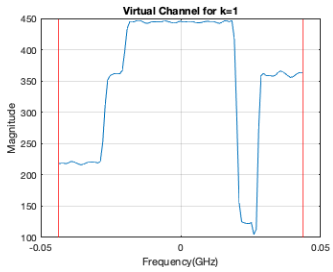
(b) The virtual sequence for $k = -2$.



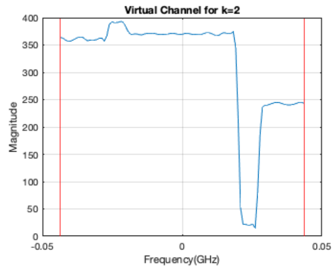
(c) The virtual sequence for $k = -1$.



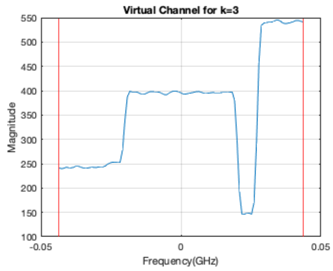
(d) The virtual sequence for $k = 0$.



(e) The virtual sequence for $k = 1$.



(f) The virtual sequence for $k = 2$.



(g) The virtual sequence for $k = 3$.

Figure 4.10: Filtering and downsampling in each virtual channel.

4.2.2 Simulation Results

The simulations performed in this section use Parameter Set 2 given in Table 4.1 unless otherwise specified. All of the cases considered find the percentage of successful support recovery over 1000 runs versus SNR, where the SNR takes values from $[-10, 30]$ dB.

First, the carrier frequency f_i was drawn randomly from the 5G FR1 range $[410MHz, 7.125GHz]$ while the periodic waveforms $p_i(t)$ were held constant. Figure 4.11 shows the percentage of successful recovery versus SNR for this case. For SNR > 5 dB, the percentage of successful recovery is greater than 90%.

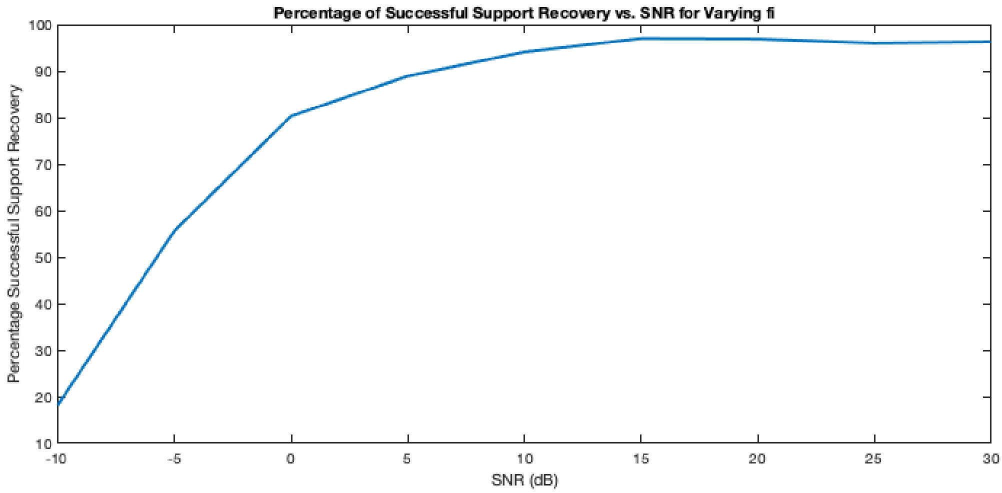


Figure 4.11: Percentage of successful support recovery vs. SNR with varying f_i.

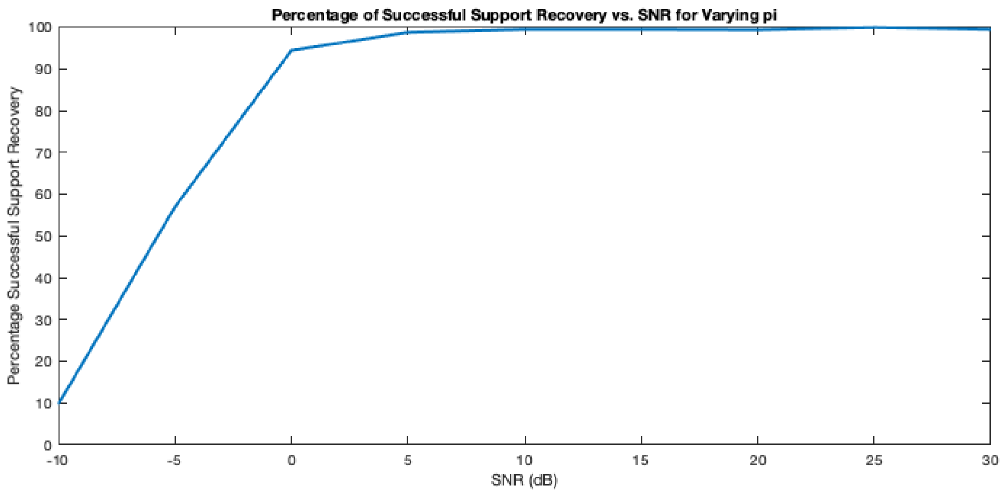


Figure 4.12: Percentage of successful support recovery vs. SNR with random $p_i(t)$.

The next simulation consists of holding f_i constant at $f_i = 3.5$ GHz and choosing new, random periodic waveforms $p_i(t)$ for each iteration. Figure 4.12 shows the percentage of successful recovery versus SNR for random $p_i(t)$. Similarly to the previous case, the percentage of successful recovery was greater than 90% for SNR > 0 dB. This experiment demonstrates the robustness of the MWC to randomly selected $p_i(t)$. This result implies the ability of a random sequence to satisfy the RIP discussed in Section 3.4.

Figure 4.13 shows the percentage of successful recovery versus SNR for odd q in the range $[1, 9]$ to examine the effect of the parameter q on MWC performance. The best performance is achieved when $q = 7$, and the worst performance occurs when $q = 1$. Table 4.2 provides

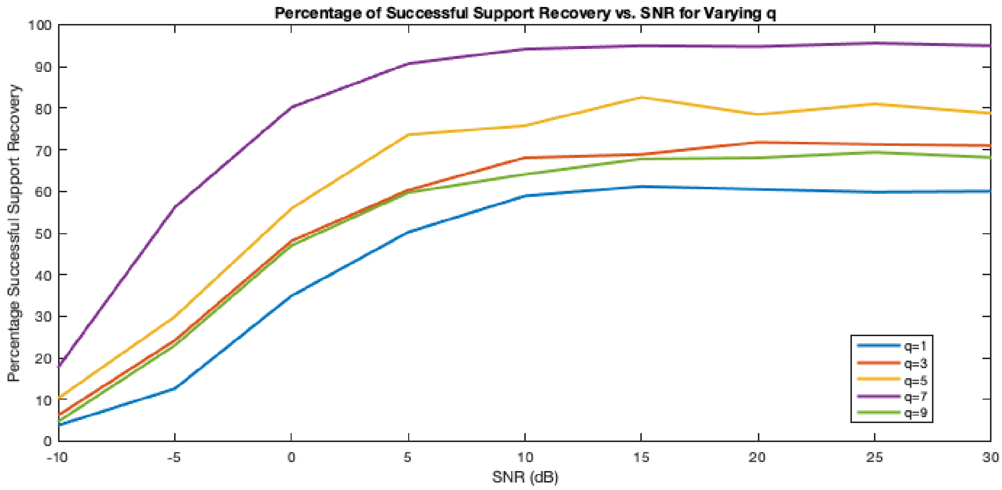


Figure 4.13: Percentage of successful support recovery vs. SNR for various q.

some insight to this result. Table 4.2 shows that the implementation using $q = 7$ has the largest number of virtual channels. The performance of the other q values can be sorted by the descending number of virtual channels. Therefore, the driving factor in successful recovery is the number of virtual channels.

Expansion Factor q	1	3	5	7	9
Physical Channels $m = \lceil \frac{4N}{q} \rceil$	8	3	2	2	1
Virtual Channels mq	8	9	10	14	9

Table 4.2: Number of physical and virtual channels for different values of q.

Figure 4.14 shows the percentage of successful recovery versus SNR for different values of B at 10 MHz intervals in the range $B = [10MHz, 80MHz]$. Overall, $B = 10$ MHz has the highest percentage of successful recovery. For SNR > 5, all of the simulated B have percentage of successful recovery greater than 90%. However, the difference is much greater for lower SNR values. It is clear for these lower SNR values that the percentage of successful recovery increases with decreasing values of B.

An observation from the results of all simulations in Figures 4.11-4.14 is that the MWC does not perform as well in low-SNR scenarios for the simulated parameters. That is because there is a tradeoff between the number of physical channels and percentage of successful recovery in [7]. Performance can be improved by increasing the number of physical channels.

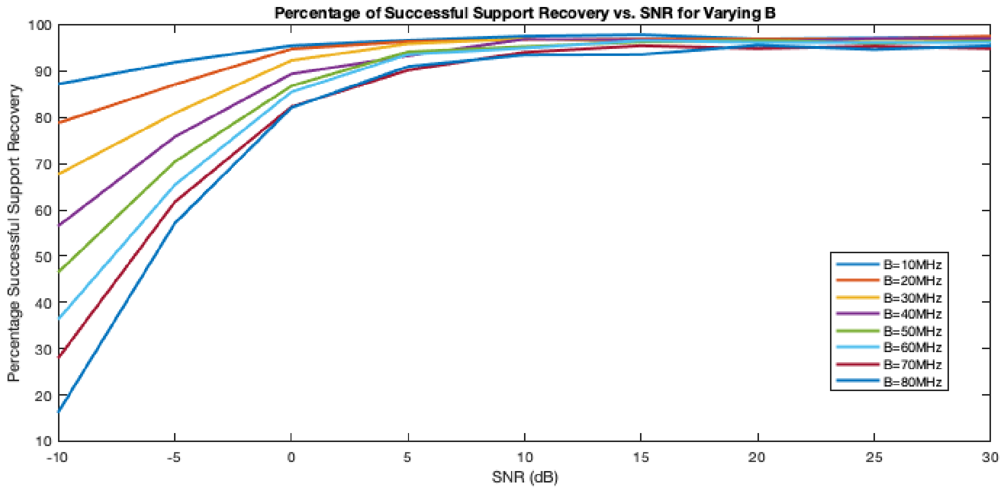


Figure 4.14: Percentage of successful support recovery vs. SNR for different values of B.

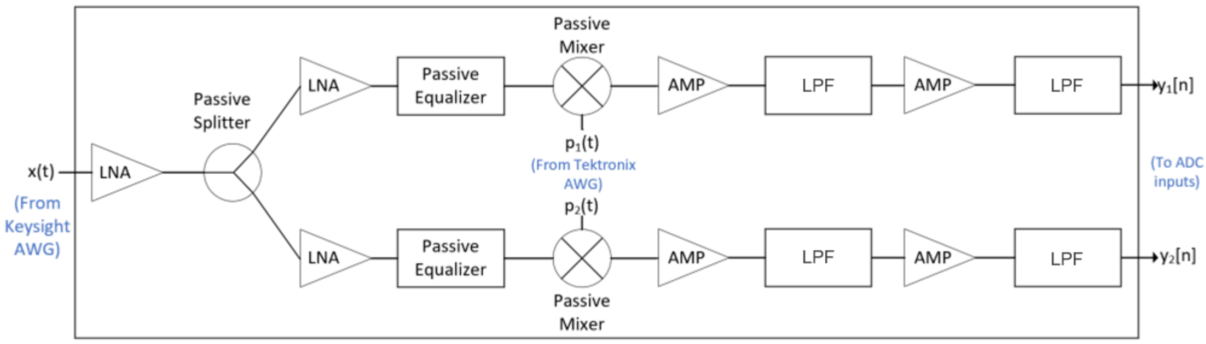


Figure 4.15: Analog front end of the MWC.

However, in order to save on cost and hardware complexity for this project, the loss in performance was deemed to be acceptable.

4.3 Hardware Design

The hardware design for the MWC consists of the amplifiers, splitter, equalizer, mixer, lowpass filter, and ADC required for the analog front-end. These devices were chosen to maximize power at the inputs of the ADC. Figure 4.15 shows the analog front-end.

4.3.1 Discussion of Hardware Choices

Some of the hardware decisions made for the MWC are nonstandard, and it is beneficial to describe the reasoning behind them. For this project, each component was either purchased

Figure 4.16: Circuit symbol for an ideal mixer.

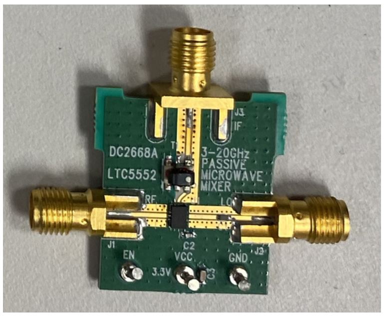


Figure 4.17: The DC2668A evaluation board with the LTC5552 mixer.

on an evaluation board or the evaluation circuit was implemented in-house. For all of the components, care was taken to find devices that operate in the desired frequency ranges for each stage.

Mixer

The mixer presents a major challenge in any MWC implementation. Figure 4.16 shows the circuit symbol for an ideal mixer. In a typical application, a signal on the RF or IF ports is multiplied by a single sinusoid to perform either upmixing or downmixing. In the MWC, however, the LO port sees the mixing functions $p_i(t)$ that alternate at approximately the Nyquist rate and are comprised of multiple sinusoids in the frequency domain. Due to the characteristics of $p_i(t)$, a passive mixer is used, because active mixers generally have a narrow input bandwidth [11]. The Analog Devices LTC5552 passive mixer on the evaluation board DC2668A in Figure 4.17 was chosen for its wideband frequency range on all three of its ports.

Equalizer

In an RF chain, signals typically suffer greater attenuation at higher frequencies. The typical application of an equalizer is to compensate for the non-flat frequency response of preceding stages. Mixing with multiple sinusoids for the MWC requires two unusual choices for the

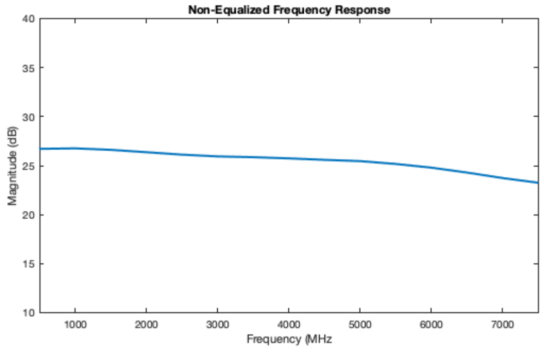


Figure 4.18: Frequency response of the AFE prior to equalization.

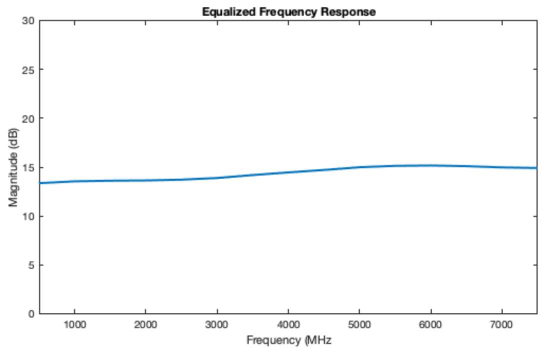


Figure 4.19: Frequency response of the AFE after equalization.

wideband equalizer. Usually, the equalizer is either placed at the end of the RF chain just before the ADC inputs or in the digital domain. Because the output of the mixer for the MWC contains energy from the entire spectrum, it would be very challenging to translate the prior stages frequency response to the mixer's output [11]. Thus, the equalizer is placed in the stage prior to the mixer, as shown in Figure 4.15. The second atypical choice for the equalizer was in selecting a passive device. Passive equalizers have large insertion loss, but an active equalizer could introduce nonlinearities that would be challenging to take into account [11].

The Mini-Circuits EQY-12-24+ used in the MWC has a 12 dB slope from DC to 20 GHz. Figure 4.18 shows an approximation of the frequency response for the devices preceding the equalizer, and Figure 4.19 shows the approximate frequency response after the equalizer. Although the response in Figure 4.19 is not completely flattened, the negative slope of the unequalized response has been eliminated after equalization.

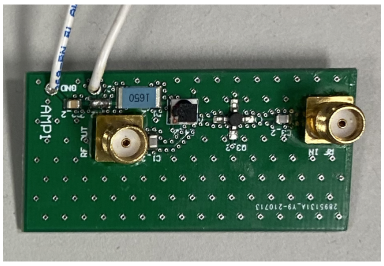


Figure 4.20: Development board for the ERA-9-SM+.

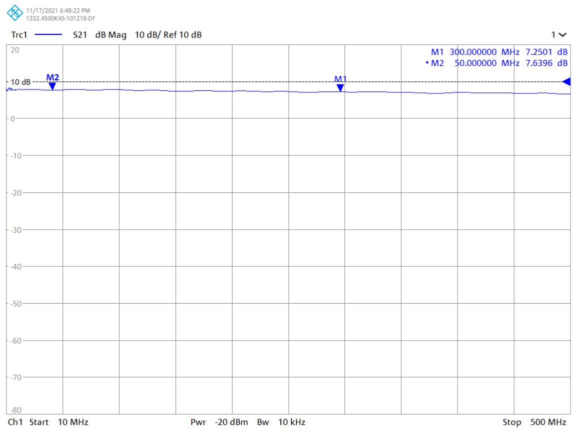


Figure 4.21: Amplifier frequency response.

Amplifier

The amplifier that precedes each lowpass filter in the chain comes after the equalizer, so it was beneficial to choose an amplifier with flat gain for the duration of the passband of the lowpass filter. The Mini-Circuits ERA-9SM in Figure 4.20 was chosen because of its flat frequency response from DC to 300 MHz. Its frequency response is shown in Figure 4.21.

Lowpass Filter

The final step in the analog front-end is lowpass filtering. Because the signal energy is spread throughout the entire spectrum after mixing [11], it is necessary for the lowpass filter to have a sharp cutoff and large stopband attenuation. An elliptic filter was chosen for the MWC due to it having the sharpest cutoff frequency of any other filter type of the same order [12]. The elliptic filter of order seven depicted in Figure 4.22 was designed for the MWC to have a cutoff frequency of approximately 364 MHz. Figure 4.23 shows the simulated frequency response of the lowpass filter. Small, modular development boards were designed using Altium Designer,

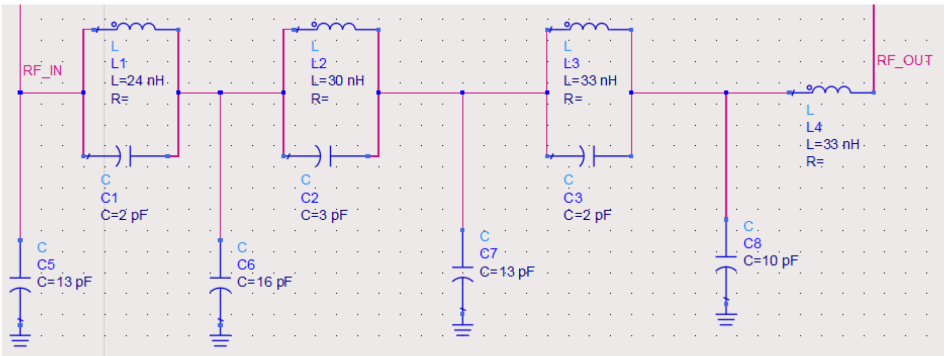


Figure 4.22: The schematic of the lowpass filter with cutoff frequency $f_c = 364MHz$.

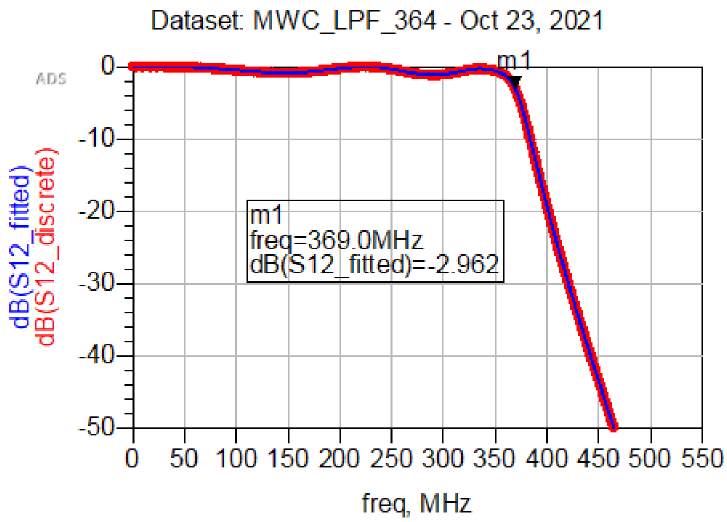


Figure 4.23: The simulated frequency response of the lowpass filter.

and then the boards and parts were ordered and assembled in-house. Figure 4.24 shows a picture of the lowpass filter.

Testing the lowpass filters showed that the frequency response did not match the simulated response in Figure 4.23. Figure 4.25 shows the measured frequency response of the lowpass filter, whose cutoff frequency is approximately 300 MHz. Although the cutoff frequency did not have the desired value, the filter has a sharp cutoff. As described in Section 4.1, the parameters for the MWC were adjusted to accommodate the actual cutoff frequency of the lowpass filter. As can be seen in Figure 4.15, the lowpass filter is applied twice to provide even greater stopband attenuation without having to design a higher order filter [11].

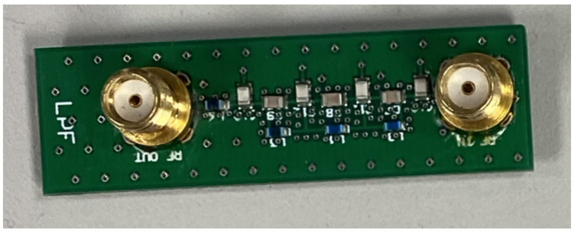


Figure 4.24: Lowpass Filter development board.

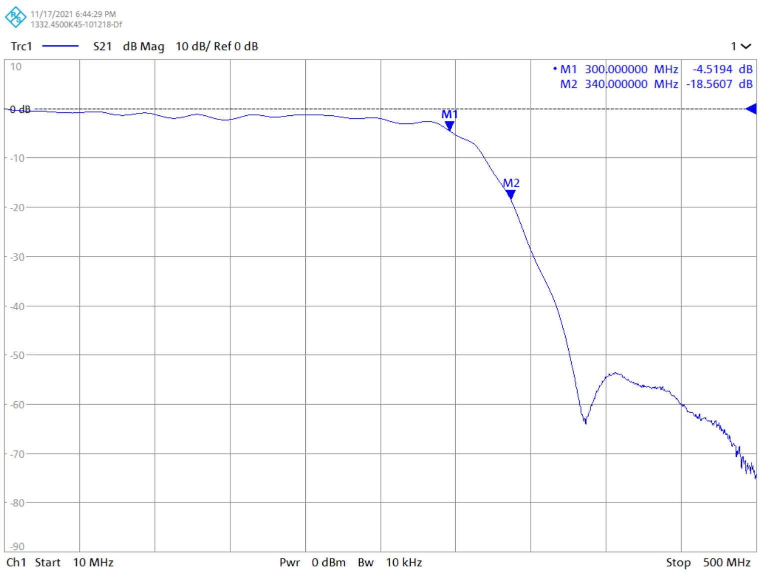


Figure 4.25: Actual frequency response of the LPF.

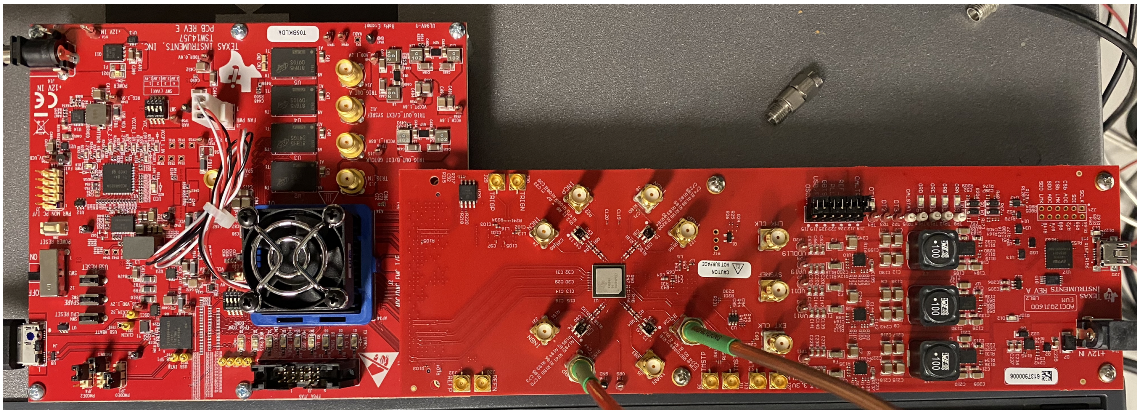


Figure 4.26: The TSW14J57EVM connected to the ADC12QJ1600EVM.

ADC

For the choice of ADC, the sampling rate, analog input bandwidth, data width, and number of analog input channels were carefully selected. The Texas Instruments (TI) ADC12QJ1600EVM is an ADC evaluation module that houses the ADC12QJ1600-Q1, a 12-bit, 1.6 GSPS quad channel ADC. It connects to the TI TSW14J57EVM data capture board via a FPGA Mezzanine Card (FMC) that handles the transfer of data from the ADC to a PC. Data from the ADC can be viewed and exported from TI's High-Speed Data Converter Pro (HSDC Pro) GUI.

The ADC was selected for its large and configurable sampling rate, which can range from 500 MSPS - 1.6 GSPS. It also has analog input bandwidth of 6 GHz, which is more than sufficient for this design. The quad channel feature allows for MWC designs that require up to four channels. Using a quad channel device means that no external synchronization is necessary to synchronize multiple ADCs. Figure 4.26 shows the ADC connected to the data capture board.

Chapter 5

Experimental Setup and Procedure

The experimental setup consists of five main sections shown in Figure 5.1. In the input signal generation block, various $x(t)$ are generated using a Keysight M8196A 92 GSa/s Arbitrary Waveform Generator. The Analog Front End (AFE) consists of the chain of amplifiers, splitter, mixers, and lowpass filters that the input signal must pass through. The Periodic Waveform Generator block provides the framework for generating $p_i(t)$ for the mixers. Each channel is sampled by the ADC and captured in CSV files on a PC. Finally, the Digital Signal Processing block reads the data captured in the CSV files and finds the successful support recovery rate for various carrier frequencies and signal bandwidths.

5.1 Input Signal Generation

The Matlab script in Appendix E creates input signals of the form given in Equation (4.1). This is the same equation used to generate the input signals in simulation. The input signals

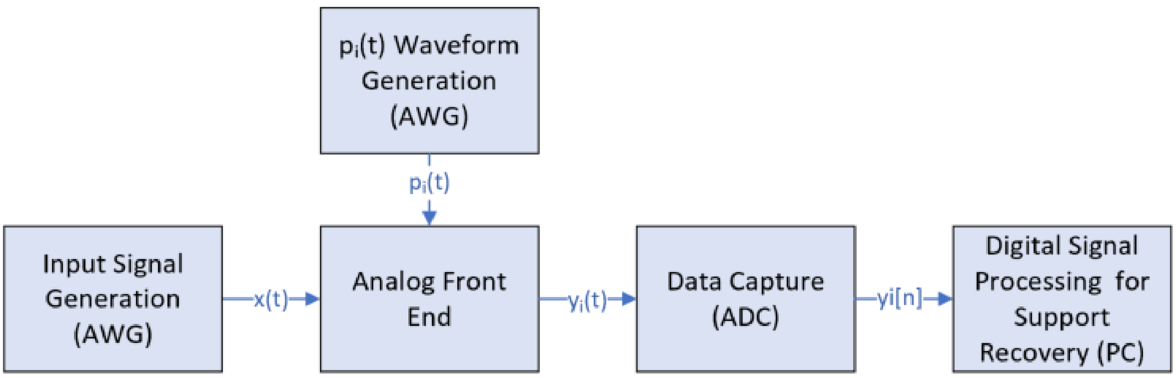


Figure 5.1: Block diagram of the hardware setup.

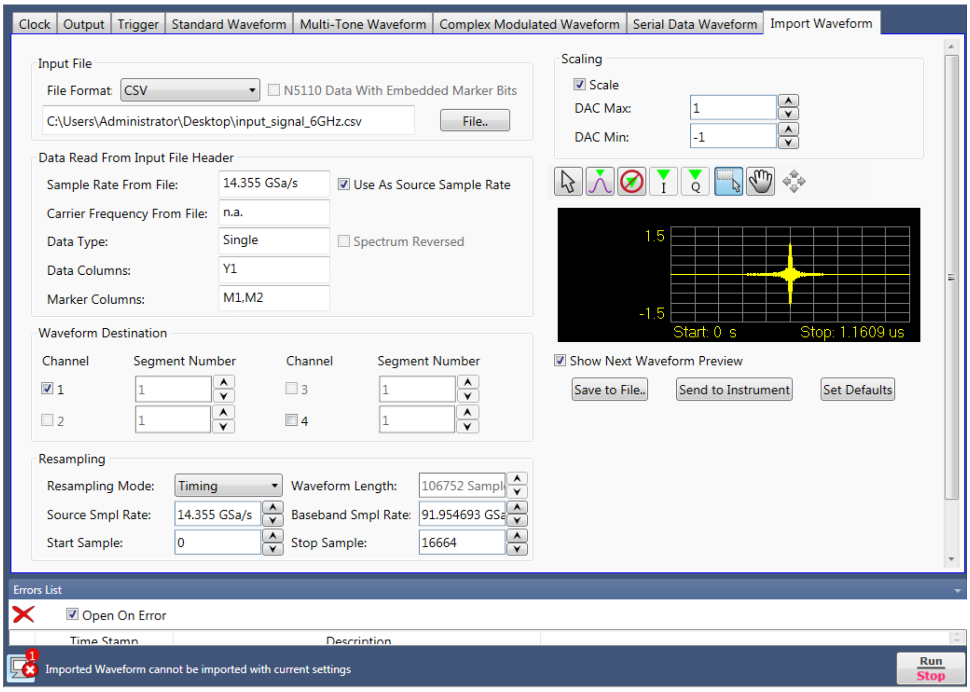


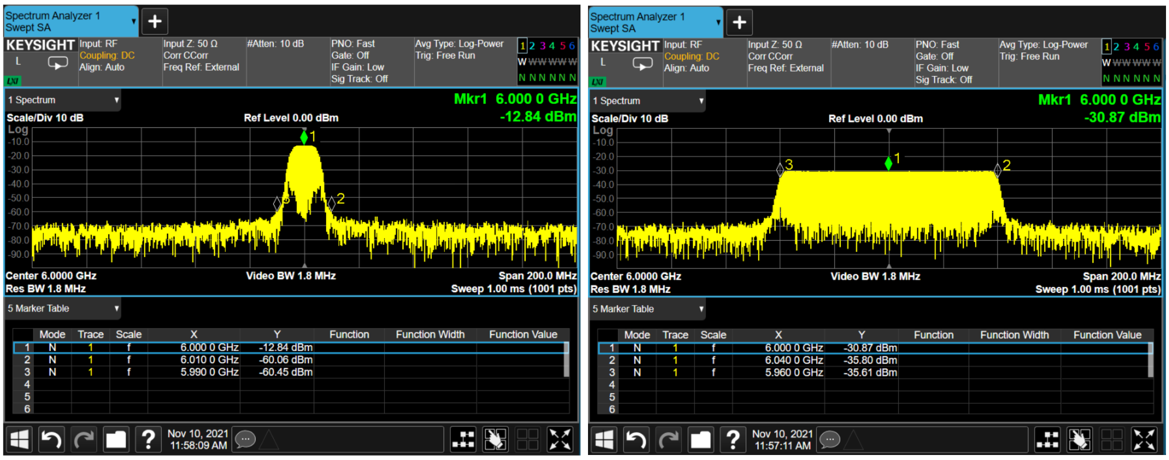
Figure 5.2: Keysight AWG Soft Front Panel with CSV file selected and sampling rate configured.

are written to a CSV file in the form that the Keysight AWG expects. For hardware testing, the carrier frequency f_i and bandwidth B are modified to produce various input signals. The sampling rate of the signal is always equal to the Nyquist rate, 14.355 GHz.

Figure 5.2 shows the Soft Front Panel (SFP) set up to import an input waveform with the aforementioned sampling rate configured. The Keysight AWG has a maximum output voltage of $1V_{pp}$. In practice, using the maximum output voltage introduced additional noise to the signal, so the output voltage was reduced to $0.8V_{pp}$ to produce a cleaner signal. Figure 5.3 shows the spectrum of two input signals at carrier frequency $f_i = 6$ GHz. In Figure 5.3a, the bandwidth is 10 MHz, but the spectrum analyzer shows that the bandwidth is slightly less than 10 MHz in practice. In Figure 5.3b, the bandwidth is 80 MHz. The power level of these signals is also a concern. Specifically, the output power of the 80 MHz bandwidth signal is approximately 18 dB lower than the 10 MHz bandwidth signal.

5.2 $p_i(t)$ Waveform Generation

The periodic waveforms $p_i(t)$ are generated using a Tektronix 70002B 25 GSa/s AWG. Appendix F shows the Matlab script used to take a CSV file containing $p_i(t)$ waveforms and write



(a) Spectrum of an input signal with $f_i = 6$ GHz and $B = 10$ MHz.

(b) Spectrum of an input signal with $f_i = 6$ GHz and $B = 80$ MHz.

Figure 5.3: Spectrum analyzer captures of different input signals.

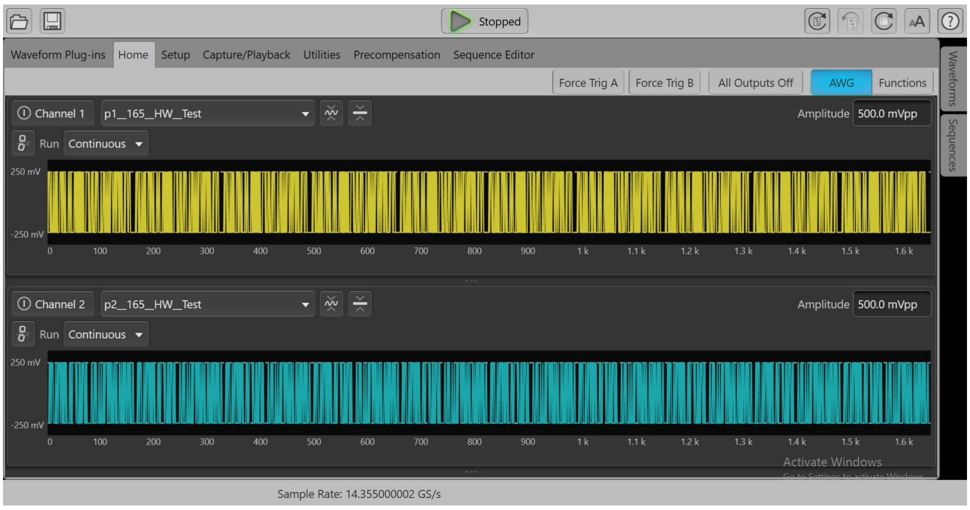
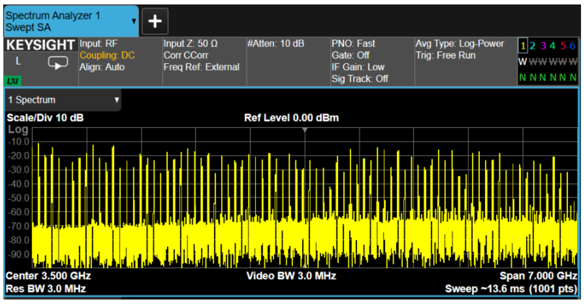


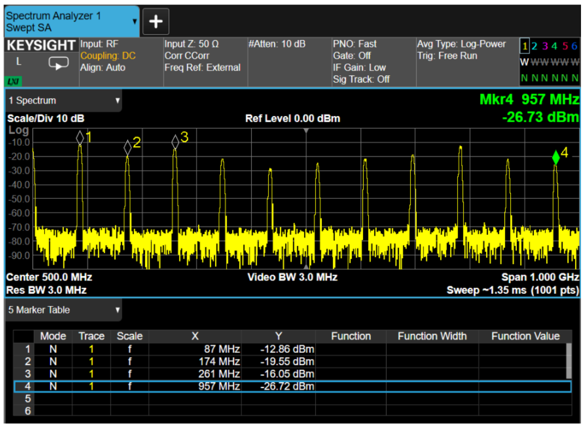
Figure 5.4: Tektronix AWG Soft Front Panel for $p_i(t)$ with sampling rate configured.

the data to a MAT file in the format that the Tektronix AWG can read. As in the previous case, the sampling rate of the Tektronix AWG is set to the Nyquist frequency, 14.355 GHz. The Tektronix AWG has a maximum output voltage of 0.5 V_{pp}, which is used for the periodic waveform generation. Figure 5.4 shows the Tektronix AWG configured to generate $p_i(t)$ for two channels at the Nyquist frequency of the system.

To demonstrate $p_i(t)$, generation, Figure 5.5a shows a spectrum analyzer capture of a single $p_i(t)$. A closer look at the spectrum and the addition of markers in Figure 5.5b verifies that the sinusoids comprising the spectrum of $p_i(t)$ always occur on integer multiples of $f_p =$ 87 MHz. Due to the maximum output voltage of the Tektronix AWG and the spreading of the energy of each $p_i(t)$ across many sinusoids in the frequency domain, the power of each individual sinusoid is not greater than -10 dBm.



(a) Spectrum analyzer capture of a single $p_i(t)$.



(b) Zoomed in spectrum analyzer capture of a single $p_i(t)$.

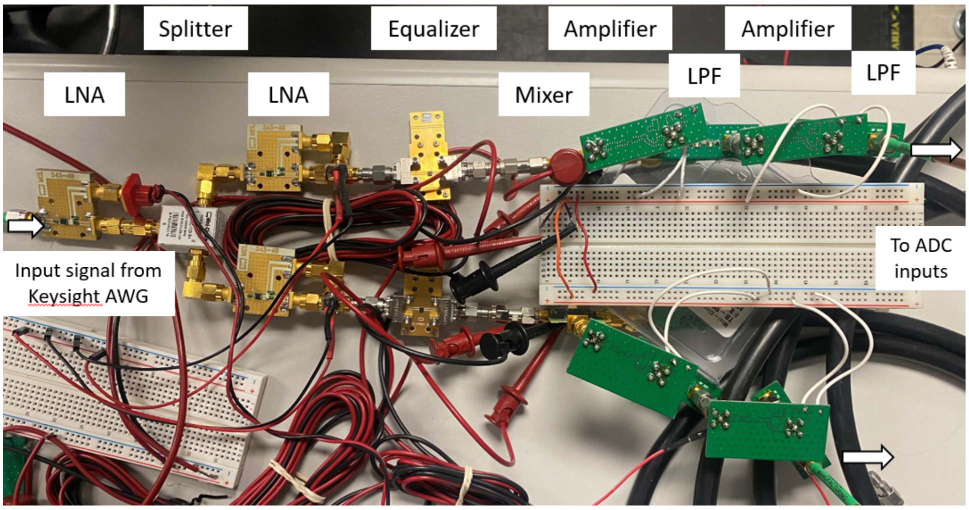
Figure 5.5: Spectrum analyzer captures of $p_i(t)$.

5.3 Analog Front End (AFE)

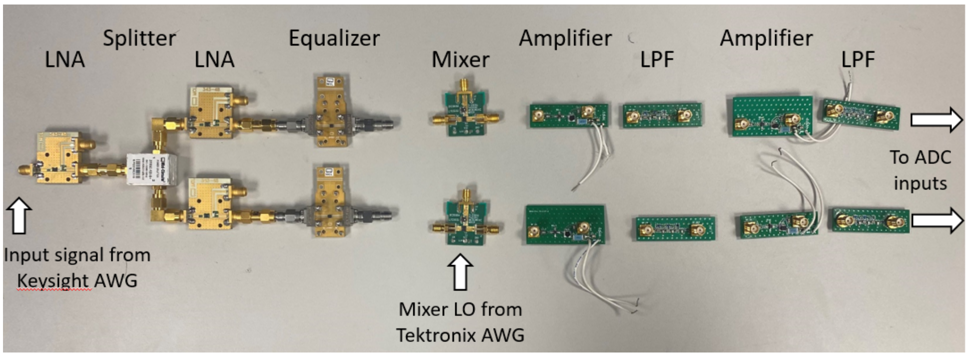
Figure 5.6a shows a top-level view of the analog front end. The input signal enters on the right from the Keysight AWG and splits into two channels before being mixed and filtered. On the left side of the figure, the output from the two channels is connected to the ADC inputs. Figure 5.6b shows an exploded view of the AFE with most of the wiring omitted for the sake of clarity. Once all of the components were connected, the power supplies were turned on to ensure that the appropriate amount of current was being drawn by each device.

To verify the behavior of the hardware setup before connecting the AFE to the ADC, a signal with $f_i = 6$ Ghz and $B = 80$ MHz was supplied at the input, and then both channels after the lowpass filters were viewed on the spectrum analyzer. Figure 5.7 provides a comparison of the output of the lowpass filters in hardware with the simulated results for each channel. Both channels are fairly accurate to the simulated versions, although Channel 1 has more differences. In both cases, it is evident that different linear combinations of the input signal $x(t)$ are present in the mixed and filtered output.

After processing in the AFE, the signals going into the ADC are very noisy. Before connecting the MWC channels to the inputs of the ADC, it is beneficial to calculate the SNR at the



(a) Top view of the analog front end.



(b) Exploded view of the analog front end .

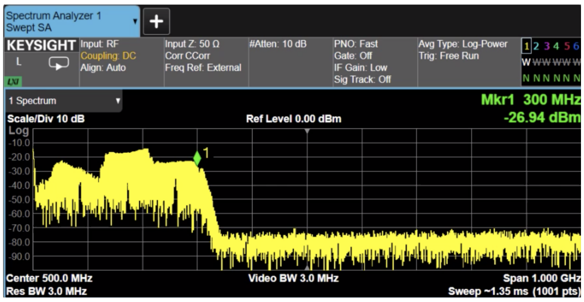
Figure 5.6: The analog front end.

ADC inputs. First, the noise figure (NF) of the AFE is calculated with the assumption that the passive splitter, equalizers, and lowpass filters do not contribute much to the noise figure and can be disregarded. The Noise Factor can be calculated for the chain of RF devices using Frii's equation in (5.1), where F_i and G_i are the noise factor and gain of each device, respectively [11]. Then the noise figure is given in

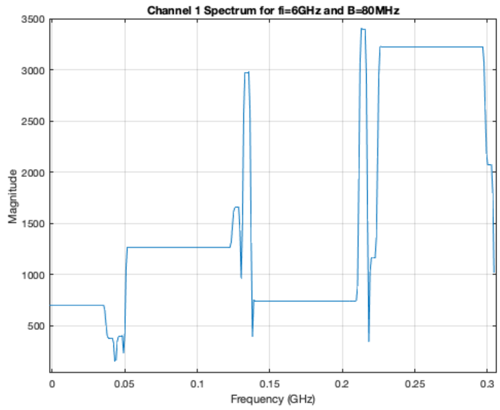
$$F = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \cdots + \frac{F_n - 1}{G_1 G_2 \cdots G_{n-1}} = 1.85 \tag{5.1}$$

$$NF = 10 \log_{10} F = 2.68dB \tag{5.2}$$

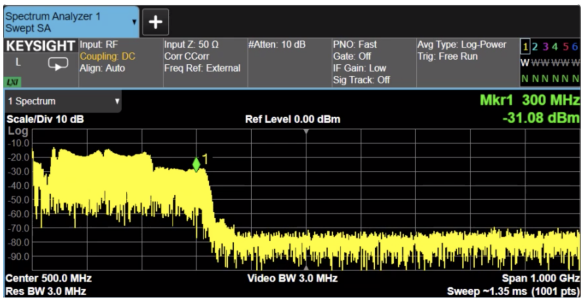
The SNR is related to the NF of the system by Equation (5.3), where all quantities are given in dB. If the input SNR is assumed to be 1 dB, then the output SNR is given by $\text{SNR}_o = -1.68$ dB. In reality, the output SNR is probably somewhat smaller than the theoretical value due to



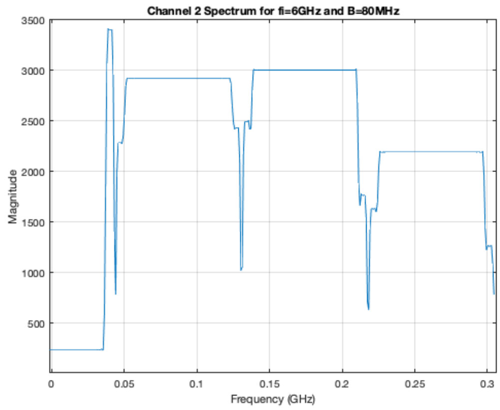
(a) Channel 1 on the spectrum analyzer after mixing and filtering.



(b) Channel 1 in simulation after mixing and filtering.



(c) Channel 2 on the spectrum analyzer after mixing and filtering.



(d) Channel 2 in simulation after mixing and filtering.

Figure 5.7: Comparison of the spectrum after mixing and filtering in hardware and software.

hardware inaccuracies. To complete the hardware setup, the output of the two MWC channels is connected to the INAP and INBP inputs of the ADC evaluation board.

$$NF = SNR_i - SNR_o \tag{5.3}$$

5.4 Data Capture

The ADC evaluation board and its data capture board capture data from AFE and transfer it to a PC for processing. This setup requires connections from both boards to a PC, connections to the ADC inputs, and a sampling clock for the ADC. The sampling clock is generated from a Keysight MXG Analog Signal Generator and is set to an output power of 5 dBm. The sampling rate for this MWC system is $f_s = 609$ MHz. However, the cutoff of the LPF is not perfect and extends beyond the theoretical cutoff $f_s/2$. Because the selected ADC can accommodate

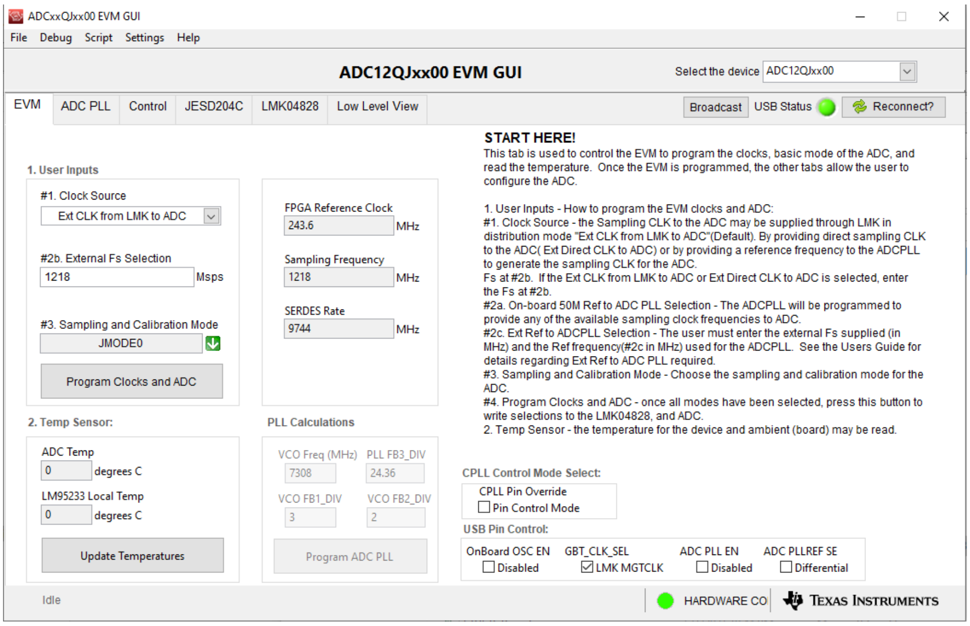
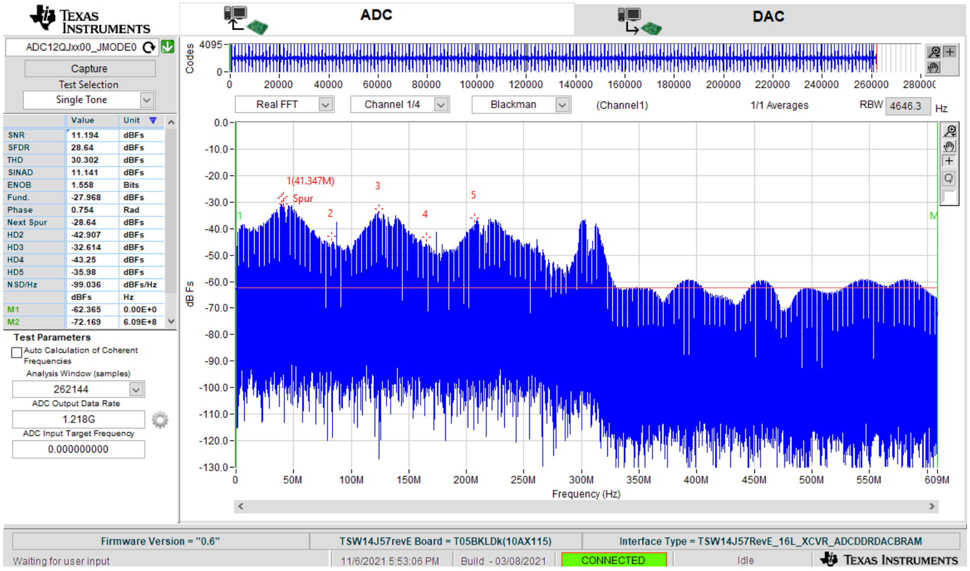


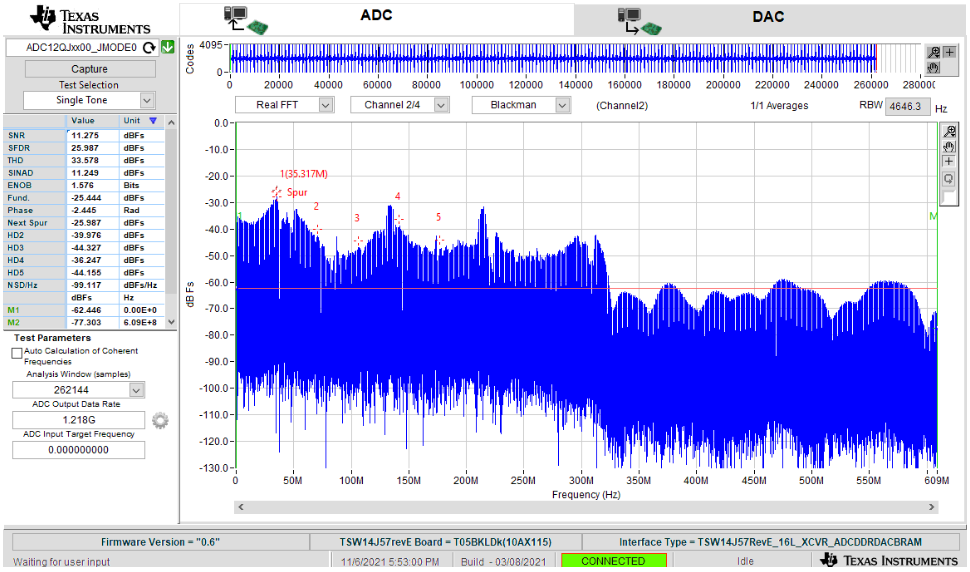
Figure 5.8: Screen capture of the ADC12QJ1600 GUI (UPDATE).

sampling rates up to 1600 MSPS, the decision was made to oversample the MWC channels by a factor of two, clean up the filter cutoff digitally, and then downsample by a factor of two. Therefore, the ADC sampling rate was configured as 1.218 GHz for all hardware tests. This must also be set in the ADC GUI provided by Texas Instruments, as shown in Figure 5.8.

Texas Instruments' software High-Speed Data Converter (HSDC) Pro allows data from the ADC evaluation model to be captured from the board and saved to a CSV file. TI also provides libraries for automating the data capture process. The Python script in Appendix G automates the process of connecting to the board, downloading the appropriate firmware to the FPGA on the data capture board, capturing data from the ADC, and saving data from the ADC as 12-bit codes to a CSV file. The script performs 50 iterations, each of which captures 262,144 samples per channel and saves them to a CSV file. It also provides an option to save screenshots of the HSDC Pro GUI. Figure 5.9 shows screenshots of the HSDC Pro GUI that display the spectrum of both channels after mixing and filtering. Note that the spectra are very similar to those in Figure 5.7.



(a) Channel 1 in HSDC Pro after mixing and filtering.



(b) Channel 2 HSDC Pro after mixing and filtering.

Figure 5.9: Screenshots of HSDC Pro for an input signal with f_i =6 Hz and $B = 80$ MHz.

5.5 Digital Signal Processing for Support Recovery

A modified version of the Matlab program used in simulation recovers the support of the signal and determines whether the recovery is accurate. It is provided in Appendix H. The simulation presented in Chapter 3.4 acquires 707 digital samples per physical channel, which will be repeated for the hardware implementation. Due to the oversampling described in Section 5.4, 1414 raw samples are required from the ADC in practice. Because of this division, each file produces 185 support recovery trials. The Python script creates 50 CSV files for every run, so it follows that there are 9,250 support recovery trials per run. These raw samples are lowpass filtered with the cutoff $f_s/2 = 304.5$ MHz and downsampled by a factor of two to arrive at 707 digital samples from each physical channel. Then, the digital samples are processed through the digital expander and CTF blocks to determine the support of the signal as described in Chapter 3.4.

Chapter 6

Results and Discussion

6.1 Data

The MWC system presented in this thesis is designed to operate on transmissions with carrier frequencies in 5G FR1 from 410 MHz - 7.125 GHz and signal bandwidths up to 80 MHz. Section 4.2.2 showed how the MWC performed for multiple carrier frequencies, signal bandwidths, and choice of periodic waveforms. The hardware experiments attempt to do the same. To that end, experiments were conducted to compare MWC performance for three different, randomly drawn $p_i(t)$ on input signals with the carrier frequencies and bandwidths shown in (6.1). The first set of periodic waveforms $p_i(t)$ was generated randomly. The other two sets were chosen such that they could successfully recover the support for all of the f_i in (6.1) at low SNR.

$$\begin{aligned} f_i &= \{0.5GHz, 2.0GHz, 3.0GHz, 3.5GHz, 4.0GHz, 5.0GHz, 6.0GHz\} \\ B &= \{10MHz, 80MHz\} \end{aligned} \tag{6.1}$$

Table 6.1 reports the percentage of successful support recovery for the various f_i in (6.1) when $B = 80$ MHz. The theoretical SNR of the system was determined to be -1.68 dB, but this is probably optimistic due to hardware inaccuracies. If the actual SNR is assumed to be closer to -5 dB, then the simulations of Chapter 4.2.2 suggest that the percentage of successful support recovery should be approximately 55%.

	0.5 GHz			2 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	38	9250	0.41%	885	9250	9.57%
Pattern 2	537	9250	5.81%	1344	9250	14.53%
Pattern 3	516	9250	5.58%	1235	9250	13.35%
	3 GHz			3.5 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	0	9250	0.00%	277	9250	2.99%
Pattern 2	168	9250	1.82%	541	9250	5.85%
Pattern 3	314	9250	3.39%	839	9250	9.07%
	4 GHz			5 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	1824	9250	19.72%	0	9250	0.00%
Pattern 2	1338	9250	14.46%	167	9250	1.81%
Pattern 3	961	9250	10.39%	2	9250	0.02%
	6.0 GHz					
	Successes	Total	Percentage			
Pattern 1	1190	9250	12.86%			
Pattern 2	1247	9250	13.48%			
Pattern 3	1441	9250	15.58%			

Table 6.1: Percentage of successful support recovery for various values of f_i when $B = 80$ MHz, and the equalizer is included in the AFE.

The results presented in Table 6.1 do not come close to the simulated percentage. In particular, the MWC struggled to successfully recover the support of the signal for $f_i = \{0.5GHz, 3.0GHz, 3.5GHz, 5.0GHz\}$. Chapters 5.1 and 5.2 note that the power level of the input signal and the periodic waveforms $p_i(t)$ are limited by the maximum V_{pp} of the AWGs used for testing. It was possible that increasing the power level at the inputs of the ADC could improve performance. Since the power level of the inputs could not be increased without additional hardware, the equalizer was removed from the AFE. The equalizer has a conversion loss of 13 dB at its lowest frequencies, and the thought was that the additional power saved in removing it from the AFE would outweigh the benefits of equalizing the frequency response of the AFE. With that in mind, the equalizer was removed from the AFE for the rest of the hardware experiments.

Table 6.2 reports the percentage of successful support recovery for the f_i in (6.1) and $B = 80$ MHz. The three $p_i(t)$ are the same for both sets of data. This data is a definite improvement over Table 6.1. With only one exception, the percentage of successful support recovery has either improved-in some cases dramatically-or remained comparable. The only exception occurs at 3.5 GHz for Pattern 1, which has a successful support percentage with the equalizer at 2.99% compared to 0.04% when the equalizer was removed. Since this only occurs once and the performance is not vastly improved, it could probably be considered a statistical outlier.

	0.5 GHz			2 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	681	9250	7.36%	1418	9250	15.33%
Pattern 2	795	9250	8.59%	1129	9250	12.21%
Pattern 3	537	9250	5.81%	4003	9250	43.28%
	3 GHz			3.5 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	1244	9250	13.45%	4	9250	0.04%
Pattern 2	921	9250	9.96%	1176	9250	12.71%
Pattern 3	265	9250	2.86%	927	9250	10.02%
	4 GHz			5 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	1703	9250	18.41%	0	9250	0.00%
Pattern 2	1754	9250	18.96%	818	9250	8.84%
Pattern 3	1454	9250	15.72%	156	9250	1.69%
	6.0 GHz					
	Successes	Total	Percentage			
Pattern 1	2247	9250	24.29%			
Pattern 2	1773	9250	19.17%			
Pattern 3	2109	9250	22.80%			

Table 6.2: Percentage of successful support recovery for various values of f_i when $B = 80$ MHz, and the equalizer is not included in the AFE.

Table 6.3 reports the percentage of successful support recovery for the f_i in (6.1) and $B = 10$ MHz. The three $p_i(t)$ are the same as used previously. From Chapter 4.2.2, the theoretical percentage of successful support recovery for -5 dB SNR at $B = 10$ MHz is approximately 92%. Although the results presented in Table 6.3 do not approach that success rate, they are generally an improvement over the the results in Table 6.2 for $B = 80$ MHz. Significant improvements are evident for the f_i values 0.5 GHz, 2 GHz Patterns 1-2, 3 GHz Pattern 3, 3.5 GHz, and 5 GHz Pattern 1. More modest improvements occur for f_i values 4 GHz Patterns 1 and 3, 5 GHz Pattern 2, and 6 GHz. In a couple of cases, the performance at $B = 10$ MHz is less than that at $B = 80$ MHz.

	0.5 GHz			2 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	1377	9250	14.89%	3410	9250	36.86%
Pattern 2	1295	9250	14.00%	3377	9250	36.51%
Pattern 3	2684	9250	29.02%	1597	9250	17.26%
	3 GHz			3.5 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	621	9250	6.71%	3002	9250	32.45%
Pattern 2	555	9250	6.00%	3042	9250	32.89%
Pattern 3	1353	9250	14.63%	2078	9250	22.46%
	4 GHz			5 GHz		
	Successes	Total	Percentage	Successes	Total	Percentage
Pattern 1	2011	9250	21.74%	0	9250	0.00%
Pattern 2	1125	9250	12.16%	1109	9250	11.99%
Pattern 3	1644	9250	17.77%	851	9250	9.20%
	6.0 GHz					
	Successes	Total	Percentage			
Pattern 1	3040	9250	32.86%			
Pattern 2	2377	9250	25.70%			
Pattern 3	3064	9250	33.12%			

Table 6.3: Percentage of successful support recovery for various values of f_i when $B = 10$ MHz, and the equalizer is not included in the AFE.

The lowest percentage of successful recovery occurs for 0.5 GHz, 3.5 GHz, and 5 GHz when $B = 80$ MHz and 3 GHz and 5 GHz when $B = 10$ MHz. To gain some insight in this phenomenon, it was necessary to perform simulations for each combination of f_i and $p_i(t)$ with a low SNR. In this scenario, all of the parameters are held constant, and only the noise changes randomly on each iteration. Table 6.4 shows the number of successful support recoveries over 1000 trials when $B = 80$ MHz, and Table 6.5 shows the number of successful support recoveries over 1000 trials when $B = 10$ MHz.

In Table 6.4, consider the data for 3.5 GHz and 5 GHz. These have a low percentage of successful support recovery in a low-SNR scenario in simulation, and this is further illustrated by the results for those frequencies in Table 6.2. Likewise, in Table 6.5, the simulation results for 3 GHz and 5 GHz is indicative of the poor performance seen in the hardware implementation at those frequencies. Looking at the number of successful recoveries of a given $p_i(t)$ for different input signal carrier frequencies appears to be an indicator of the failure of that pattern at those frequencies in the hardware implementation. However, it does not always seem to be as accurate at predicting the success a pattern will have in the hardware implementation.

	0.5 GHz	2 GHz	3 GHz	3.5 GHz	4 GHz	5 GHz	6 GHz
Pattern 1	135	941	390	32	213	29	276
Pattern 2	309	489	512	144	997	170	997
Pattern 3	252	799	590	97	999	537	977

Table 6.4: Successful support recovery over 1000 runs for each f_i and $p_i(t)$ when $B = 80$ MHz.

	0.5 GHz	2 GHz	3 GHz	3.5 GHz	4 GHz	5 GHz	6 GHz
Pattern 1	997	985	252	999	962	8	992
Pattern 2	996	763	542	998	998	127	1000
Pattern 3	1000	897	436	995	1000	177	998

Table 6.5: Successful support recovery over 1000 runs for each f_i and $p_i(t)$ when $B = 10$ MHz.

There are two main issues to address with regards to the MWC performance: its overall decreased performance compared to simulations and its decreased performance at some frequencies. The overall decreased performance compared to the simulations can likely be attributed to issues in the AFE. Chapters 5.1 and 5.2 make note of the output power limitation of the AWGs used to generate these signals, and the beginning of this section demonstrated that removing the equalizer (and its associated conversion loss) from the AFE improved the performance of the MWC. It is possible that increasing the power level at the ADC inputs could lead to further improvements in MWC performance. The mixer may also contribute to the loss in performance. The LTC5552 mixer's LO frequency range spans 1 to 20 GHz, and its RF frequency range spans 3 to 20 GHz. The LTC5552 mixer was selected for its performance at high frequencies, but the signals seen on the LO and RF ports may have information at lower frequencies as well.

The results presented in this section demonstrate that the MWC has decreased performance at some frequencies compared to others, particularly in a low-SNR scenario. This suggests that the use of random $p_i(t)$ is not necessarily robust for all possible frequencies that the MWC is designed to operate. A more exhaustive search to find a more successful choice of $p_i(t)$ may be necessary to improve performance across all possible frequencies.

6.2 Future Work

There are many improvements that could be made to the MWC implementation presented in this thesis, some of which have already been alluded to in this work. Section 5.2 revealed that the power level of the periodic waveforms $p_i(t)$ are currently limited by the maximum V_{pp} of the Tektronix AWG. Future iterations should look into amplifying the power of these $p_i(t)$. A similar issue is encountered with the input signal $x(t)$. The same solution could be applied to the input signal, but modifying the AFE to be more robust to a wider range of input power levels could be a more practical option.

Section 6.1 suggested that the frequency ranges on the LO and RF ports of the mixer could be negatively impacting the performance of the MWC, and the mixer is one of the most important components in the MWC. The MWC requires nonstandard use of its mixers, because the RF input is mixed with the multiple sinusoids found in each $p_i(t)$ instead of a single sinusoid [11]. As opposed to upmixing or downmixing a smaller subset of frequencies, the LO and RF ports of the mixer must instead operate over a wide frequency range. Although it is challenging to find a mixer that satisfies the specifications required by this implementation of the MWC, future versions should consider trying other mixers that better match the frequency range for this implementation.

Another possible performance bottleneck is the choice of random $p_i(t)$. Section 6.1 demonstrated that some choices of random $p_i(t)$ perform better than others, particularly in low-SNR scenarios. A more exhaustive search for an appropriate $p_i(t)$ could improve performance across all frequencies. One such search could involve generating random $p_i(t)$ and testing it for multiple frequencies throughout the MWC operating range. Only those sequences that had a high success rate across all frequencies would be used for the MWC. Another avenue of investigation would be to consider using Maximal or Gold sequences, because they also have high probability of satisfying the RIP and being suitable for the MWC [13].

For use as a testbed, generating the $p_i(t)$ from an AWG is an acceptable choice. However, using an AWG is not practical in most real-world scenarios. Future versions of the MWC should look into generating the $p_i(t)$ from a high-rate shift register or an FPGA [7].

The MWC performance could also be improved by calibrating the matrix $\mathbf{A}$. Although the relationship between the original input signal and low-rate sampled sequences is well-defined by the matrix $\mathbf{A}$ in simulation, it may be less effective in a hardware implementation due to the non-ideal behavior present in the analog components [14]. The calibration algorithm presented in [14] proposes to compensate for the non-ideal behavior of the components in the AFE to improve performance.

Although originally this MWC implementation was designed to sense up to two simultaneous transmissions, issues with some of the hardware components initiated a design change that reduced the number of input transmissions to one. Future iterations of the MWC would benefit from being modified to sense multiple transmissions simultaneously. This would likely involve additional physical channels. The modular design of this implementation of this MWC lends itself to the addition of physical channels for further testing. Because the individual components in the AFE are on their own development boards, adding additional components will not require modifications to the portions of the design that are already present. The modular design also allows different parts to be substituted into the AFE to determine their performance. However, a practical implementation of the MWC would benefit from being integrated on a single board to save on physical size.

6.3 Conclusion

CR technologies seek to promote spectrum sharing through opportunistic spectrum access in an effort to mitigate the spectrum scarcity problem [1]. The fixed frequency allocation scheme creates temporal and spatial spectrum holes, because the primary user of a band may be using its allocated spectrum infrequently [1]. Thus, opportunistic spectrum access provides a means of increasing the efficiency of spectrum use. In order to take advantage of spectrum holes, it is necessary to identify, characterize, and geolocate nearby transmitters.

This thesis presented a design and hardware implementation of a spectrum awareness testbed intended as a proof of concept for spectrum sensing in the 5G FR1 range (400 MHz-7.125 GHz) that does not rely on $a\ priori$ information. It is designed to detect one transmission that has a maximum bandwidth of 80 MHz. The targeted frequency range encompasses several Gigahertz, which classifies it as a wideband spectrum sensing problem. One of the most substantial challenges in wideband spectrum sensing is the sampling rate determined by the Nyquist rate. Sampling at the Nyquist rate is generally infeasible for a wide frequency range, but compressive sensing provides an avenue for reducing the sampling rate.

The MWC is a hardware implementation of a compressive sensing system and is the design chosen for the testbed in this thesis. It splits the input signal into multiple channels, each of which is mixed with a high-rate alternating sign pattern. The result of mixing is that the signal is intentionally aliased such that different linear combinations of the original signal appear at baseband. After lowpass filtering, the channels can be sampled with a low-rate ADC. The alternating sign patterns relate the original input signal to the low-rate samples, allowing the support of the original signal to be recovered using compressive sensing.

The MWC with the specified parameters was simulated in Matlab before constructing a hardware implementation. Simulations found that the MWC could achieve a successful support recovery percentage in excess of 90% when $B = 80$ MHz in high-SNR scenarios. In general, the percentage of successful support recovery increases with decreasing signal bandwidth. The performance difference is mostly noticeable in low-SNR scenarios, and the disparity becomes negligible for larger SNRs.

A hardware version of the MWC with the specified parameters was implemented as a modular design and tested for multiple carrier frequencies, bandwidths and sign patterns $p_i(t)$. It has an estimated SNR of -5 dB. When $B = 80$ MHz the best percentage of successful support recovery is 43.28% and occurs when $f_i = 2.0$ GHz. When $B = 10$ MHz, the highest percentage of successful support recovery is 36.86% and occurs when $f_i = 2.0$ GHz. However, the MWC had overall better performance across the tested input signal carrier frequencies when $B = 10$ MHz compared to $B = 80$ MHz.

The performance of the MWC hardware implementation does not meet the performance of the simulations. The degradation of performance is likely caused by hardware issues such as low input signal power levels, the frequency range of the chosen mixer, and general analog component inaccuracies that cause the relationship between the input signal and the low-rate samples to be inaccurate compared to the theoretical value. Section 6.2 suggests avenues for improvements to the MWC implementation that will hopefully increase performance. Despite its lower success rate, this implementation of the MWC does succeed as a proof of concept for wideband spectrum sensing in the 5G FR1 range, and the flexibility of the MWC design allows it to be extended for additional capabilities.

References

[1] R. Umar and A. Sheikh, "A comparative study of spectrum awareness techniques for cognitive radio oriented wireless networks," *Elsevier Physical Communications*, vol. 9, 08 2012.

[2] Y. Arjoune and N. Kaabouch, "A comprehensive survey on spectrum sensing in cognitive radio networks: Recent advances, new challenges, and future research directions," *Sensors*, vol. 19, no. 1, 2019.

[3] B. I. Ahmad, "A survey of wideband spectrum sensing algorithms for cognitive radio networks and sub-nyquist approaches," 2020.

[4] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[5] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[6] J. A. Tropp, J. N. Laska, M. F. Duarte, J. K. Romberg, and R. G. Baraniuk, "Beyond nyquist: Efficient sampling of sparse bandlimited signals," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 520–544, 2010.

[7] M. Mishali and Y. C. Eldar, "From theory to practice: Sub-nyquist sampling of sparse wideband analog signals," *CoRR*, vol. abs/0902.4291, 2009.

[8] M. Mishali and Y. C. Eldar, "Blind multiband signal reconstruction: Compressed sensing for analog signals," *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 993–1009, 2009.

[9] Y. C. Eldar, *Sampling Theory: Beyond Bandlimited Systems*. USA: Cambridge University Press, 1st ed., 2015.

[10] R. Baraniuk, M. A. Davenport, R. A. DeVore, and M. B. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constructive Approximation*, vol. 28, pp. 253–263, 2008.

[11] M. Mishali, Y. C. Eldar, O. Dounaevsky, and E. Shoshan, "Xampling: Analog to digital at sub-nyquist rates," 2009.

[12] P. Podder, M. M. Hasan, M. R. Islam, and M. Sayeed, "Design and implementation of butterworth, chebyshev-i and elliptic filter for speech signal analysis.," *arXiv: Signal Processing*, 2020.

[13] M. Mishali and Y. C. Eldar, "Expected rip: Conditioning of the modulated wideband converter," 2009.

[14] E. Israeli, S. Tsiper, D. Cohen, E. Shoshan, R. Hilgendorf, A. Reysenson, and Y. C. Eldar, "Hardware calibration of the modulated wideband converter," in *2014 IEEE Global Communications Conference*, pp. 948–953, 2014.

Appendices

Appendix A

MWC System with Iterations

breaklines

```
% Simulation modeling operation of the Modulated Wideband Converter (MWC)
% The necessary system parameters are defined at the top, and the rest
% of the simulation is divided into the following sections:
%   - Signal Generation
%   - Mixing
%   - Analog Lowpass Filtering and Sampling
%   - Digital Expander
%   - Continuous to Finite (CTF) Block
% This program is currently configured to test different SNR values
% but can be modified to evaluate performance of other parameter
% options, such as q or B.
successes = 0;
successes_table = zeros(1,9);
r = 1;
pit = readmatrix('SuccessPattern_HW_Test1_m2.csv');
for snr=-10:5:25
    successes = 0;
for z=1:1000
    %% Signal Parameters

    N = 2;                          % Number of bands (including conjugates)
    B = 80e6;                       % BW of each band
    fmin = 410e6;                   % Minimum carrier frequency
    fmax = 7.125e9;                 % Maximum carrier frequency
    fnyq = 14.355e9;                % Nyquist frequency
    Tnyq = 1/fnyq;                  % Nyquist period
    Ei   = [1 2];                   % Energy of the ith band

    q = 7;                          % Expansion factor (must be odd)

    len  = 91;                      % Length of the signal
    zp   = 10;                      % Save room for zero-padding
    L    = 165;                     % Aliasing rate
    tRes = Tnyq/q;                  % Time axis resolution

    obsPeriod = [0  L*q*len-1 L*q*(len+zp)-1]*tRes;% Observation window
    Taui = [0.4  0.7]*max(obsPeriod);             % Offset of the ith band

    %% Sampling Parameters

    M  = L;                         % Length of the sign patterns,
```

```
                                % set to M=L for simplicity
fp = fnyq/L;                    % Frequency of the sign patterns
fs = q*fp;                      % Sampling rate at each channel,
                                % use fs=qfp, with odd q
m  = ceil((4*N)/q);             % Number of channels

L0 = floor(M/2);                % L = 2L0 + 1

% Generate random sign patterns for pi(t)
pit = randsrc(m,M);

%% Generate Input Signal x(t)

timeAxis      = obsPeriod(1) : tRes : obsPeriod(end); % Time axis
timeAxisShort = obsPeriod(1) : tRes : obsPeriod(2);   % Without padding

% Choose random carriers from [fmin, fmax]
fi = (fmax-fmin).*rand(1,N/2) + fmin;

% Calculate x
n = 1:(N/2);
x = sum(sqrt(Ei(n)') * sqrt(B).*sinc(B*(timeAxisShort-Taui(n)')) .* ...
    cos(2*pi*fi(n)'.*(timeAxisShort-Taui(n)')),1);

% Add a Hann window for smoothing
hannWin = hann(length(x))';         % Hann window
x = [x.*hannWin, zeros(1,q*zp*L)];  % Zero padding

% Calculate the original support set
supp = [];
for i=1:(N/2)
    supp = [supp indices_calc(L0, fi(i), fp, B)];
end
supp = sort(unique(supp));

%% Mixing

% Mix the input signal x with the sign pattern of each channel.
% For q > 1, ensure that the sign pattern is expanded appropriately
ch = 1:m;
repeat = length(x)/(M*q);

% First term:  Split into m channels
% Second term: Expand SignPatterns for q > 1, then expand to fill
%              the full length of x
mixedChannels = repmat(x, m, 1) .* ...
                repmat(repelem(pit(ch,:), 1, q), 1, repeat);

%% Analog Lowpass Filtering and Sampling

% Mock of analog LPF with cutoff fs/2
w = (fs/2/fnyq/q)*2;
k = (w*pi*length(timeAxis))/(2*pi);
wp = floor(k);
H = [ones(1, wp+1), zeros(1, length(timeAxis)-2*wp-1) ones(1, wp)];
lpfAnalog = repmat(ifft(H), m, 1);

% Mock the analog filtering and sampling
% NOTE: fft and downsample do things by columns, so the signals must
```

```
    % be transposed before performing those operations
    digSamples   = downsample(ifft(fft(mixedChannels.') .* ...
                    fft(lpfAnalog.')), L).';
    digLen       = length(digSamples(1,:));

    %% Digital Expander

    % Digital LPF with cutoff fp/2
    w = (fp/2/fs)*2;
    k = (w*pi*digLen)/(2*pi);
    wp = floor(k);
    H = [ones(1, wp+1), zeros(1, digLen-2*wp-1) ones(1, wp)];
    lpfDig = repmat(ifft(H), m*q, 1); % Replicate for frequency shifting

    n = repmat(0:digLen-1, m*q, 1); % Replicate for frequency shifting

    % Frequency Shift for k=[-q',q'], where q = 2q' + 1
    k = repmat(-floor(q/2):1:floor(q/2), 1, m);
    freqShift = repelem(digSamples, q, 1) .* exp(-1j*(2*pi).*(k.'/q).*n);

    % Digital filtering and sampling
    % NOTE: fft and downsample do things by columns, so the signals must
    % be transposed before performing those operations
    expanderSamples = downsample(ifft(fft(freqShift.') .* ...
                        fft(lpfDig.')), q).';

    %% Continuous to Finite (CTF) Block for Support Recovery

    % Define matrix A without expansion
    S = pit;
    theta = exp(-j*2*pi/L);
    F = theta.^([0:L-1]'*[-L0:L0]);
    pos = 1:L0;
    neg = (-L0):-1;

    d = [(1-theta.^neg)./(2*j*pi*neg) 1/L (1-theta.^pos)./(2*j*pi*pos)];
    D = diag(d);
    A = S*F*D;
    A = conj(A);


    % Expand A by shifting (apparently a loop is faster here)
    if q > 1
        expandedA = zeros(q*m, 2*L0+1);

        A = repelem(A, q, 1);                          % go ahead and expand
        k = repmat(-floor(q/2):floor(q/2), 1, m); % to avoid a second loop

        % Shift each row of A by the value given in k
        for i=1:m*q
            expandedA(i,:) = circshift(A(i,:), L+k(i));
        end
    else
        expandedA = A;
    end
    % Add noise to the signal at the specified SNR
    expanderNoisySamples = awgn(expanderSamples, snr, 'measured');

    [recoveredSupp] = SpectrumBlindReconstruction4(expanderNoisySamples, ...
```

```
                                                           expandedA, N, supp);

      % Check if the support recovery was successful and track the
      % number of successes
      if (length(intersect(supp,recoveredSupp))== length(supp) && ...
          (rank(expandedA(:,recoveredSupp)) == length(recoveredSupp)))
           fprintf(' fi=%d GHz\n', fi(1)/10^9);
           successes = successes + 1;
      else
           fprintf('Failed recovery fi=%d GHz\n', fi(1)/10^9);
      end

end
    % Record number of successful support recoveries for each SNR value
    successes_table(1,r) = successes;
    r = r + 1;
end
```

Appendix B

Input Signal Indices Calculation

breaklines

```matlab
function [Supp] = indices_calc(L0, fi, fp, B)
    % Calculate the slice indices for a given carrier frequency fi and
    % bandwidth B, where there are 2L0+1 slices of width fp.

    L = 2*L0+1;
    j=0:(L0-1);
    % Build a matrix with the positive slice boundaries
    supp_ref = zeros(1,L0+1);
    supp_ref(1,1:L0+1)=[0 fp*(1/2+j)];
    % Find the index that is closest to the left edge of fi
    [m1,i1] = min(abs(repmat(fi-B/2, 1, L0+1) - supp_ref));
    % Find the actual index of the left-slice
    if (fi-B/2 > supp_ref(i1-1) && fi-B/2 < supp_ref(i1))
        StartPos = L0 + i1-1;
    elseif (fi-B/2 > supp_ref(i1) && i1 == length(supp_ref))
        StartPos = L0 + i1;
    elseif (fi-B/2 > supp_ref(i1) && fi-B/2 < supp_ref(i1+1))
        StartPos = L0 + i1;
    end

    % Find the actual index of the right-slice
    [m2,i2] = min(abs(repmat(fi+B/2, 1, L0+1) - supp_ref));
    % Find the index that is closest to the right edge of fi
    if (fi+B/2 > supp_ref(i2-1) && fi+B/2 < supp_ref(i2))
        EndPos = L0 + i2-1;
    elseif (fi+B/2 > supp_ref(i2) && (i2 == length(supp_ref)))
        EndPos = L0 + i2;
    elseif (fi+B/2 > supp_ref(i2) && fi+B/2 < supp_ref(i2+1))
        EndPos = L0 + i2;
    end

    % Return the positive and negative support indices
    Supp = [L+1-StartPos L+1-EndPos StartPos EndPos];
end
```

Appendix C

Spectrum Blind Reconstruction 4

breaklines

```

function recoveredSupp = SpectrumBlindReconstruction4(y, A, N, OrigSupp)
    % SpectrumBlindReconstruction4 implements the SBR4 algorithm presented
    % in "Blind Multiband Signal Reconstruction: Compressed Sensing for
    % Analog Signals".
    %
    % It constructs a frame V from the vector of noisy measurements y, then
    % solves the compressed sensing problem V=AU, where the recovered U
    % is the support of the measurements y.

    % Form the correlation matrix Q
    Q = y*y';
    [Vinit, Dinit] = eig(Q);
    d = diag(Dinit);
    numNonZeroEigVals = sum(abs(diag(Dinit) > 1e4));

    % Now isolate the largest eigenvectors from the rest
    [dSorted, ind] = sort(d);
    numNonZeroEigVals = min(numNonZeroEigVals, 2*N);
    dShort = dSorted(ind(length(ind)-numNonZeroEigVals+1:length(ind)));
    VShort = Vinit(:,ind(length(ind)-numNonZeroEigVals+1:length(ind)));

    % Construct the frame from the short list of eigenvalues and
    % eigenvectors
    VFrame = VShort*diag(sqrt(dShort));

    % Recover the support of the signal via Orthogonal Matching Pursuit
    recoveredSupp = OMP(VFrame, A, length(OrigSupp)/2);

    % Return the recovered support
    recoveredSupp = sort(unique(recoveredSupp));
end
```

Appendix D

Orthogonal Matching Pursuit

breaklines

```
function U = OMP(V, A, N)
    % Simultaneous Orthogonal Matching Pursuit (OMP) algorithm
    % Solves the compressed sensing problem V=AU by finding the sparsest
    % support set U.

    R = V;  % The residual
    U = []; % Recovered support set
    normColsA = vecnorm(A).';
    for i=1:N
        b1 = A' * R; % Form residual signal estimate
        b = vecnorm(b1.').'./normColsA;

        [maxVal, maxPos] = max(b);

        % Add the new positive and negative frequency slice indices to the
        % existing support set
        L = length(A);
        U = [U maxPos L+1-maxPos]; % Add positive and negative frequencies

        % Update the residual before the next iteration
        soln = pinv(A(:,U))*V;
        R = V - A(:,U)*soln;
    end
end
```

Appendix E

Input Signal Generation

breaklines

```
%% input_gen.m
% Script to generate a CSV file for an input signal to the MWC, targeted
% to the Keysight M8196A AWG.  The parameters under "Signal Parameters"
% should be appropriately specified, and the output will be saved to a
% CSV file.
%
%% Signal Parameters

N = 2;                          % Number of bands (including conjugates)
B = 80e6;                       % BW of each band
fnyq = 14.355e9;                % Nyquist frequency
Tnyq = 1/fnyq;                  % Nyquist period
Ei   = [1 2];                   % Energy of the ith band

q = 7;                          % Expansion factor (must be odd)

len  = 91;                      % Length of the signal
zp   = 10;                      % Save room for zero-padding
L    = 165;                     % Aliasing rate
tRes = Tnyq/q;                  % Time axis resolution

obsPeriod = [0  L*q*len-1 L*q*(len+zp)-1]*tRes;% Observation window
Taui = [0.4 0.7]*max(obsPeriod);               % Offset of the ith bandz

%% Generate Input Signal x(t)
timeAxisShort = obsPeriod(1) : tRes : obsPeriod(2);    % Without padding

% Choose random carriers from [fmin, fmax]
fi = 5.0e9;

% Calculate x
n = 1:(N/2);
x = sum(sqrt(Ei(n)') * sqrt(B).*sinc(B*(timeAxisShort-Taui(n)')) .* ...
    cos(2*pi*fi(n)'.*(timeAxisShort-Taui(n)')),1);

% Add a Hann window for smoothing
hannWin = hann(length(x))';          % Hann window
x = [x.*hannWin, zeros(1,q*zp*L)];   % Zero padding

x = downsample(x,q);

```

```
%% Save the signal to a CSV file
output_data = zeros(length(x), 3); % Two extra columns for markers (unused)
output_data(:,1) = x(1,:)';
filename = sprintf('input_signal_B20MHz_%dGHz.csv', fi/10^9);
writematrix(output_data, filename);
```

Appendix F

$p_i(t)$ Generation

breaklines

```
%% pit_gen.m
% Read in a CSV file containing a set of pi(t), then write it to a MAT
% file in the format expected by the Tektronix AWG 70002B.
patterns = readmatrix('SuccessPattern_HW_Test1_m2.csv');
ind = size(patterns);
for i=1:ind(1)
    baseWfm = patterns(i,:);
    baseWfm = repmat(baseWfm,1,10);
    Waveform_Name_1 = sprintf('%dtest',i);
    Waveform_Data_1 = baseWfm;
    % Save as a MAT file
    save(Waveform_Name_1, '*_1', '-v7.3'); % MAT 7.3 Can save > 2GB
end
```

Appendix G

Data_Capture.py

```python
from ctypes import *
import os


'''***** Loading the HSDCPro Automation DLL *****
   Python Script to capture data from a compatible TI ADC
   Evaluation Module using a TI Data Capture Board though
   the HSDC Pro software.
   The pertinent parameters are:
        - Board serial number
        - ADC device
        - ADC sampling rate
        - # samples to capture
        - # samples per channel
        - Timeout
        -
'''

if'PROGRAMFILES(X86)' in os.environ:
    dll_path = "C:\\Program Files (x86)\\Texas Instruments\\" \
               "High Speed Data Converter Pro\\" \
               "HSDCPro Automation " \
               "DLL\\32Bit DLL\\HSDCProAutomation.dll "
else:
    dll_path = "C:\\Program Files\\Texas Instruments\\" \
               "High Speed Data Converter Pro\\" \
               "HSDCPro Automation DLL" \
               "\\32Bit\\DLL\\HSDCProAutomation.dll "
HSDC_Pro = cdll.LoadLibrary(dll_path)


'''*******************************************'''
'''*****ADC Configuration Settings*****'''
'''*******************************************'''


```

```python
Boardsno = "T05BKLDk(10AX115)"  # Board serial number
Devicename = "ADC12QJxx00_JMODE0"  # ADC device (matches HSDC Pro)


Datarate = 1218000000  # ADC output Data Rate
SamplesForAnalysis = 262144  # ADC Analysis Window Length
ChannelIndex = 0  # 0-Based Select ADC Channel
PNGChannelIndex = 0  # 0-Based For Saving FFT as PNG
ImageFormat = 2  # 0-BMP 1-JPEG 2-PNG


# FFT Window Notching
FFTSettingsType = 0  # 0-Rectangular 1-Other Windows
NumberOfCustomFreq = 0
NoofHarmonics = 5
BinsToRemove = 0
BinsToRemoveDC = 1
BinsToRemoveFund = 0
CustomNotchFrequeancies = (c_double*NumberOfCustomFreq)()
BinsToRemove = (c_ulonglong*NumberOfCustomFreq)()
enableFsby2MinusFinNotching = 0  # 0-Disable 1-Enable
binsToRemoveOnEitherSideOfFsby2 = 0


# Trigger Options
TriggerOption = 0  # 0-Normal Capture 1-SW Trigger 2-HW Trigger


NoofChannels = 4


# Enable or Disable Capture to File Option
FileCapEn = 1


# Get Array of Capture Data
SamplesPerChannel = 262144
OffsetSamplePerChannel = 0
Capture_Data_Array_Len = NoofChannels * SamplesPerChannel
CaptureData_16bits = (c_ulong*Capture_Data_Array_Len)()


TimeoutinMs = 30000  # 30 seconds
ADC_Average = 0
Num_Captures = 0
take_screenshots = 0


'''*********************************************************'''
'''** The actual call to the function contained in the dll **'''
```

```
'''***********************************************************'''


# Connecting to board
Err_Status = HSDC_Pro.Connect_Board(Boardsno, TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Select the ADC device and automatically download its FW.
Err_Status = HSDC_Pro.Select_ADC_Device(Devicename, 120000)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Reloading Device INI..."
Err_Status = HSDC_Pro.Reload_Device_INI(TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Using HSDC Ready function to check if the GUI is Ready...
Err_Status = HSDC_Pro.HSDC_Ready(120000)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Passing ADC Output Data Rate
Err_Status = HSDC_Pro.Pass_ADC_Output_Data_Rate(c_double(Datarate),
                                                TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Setting No of Samples
Err_Status = \
    HSDC_Pro.Set_Number_of_Samples(c_ulonglong(SamplesPerChannel),
                                   TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Setting ADC Analysis Window Length
Err_Status = \
```

```
    HSDC_Pro.ADC_Analysis_Window_Length(c_ulong(SamplesForAnalysis),
                                        TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# FFT Window Notching...
Err_Status = \
    HSDC_Pro.FFT_Window_Notching(c_ulong(FFTSettingsType),
                                 c_ulonglong(NoofHarmonics),
                                 c_ulonglong(BinsToRemove),
                                 c_ulonglong(BinsToRemoveDC),
                                 c_ulonglong(BinsToRemoveFund),
                                 CustomNotchFrequeancies,
                                 BinsToRemove,
                                 c_ulonglong(NumberOfCustomFreq),
                                 enableFsby2MinusFinNotching,
                                 binsToRemoveOnEitherSideOfFsby2,
                                 TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Setting Enable Capture to File
Err_Status = \
    HSDC_Pro.Set_Write_Capture_to_File(c_ubyte(FileCapEn),
                                       TimeoutinMs)
if Err_Status != 0:
    print "Error Status = " + str(Err_Status)
    quit()


# Take 50 data captures
for x in range(1, 51):
    CSVData = "C:/Users/aew0056/OneDrive - Auburn University/" \
              "Spectrum Awareness/CFRdemo/11_11/" \
              "6GHz_pi6/6GHz_2Ch_" \
              + "Test" + str(x) + ".csv"

    if TriggerOption == 0:
        TriggerModeEnable = 0
        SoftwareTriggerEnable = 0
        ArmOnNextCaptureButtonPress = 0
        TriggerCLKDelays = 0

        # Setting Normal Capture...
        Err_Status = \
            HSDC_Pro.Trigger_Option(TriggerModeEnable,
```

```
                                    SoftwareTriggerEnable,
                                    ArmOnNextCaptureButtonPress,
                                    c_ubyte(TriggerCLKDelays),
                                    TimeoutinMs)
        if Err_Status != 0:
            print "Error Status = " + str(Err_Status)
            quit()

        # Setting ADC Average Settings...
        Err_Status = HSDC_Pro.ADC_Average_Settings(ADC_Average,
                                                   Num_Captures,
                                                   TimeoutinMs)
        if Err_Status != 0:
            print "Error Status = " + str(Err_Status)
            quit()

        # Starting normal capture...
        Err_Status = HSDC_Pro.Pass_Capture_Event(TimeoutinMs)
        if Err_Status != 0:
            print "Error Status = " + str(Err_Status)
            quit()

    # Getting an array of Capture Data in 16 bits...
    Err_Status = \
        HSDC_Pro.Get_Capture_Data_16bits(SamplesPerChannel,
                                         OffsetSamplePerChannel,
                                         CaptureData_16bits,
                                         Capture_Data_Array_Len,
                                         TimeoutinMs)
    if Err_Status != 0:
        print "Error Status = " + str(Err_Status)
        quit()

    # Saving ADC Codes and Measurement as CSV File...
    Err_Status = \
        HSDC_Pro.Save_ADC_Codes_And_Measurements_As_CSV(CSVData,
                                                    TimeoutinMs)
    if Err_Status != 0:
        print "Error Status = " + str(Err_Status)
        quit()

    if take_screenshots != 0:
        for y in range(2):
            FFTPNGFilePathWithName = \
                "C:/Users/aew0056/OneDrive - Auburn University/" \
                "Spectrum Awareness/CFRdemo/11_11/" \
                "screenshots/0.5GHz_2Ch_FFT_Test" \
                + str(x) + "_Ch" + str(y + 1) + ".png"

            # Saving FFT as PNG File...
```

```
            Err_Status = \
                HSDC_Pro.Save_FFT_As_PNG(c_ulong(y),
                                         FFTPNGFilePathWithName,
                                         TimeoutinMs)
            if Err_Status != 0:
                print "Error Status = " + str(Err_Status)
                quit()

# Disconnecting from the board
Err_Status = HSDC_Pro.Disconnect_Board(30000)
if Err_Status != 0:
    quit()
```

Appendix H

MWC Hardware Data Processing

breaklines

```
%% Modulated Wideband Converter (MWC) Data Processing from ADC
%
% Process data and perform support recovery for actual samples
% from an MWC system.
% The necessary system parameters are defined at the top, and the rest
% of the program is divided into the following sections:
%   - Analog LPF cleanup
%   - Digital Expander
%   - Continuous to Finite (CTF) Block
% This program is currently configured to perform support recovery on
% 50 CSV files in a single folder. The filename for the CSV file
% containing the correct pi(t) and the folder location of the data
% CSV files are required. For 50 CSV files with 262,144 samples, a
% total of 9,250 trials are run. The number of successful support
% recoveries is saved and printed at the end.
successes = 0;
total = 0;
pit = readmatrix('SuccessPatterns_HW_Test6_m2.csv');

for z=1:50
    %% Parameters
    N = 2;              % Number of bands (incl. conjugates)
    B = 10e6;           % BW of each band
    fnyq = 14.355e9;    % Nyquist frequency
    Tnyq = 1/fnyq;      % Nyquist period

    q = 7;              % Expansion factor (must be odd)
    L = 165;            % Aliasing rate
    M = L;              % Length of the sign patterns
    L0 = floor(M/2);    % L = 2L0 + 1

    fp = fnyq/L;        % Frequency of the sign patterns
    fs = q*fp;          % Sampling rate at each channel,
                        % use fs=qfp, with odd q
    fs_hw = 2*fs;       % Sampling rate in HW is 2*fs to allow the cutoff
                        %  of the analog LPF to be cleaned up

    m = ceil((4*N)/q);  % Number of channels
    numSamples = 1414;  % Number of samples per test per support recovery

    fi = [3.5e9];       % The expected carrier frequencies from the
```

```
                        % recovered signal

 % Calculate the original support set
supp = [];
for i=1:(N/2)
    supp = [supp indices_calc2(L0, fi(i), fp, B)];
end
supp = sort(unique(supp));

% Point to the folder where the CSV files for a single frequency and
% sign pattern are located
filename = sprintf('11_11_B10MHz\\3.5GHz_pi6\\3.5GHz_2Ch_Test%d.csv', z);
x1 = readmatrix(filename);
x1 = x1(1:floor(length(x1)/numSamples)*numSamples,1:m);

% Recover the support for every subset of samples within a single file
for u=1:floor(length(x1)/numSamples)
    %% Select a subset of samples, filter and downsample
    total = total + 1;
    % Clean up for the analog LPF using a digital LPF with cutoff
    % fs/2
    w = (fs/2/fs_hw)*2;
    k = (w*pi*numSamples)/(2*pi);
    wp = floor(k);
    H = [ones(1, wp+1), zeros(1, numSamples-2*wp-1) ones(1, wp)];

    % Select a subset of samples from the long signal
    x = x1(((u-1)*numSamples)+1:(u)*numSamples,1:m);
    % Apply LPF to each channel
    lpfHW = repmat(ifft(H), m, 1);
    digSamples = downsample(ifft(fft(x) .* fft(lpfHW.')), 2).';
    digLen = length(digSamples(1,:));


    %% Digital Expander

    % Digital LPF with cutoff fp/2
    w = (fp/2/fs)*2;
    k = (w*pi*digLen)/(2*pi);
    wp = floor(k);
    H = [ones(1, wp+1), zeros(1, digLen-2*wp-1) ones(1, wp)];
    lpfDig = repmat(ifft(H), m*q, 1); % Replicate for freq shifting

    n = repmat(0:digLen-1, m*q, 1); % Replicate for frequency shifting

    % Frequency Shift for k=[-q',q'], where q = 2q' + 1
    k = repmat(-floor(q/2):1:floor(q/2), 1, m);
    freqShift = repelem(digSamples, q, 1) .* ...
                        exp(-1j*(2*pi).*(k.'/q).*n);

    % Digital filtering and sampling
    % NOTE: fft and downsample do things by columns, so the signals
    % must be transposed before performing those operations
    expanderSamples = downsample(ifft(fft(freqShift.') .* ...
                                fft(lpfDig.')), q).';


    %% Continuous to Finite (CTF) Block for Support Recovery
    % Define matrix A without expansion
```

```
        S = pit;
        theta = exp(-j*2*pi/L);
        F = theta.^([0:L-1]'*[-L0:L0]);
        pos = 1:L0;
        neg = (-L0):-1;

        dn = [(1-theta.^neg)./(2*j*pi*neg) 1/L ...
              (1-theta.^pos)./(2*j*pi*pos)];
        D = diag(dn);
        A = S*F*D;
        A = conj(A);


        % Expand A by shifting (apparently a loop is faster here)
        if q > 1
            expandedA = zeros(q*m, 2*L0+1);

            A = repelem(A, q, 1);                       % go ahead and expand
            k = repmat(-floor(q/2):floor(q/2), 1, 2); % to avoid a 2nd loop

            % Shift each row of A by the value given in k
            for i=1:m*q
                expandedA(i,:) = circshift(A(i,:), L+k(i));
            end
        else
            expandedA = A;
        end

        recoveredSupp = SpectrumBlindReconstruction4(expanderSamples, ...
                                                     expandedA, N, supp);

        % Check if the support recovery was successful and track the number
        % of successes
        if (length(intersect(supp,recoveredSupp))== length(supp) && ...
           (rank(expandedA(:,recoveredSupp)) == length(recoveredSupp)))
            fprintf('Successful fi=%d GHz and Test %d, iteration %d\n', ...
                     fi(1)/10^9, z, u);%, fi(2)/10^9);
            successes = successes + 1;
        else
            fprintf('Failed fi=%d GHz and Test %d, iteration %d\n', ...
                     fi(1)/10^9, z, u);
        end

    end
end
successes
```