UNIVERSITY OF CALIFORNIA

Los Angeles

Analysis and Simulation of Impairments in Xampling Based Receivers

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

by

John Man Ong

2012

ABSTRACT OF THE THESIS

Analysis and Simulation of Impairments in Xampling Based Receivers

by

John Man Ong

Master of Science in Electrical Engineering

University of California, Los Angeles, 2012

Professor Sudhakar Pamarti

Recent advances in Compressed Sensing theory has lead researcher to apply these techniques to the design of sampling systems which can sample spectrally sparse signals with a wide bandwidth below the Nyquist rate. Xampling is a framework proposed to design such Sub-Nyquist sampling systems. This work presents results and analysis of simulations of impairments in of such systems. The impacts of hardware impairments, choice of reconstruction algorithm, signal energy dynamic range and channel mismatches on system performance are studied. Results show that if a system is designed with a sufficient number of channels then it may be able to maintain performance despite impairment.

The thesis of John Man Ong is approved.

Babak Daneshrad

William J. Kaiser

Sudhakar Pamarti, Committee Chair

University of California, Los Angeles

2012

This work is dedicated to my father. Thanks dad.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank first my advisor, Professor Pamarti, for guiding me during the research and preparation of this work. I would also like to thank my fellow graduate students, Mansour Rachid, Long Nyguen and Abhishek Ghosh, with whom I have had many enlightening and fruitful conversations. Furthermore, I could like to thank Abhishek again for his help in reviewing this work. Additionally, I would like the thanks those with whom I've worked with on class projects in related areas, specifically Edward Loden and Long Nguyen. Lastly, I'd like to thank the staff in the Electrical Engineering Department, whose help and patients were indispensible throughout.

I. MOTIVATION

In many areas of electronics, the conversion from an analog to digital information is being pushed closer and closer to the front end of the signal path. One example where this is evident is in software defined radio, where the conversion from analog to digital is as close to the front end as possible. In this way, systems can take advantage the flexibility and scalability of manipulating the system through digital and software means.

However, data converters have an inherent trade-off between temporal and spacial resolution. The conversion rate dictates the former, while the number of bits of resolution dictates the later. In conventional uniform sampling, the Nyquist Theorem provides a simple relation between the sampling frequency and the highest frequency which can be accurately sampled. This theorem states that the sampling frequency must be at least twice the highest frequency component present in the sampled signal.

Today's communication technology can operate at very high frequencies over a wide band. For example, today's Radio Frequency (RF) technologies can operate at the tens of Gigahertz range. ADCs cannot achieve this kind of bandwidth, except at very low resolutions. Figure 1 shows several ADCs which have been published in the International Solid-State Circuits Conference (ISSCC) over the past several years [1]. The conversion speed is listed on the y-axis and is given in Hertz. The x-axis reflects the resolution and is given as a Signal-to-Noise and Distortion Ratio (SNDR) with units of decibels. Note that the effective number of bits (ENOB) can provide a more intuitive measure of the resolution and a given SNDR can be converted to ENOB following the convention 6dB of SNDR ≈ 1 Effective Bit of resolution. Note that the ADCs represented here are published research papers and may not be as reliable as commercially available devices.

This means that the ADCs shown in Figure 1 represent some of the highest performance devices available.

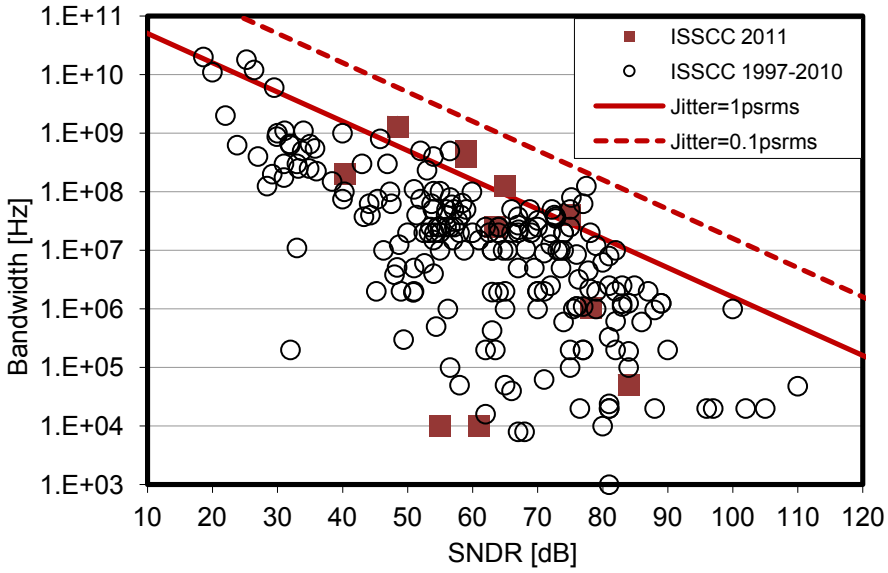

Figure 1 - Published ADCs, Bandwidth vs. Resolution [1]

In many communication settings, it is not uncommon for the Nyquist rate of the transmitted signal to be too high for direct sampling, yet the actual spectral occupancy of that signal may be relatively low. That is, the information bandwidth of the signal is much lower than the Nyquist bandwidth of the overall transmitted signal. An example of this scenario is depicted in Figure 2.

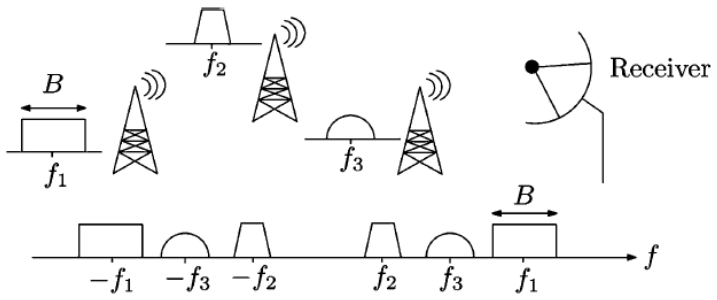

Figure 2 - Typical communication setting with info bandwidth less than overall bandwidth [2]

In this setting, it is common to demodulate a given band of interest by carrier frequency, bringing the signal of interest to baseband. The signal is then low pass filtered to reject unwanted portions of the spectrum and the information band can be sampled at a low rate. While this is a common and relatively straightforward approach, a priori knowledge of the carrier frequency is required. When the carrier frequency is unknown, the task of sensing and sampling the signal becomes very difficult.

Within the past several years, various approaches have been proposed which leverage recent advances in the field of Compressed Sensing (CS) to sample high bandwidth signals which have low spectral occupancy. One approach, deemed the Analog-to-Information Converter, uses Random Demodulation and targets multi-tone signals [3]. Another approach, the Modulated Wideband Converter (MWC), uses a modified Multi-Coset sampling approach and targets signals with some known, finite bandwidth [2]. The MWC fits into the broader Xampling framework, which provides a methodology for designing sampling systems which can treat wideband analog signals at Sub-Nyquist rates [4]. A board level implementation of the MWC can be found in [5], yet how to design a practical hardware realization is still not completely understood. This work studies these challenges, and provides analysis and simulation results related to hardware impairments and other practical considerations which alter the theoretical expectations of Xampling based receivers.

A. Introduction to this work

The main focus of this work is to simulate various impairments associated with practical implementations of Xampling based receivers and study their effects on system performance.

In Section IV.B various hardware impairments associated with each channel of the receiver are considered and their effects on the probability of successful support recovery studied. The results of this experiment show that increased randomness associated with a higher channel count allow the system to remain robust to impairments.

In Section IV.C different OMP based CS reconstruction algorithms were implemented and their performance for various input SNR values was calculated. In this context performance considerations include both probability of successful support recovery and associated computational load, reported by average runtime. The results show that running OMP with an additional refinement step is most robust to low input SNR. Also it is shown that performing additional iterations in the CoSaMP algorithm does not significantly improve performance

In Section IV.D the energy of each input signal component are varied to find the energy dynamic range of the receiver that is, finding the limit to the largest and smallest signal which can be resolved simultaneously. The probability of successful support recovery of both signals was considered. The results of show that if the ratio of signal energies was greater than 10^3, there was a significant drop in this probability of success and for a ratio equal to 10^4, even a system with a large number of channels could not reliably recover the total signal.

In Section IV.E the Xampling based 2.4GHz receiver presented in [6] is studied for various hardware impairments and channel mismatches. In addition to the probability of successful support recover, the reconstructed signal SNR is also calculated for each impairment. The results

of channel mismatches are similar to those found in [6]. Results of hardware impairments are similar to those found in Section IV.B and the same general conclusion can be drawn; for a sufficient number of channels, the signal support can still be recovered with high probability. Additionally, this experiment shows how each impairment impacts the reconstruction SNR.

B. Report Organization

The rest of this report will be organized as follows. Section II contains background material and provides a brief overview of Nyquist and Sub-Nyquist sampling approaches, an overview of Compressed Sensing fundamentals, and an overview of the MWC. Section III provides an overview of a simple MWC system simulation and results. Section IV contains the results of this work, including results from simulations of the hardware impairments and their effect on signal reconstruction, a study of various Compressed Sensing algorithms in practice, simulations of signal reconstruction for input signal energy dynamic range, and a study of the effects of impairments for a 2.4GHz Xampling based receiver. Section V provides the conclusion to this work. Section VI is the appendix, which has a sample of code used in the presented simulations.

II. Background

A. Nyquist and Sub-Nyquist Sampling

1. Nyquist Sampling

The Nyquist rate states that a signal must be sampled with a sampling frequency at least twice the highest frequency component present in the signal to ensure a unique representation. There have been many different ADC architectures employed to increase the ADC sampling frequency and as a result the signal bandwidth it can handle. One such architecture is Time-Interleaved ADCs and is discussed in the following section.

2. Time-Interleaved ADC

Time interleaving is an architecture which aims to increase the overall sampling rate of the ADC system by running several ADCs in parallel [7]. The overall sampling rate is proportional to the number of parallel channels times the sampling frequency of each branch. For example, if there are n parallel channels and the overall rate of the system is R, then each channel ADC runs at a rate of only R/n. This topology suffers from two main drawbacks. First the analog front end which, directs the signals to each channel, must be able to handle the entire bandwidth of the signal [2]. Second, each channel is required to be a timed shift relative to adjacent channels and accuracy of these time shifts becomes difficult at high sampling rates [2]. A special case of Time interleaving, Multi-Coset sampling, is the basis for the sampling stage of the MWC [2].

3. Nyquist Folding

Another sampling topology, overviewed in [4], is Nyquist folding, which achieves a low sampling rate by multiplying the signal with a pulse train generated by the zero crossings of a sinusoid, a block diagram of which can be seen in Figure 3.

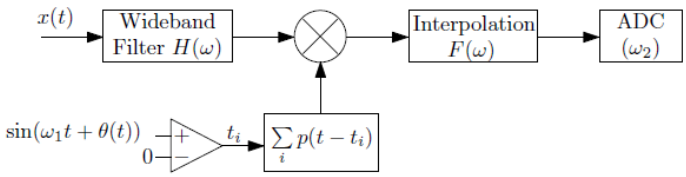

Figure 3 - Block diagram of Nyquist Folding ADC

The main limitation of this system is maintaining the timing accuracy needed to align pulses p(t-t_i) to the time instances t_i. Also signals with a bandwidth > 0 can complicate the folding effect. These limit this topologies use at wide bandwidths.

4. Random Demodulator

A CS based system data converter is the Random Demodulator, and is brief overview can be found in [4]. This system essentially mixes the incoming signal with the output of a Pseudo Random Number Generator, integrates the mixture, and samples it at a low rate. Figure 4 depicts this system.

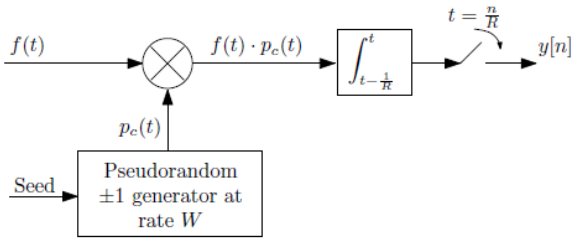

Figure 4 - Block diagram of the Random Demodulator

The main drawback of this system comes from the reconstruction of the input signal, f(t). When solving for this signal, it assumes that f(t) is a multi-tone signal and can only exist at a finite number of frequencies. However, for a f(t) with some bandwidth > 0 this is not the case and this fact severely degrades the overall performance, leading to huge computational loads when solving the CS problem.

5. Periodic Non-Uniform Sampling

Periodic non-uniform sampling uses a bank of samplers where each channel is time delayed by a different amount. Three main drawbacks are the wide bandwidth of the signal at the sampler, the accuracy required in the time shifts and the reconstruction of the signal from the output. Figure 5 shows a block diagram of this system and an overview can be found in [2] [4].

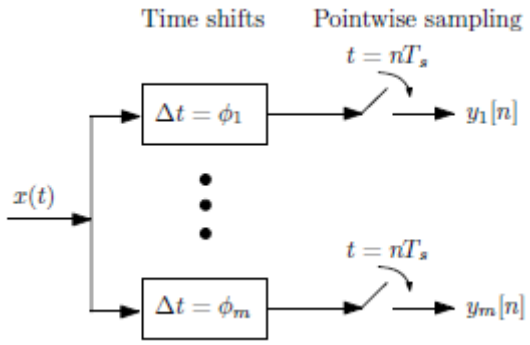

Figure 5 - Block diagram of periodic non-uniform ADC [4]

The first issue to address is the bandwidth of the signal presented to the sampler. In practical ADCs, the full power analog bandwidth, b, dictates the highest bandwidth the inherent track and hold can handle before it distorts the signal [2]. Note that b is inseparable from the ADC and is not directly tied to distortion due to aliasing, or sampling below the Nyquist rate. While the samplers in each branch do sample at only a fraction of the total rate, the full bandwidth of the signal can be presented and thus the sampled data is distorted if that bandwidth is higher than the b of the ADC.

Like time interleaved ADCs, accuracy of the time shifts mentioned above may be difficult to maintain at high speeds. Also during reconstruction, the signal must be interpolated back to Nyquist rate, which results in a large, computational complexity in reconstructing the signal. A Multi-Coset version of this system can be implemented, where the time shifts can only be taken

from discrete values [2]. Although the Multi-Coset version suffers from the same limitations mentioned above, it acts as a basis for the sampling scheme presented in the MWC.

B. Compressed Sensing

1. Introduction

Compressed sensing or compressed sampling (CS) is a mathematical concept which allows accurate reconstruction of a signal from fewer samples than originally thought with a high probability. Unlike many traditional compression schemes which sample then discard information, CS offers a way to directly sample a signal into a compressed format. This saves sensing resources and is important in scenarios where the sensing operation is costly. Figure 6 shows a block diagram comparing the steps of traditional compression and those of CS. In traditional compression, N samples are reduced to M samples which are used to approximate the original signal when uncompressed. In a CS system, only M samples are taken during sampling, which can be later decompressed to approximate the original signal.

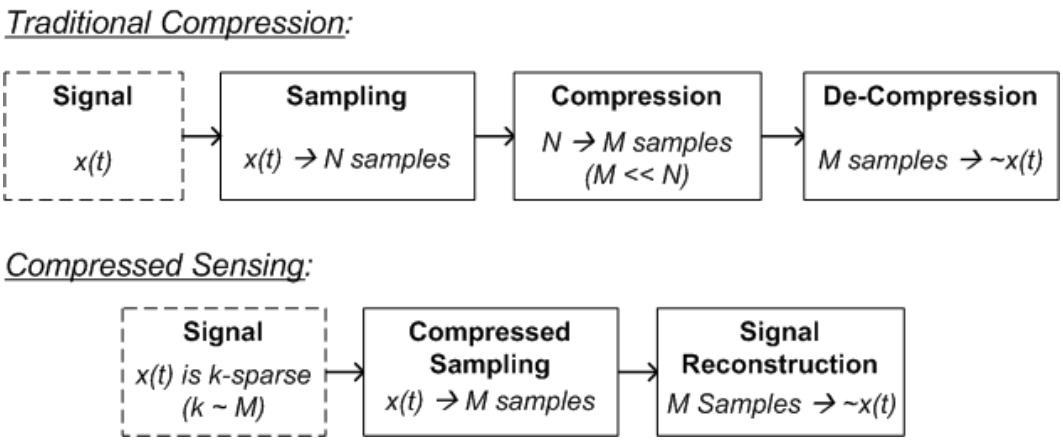

Figure 6 - Block diagram of traditional compression and Compressed Sensing

CS has recently gained popularity due to a seminal paper by Tao, Romberg and Candès and has since become an extremely active area of research [8] [9]. CS theory asserts that signals which conform to certain sparsity conditions can be reconstructed using a linear program exactly or almost exactly even if sampled below the Nyquist rate. For a discrete vector, the sparsity is

defined as the number of non-zero entries. For example, a vector with sparsity k is considered a k-sparse vector and has k non-zero entries with the remaining entries equal to zero. For a sparse, yet noisy signal where most entries will be close to, but not exactly zero, approximate sparsity can be defined.

CS theory can be split into three main topics: the signal to be sensed, the sensing operation, and signal reconstruction. Firstly, the sensed sparse signal is not arbitrary and must be understood before sensing. Next the sensing operation must meet certain criteria to ensure that information is preserved. Analysis methods such as Incoherence Guarantees, Restricted Iosometry Property (RIP), and Expected RIP (exRIP) have been proposed to try to quantify sensing matrices [10] [11] [12] [13] [14]. Finally an appropriate reconstruction algorithm must be implemented to recover the original signal. These topics are discussed in more detail in the following sections.

2. Signal

CS techniques are not applicable to arbitrary signals. Signals which are appropriate must be considered sparse or approximately sparse and therefore compressible in some domain. Fortunately there are several types of signals which fit this signal model, such as a sinusoid when transformed to the Fourier domain. Figure 7 depicts a signal **x** of length N, which can be represented by the appropriate transform basis, **Ψ**, and the k-sparse vector, **z**.

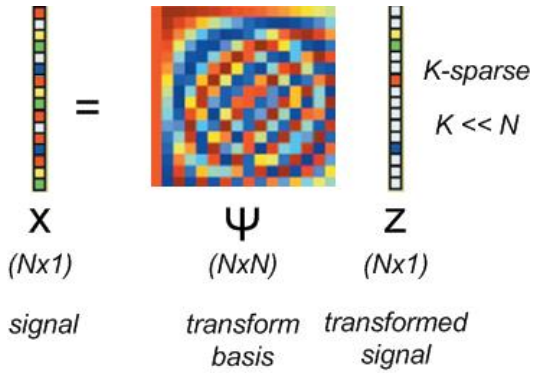

Figure 7 - Vector represented by a transform basis and a sparse representation [15]

Generally, to achieve a reasonable compression ratio, the sparsity is assumed to be much less than the length of the vector. This means the information contained in the signal is stored in only a few of its entries. While the positions of these non-zero entries do not need to be known, the sparsity or number of non-zero entries does. This information is needed to design a sensing matrix to perform the sensing operation, which is usually designed for a minimum required signal sparsity.

3. Sensing

The sensing operation translates a signal into a set of measurements, reducing the length of the vector. This operation can be thought of as multiplication by a sensing matrix, which has fewer rows than columns. This reduction in dimensionality leads to an underdetermined linear system, which generally cannot yield an exact solution when solved through direct methods. However, CS theory states that solving a linear program can still recover the original signal with relatively high accuracy.

a) The Sensing Operation

Since the information in a k-sparse vector is stored in only k of the N entries (k << N), it is important that this information is not destroyed during multiplication with the sensing matrix.

This places certain constraints on how this matrix must be designed for a given sparsity. Figure 8 depicts the sensing operation of a sparse vector, **z**, with sensing matrix, **θ**, resulting in the measurements vector, **y**. Note that this matrix is somewhat agnostic, only depending on the sparsity of the signal to be sensed, and can be fixed, making no attempt to adapt to the signal.

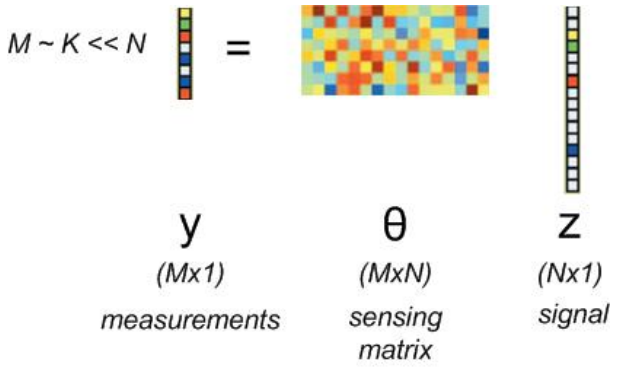

Figure 8 - Sensing operation of sparse vector with sensing matrix [15]

However, it may be the case the signal to be sensed is not sparse in the measurement domain. This would require that the sensing matrix be composed of two matrices: a measurement basis and a transform basis. The former represents the actual measurement action while the later transforms the sensed vector into a sparse representation. Figure 9 depicts the sensing operation of the non-sparse vector, **x**, using measurement basis, **Φ**, and the resulting measurements, **y**. Note that **Φ** is not the overall sensing matrix, only the measurement basis.

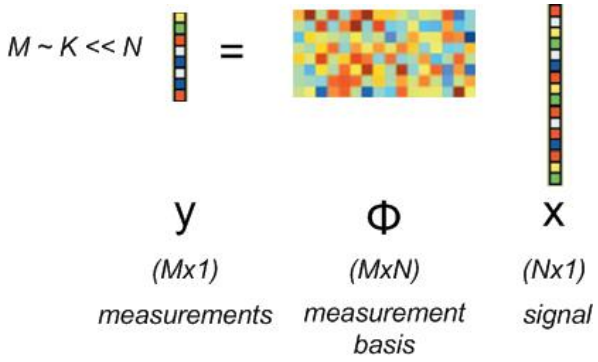

Figure 9 - Sensing of non-sparse signal with measurement basis before transform [15]

Figure 10 depicts the same sensing operation with the signal to be sensed, **x**, represented by a transform basis, **Ψ**, a the sparse representation, **z**. Figure 11 depicts the same sensing operation with the measurement basis, **Φ**, and the transform basis, **Ψ**, combined into the overall sensing matrix, **θ**.

Since the measurements are a result of the product of the sensing matrix and sparse vector, only k columns of the sensing matrix contribute to the measurement vector, **y**. This leads to the notion of the support of a sparse vector or the indices of the sparse vector which has non-zero value. Only the columns of the sensing matrix which correspond to the support of **z** contribute to the measurement, which is a key observation for certain recovery algorithms, such as Orthogonal Matching Pursuit [16].

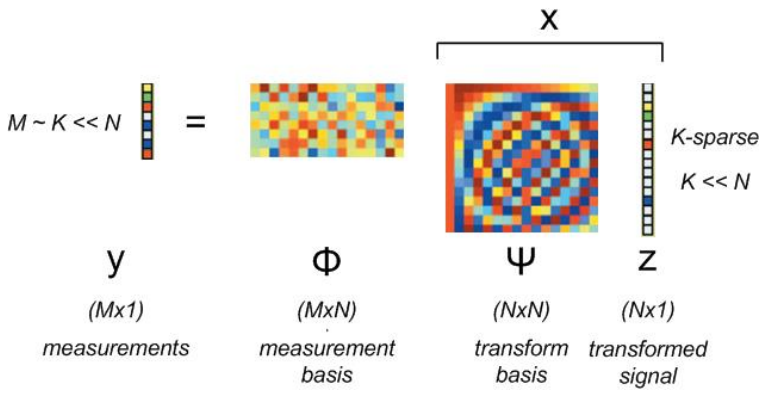

Figure 10 - Sensing operation with measurement basis and transform basis [15]

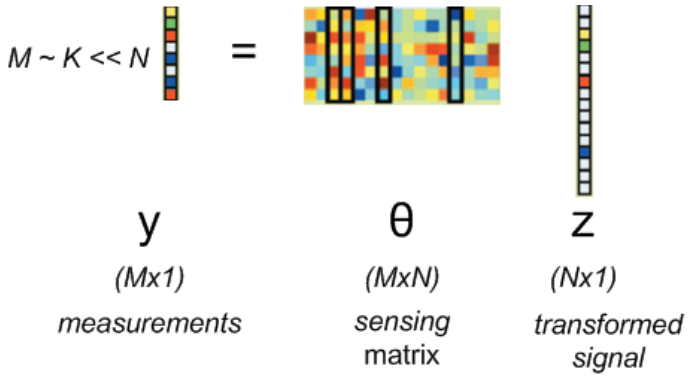

Figure 11 - Sensing operation sensing matrix as a combination of measurement and transform bases [15]

b) Choosing the Sensing Matrix

As stated before, the sensing operation should preserve the information contained in the sparse vector and therefore the sensing matrix cannot be chosen arbitrarily. Only if the information is preserved can reconstruction algorithms successfully reconstruct the original signal. There have been several proposed methods to attempt to quantify which matrices would yield the best results. In general, these methods specify a number of samples needed to ensure a certain probability of successful recovery. Some of these methods include Incoherence Guarantees [10] [11] [14], the Restricted Isometry Property (RIP) [10] [12] [13], and Expected RIP [14]. Generally it has been shown that matrices whose entries are drawn from random distributions are shown to perform well.

(1) Mutual Incoherence

Along with sparsity, the notion of incoherence between the measurement matrix $\mathbf{\Phi}$ and the transform matrix $\mathbf{\Psi}$ is a fundamental part of CS theory. This means that if the signal has a sparse representation in $\mathbf{\Psi}$, then a low coherence (or being incoherent) between the measurement and transform bases guarantees that the signal will have a dense representation in the domain in which it is measured [10]. Coherence is defined in [10] as

$$\mu(\Phi, \Psi) = \sqrt{n} \max_{1 \le k,j \le n} \left| \langle \varphi_k, \psi_j \rangle \right| \quad (1)$$

where φ_k and ψ_j represent the k^{th} column of $\mathbf{\Phi}$ and the j^{th} column of $\mathbf{\Psi}$, respectively, and $\mathbf{\Psi}$ is an (N x N) matrix. If the elements of these bases are correlated, then the coherence is high and vice versa. The coherence can range in value from $\mu(\Phi, \Psi) \in [1, \sqrt{N}]$ and CS theory is mainly concerned with bases pairs with low coherence.

An example of this is classical sampling, that is, the measurements are performed by a spike basis and the transform basis is the Fourier basis. This pair of bases has a $\mu(\Phi, \Psi) = 1$ and is considered maximally incoherent. It has also been shown that sensing matrices made up of random waveforms with entries which are identically independent distributed (i.i.d.) have low coherence with any fixed representation $\mathbf{\Psi}$. This leads to the idea of a universal measurement matrix, such that, if this matrix satisfies randomness assumptions, than it can be used in combination with several different transform bases while ensuring low coherence.

A result from [11] states the required number of measurements needed to ensure exact signal recovery through solving the l_1 minimization problem with high probability.

$$m \ge C \cdot \mu^2(\Phi, \Psi) \cdot k \cdot \log(N) \quad (2)$$

C is some positive constant. It is also shown that the probability of successful reconstruction exceeds $1 - \delta$ if

$$m \ge C \cdot \mu^2(\Phi, \Psi) \cdot k \cdot \log(N/\delta) \quad (3)$$

This shows the relationship between coherence and the number of measurements needed; the lower the coherence, the fewer measurements are needed.

(2) Coherence Based Guarantees

Coherence guarantees can also be applied to the overall sensing matrix, **θ**, with a survey of results found in [14]. These relate the coherence of a sensing matrix to the sparsity of the sensed signal, which in turn dictates the size of the matrix and the number of measurements which need to be taken. Here, coherence of the matrix **θ** is defined by

$$\mu(\theta) = \max_{i \neq j} \frac{|\langle \theta_i, \theta_j \rangle|}{\|\theta_i\| \, \|\theta_j\|} \tag{4}$$

Theorem 7, found in [17] states that given the sparsity is low enough compared to the coherence as defined in **(4)**, the sparse signal can be recovered through l_1 minimization. The relationship between sparsity, k, and the coherence is given below.

$$k < \frac{1}{2}\left(1 + \frac{1}{\mu}\right) \tag{5}$$

Similar results for reconstruction through l_1 minimization in the presence of noise can be found in [17] [18] [19]. Another result which considers noise in addition to the assumption that the locations of the non-zero entries in the sparse vector are drawn uniformly at random can be found in [20]. While the details are omitted for brevity, it is clear that a low coherence is ideal in the context of CS.

(3) Restricted Isometry Property (RIP)

The Restricted Isometry Property (RIP) was first proposed in [12] and later discussed in [8] and [13], among others. It is defined by an isometry constant, δ_k, which is defined for a sensing matrix, **θ**, used to sense an k-sparse vector, **x**, as defined in **(6)**.

$$y = \theta x \tag{6}$$

This constant defines the ability for a sensing matrix to persevere the l_2 norm of an arbitrary k-sparse vector. Formally, this constant has the following definition.

For each integer k = 1,2,…, define the isometry constant δ_k of a matrix θ as the smallest number such that

$$(1 - \delta_k)\|x\|_{l_2}^2 \leq \|\theta x\|_{l_2} \leq (1 + \delta_k)\|x\|_{l_2}^2 \quad (7)$$

for all k-sparse vectors. A k-sparse vector has at most k non-zero entries.

The ideal sensing matrix would have the lowest possible isometry constant. Explicitly solving for this constant is not arbitrary; however, if an explicit bound on its value can be assumed, then it implies certain guarantees for signal reconstruction via solving the l_1 problem. Assuming the isometry constant meets this bound signal reconstruction is exact in a noiseless system and approximate within some bound for a noisy system.

Matrices with entries drawn from random distributions such as Gaussian or Bernoulli distributions have been shown to satisfy RIP conditions with high probability given a sufficient number of measurements are taken [10]. Additionally, it has been shown that the combination of this random matrix and a transform matrix, which itself is an orthonormal basis, also adheres to the same bound [10]. This lower bound on the number of measurements is O(log(N/k)) and given by **(8)**.

$$m \geq C \cdot k \cdot \log\left(\frac{N}{k}\right) \quad (8)$$

Here, m is the number of measurements required, C is some constant, N is the length of the sensed vector and k is the sparsity. The reader is further referred to [8] [9] [21] for more details.

(4) Expected RIP (ExRIP)

The Expected RIP property, proposed in [14], relaxes the requirements associated with RIP by considering a random signal model. This assumes that the locations and amplitudes of the non-

zero entries in the sparse vector are uniformly random, something not assumed in RIP. ExRIP was proposed as a way to condition the sensing matrix associated with the MWC [2] due to the fact that to meet coherence or RIP based guarantees, an impractical number of hardware channels would be required. Through simulation in [14], the MWC was shown to meet ExRIP principles with a high probability for a reasonable number of hardware channels.

(5) Statistical RIP (StRIP)

Statistical RIP (StRIP) [22] is another RIP-like principle which assumes randomness only in the locations of the non-zero entries. StRIP also assumes the sensing matrix is closed form under element-wise multiplication, has rows with are orthogonal and, additionally, sum to zero [14]. The results provide an expression for the probability of StRIP being satisfied based on the isometry constant proposed in RIP. However, in [14] , it is deemed inapplicable to the MWC since the sensing matrix does not meet the assumed criteria, despite a large number of channels. The reader is referred to [22] for details.

4. Reconstruction

There are several reconstruction algorithms proposed to recover a sparse vector from an incomplete set of measurements. L_1 minimization or Basis Pursuit [8] [17] [18] [9] is applicable in the noiseless setting, and Basis Pursuit De-Noising can be used to recover a noisy signal [19]. Another group of methods based on Orthogonal Matching Pursuit (OMP) [16], include Compressed Sensing Matching Pursuit (CosMP) [23], Stagewise Orthogonal Matching Pursuit (stOMP) [24] and Regularized OMP (ROMP).

(1) Basis Pursuit (BP) (l_1 norm minimization, noiseless)

Assume an underdetermined system of equations given by **(9)**.

$$y = \theta x \tag{9}$$

Finding an exact solution for **x** through conventional means is not possible for an arbitrary **x**. However, assuming that **x** can be considered sparse, the solution **x*** to

$$\min_{\tilde{x}\in\mathbb{R}^n} \|\tilde{x}\|_{l_1} \quad \text{subject to} \quad \theta\tilde{x} = y \tag{10}$$

can recover exactly the original signal **x** given the following conditions are satisfied: [25] [26] [13] [17]

1) **x** is sufficiently sparse
2) the sensing matrix **θ** obeys the RIP condition

To measure the quality of reconstruction under the RIP condition, the solution to l_1 problem, **x***, is compared to the oracle solution, $\mathbf{x}_k$. The oracle solution is defined as the best sparse approximation, as if one knew the exact locations and amplitudes of the k-largest entries of **x**. In $\mathbf{x}_k$ all but the k-largest entries are zero.

Assume a sensing matrix with isometry constant which satisfies the inequality **(11)**.

$$\delta_{2k} < \sqrt{2} - 1 \tag{11}$$

The solution to the linear program **(10)**, **x***, obeys **(12)** and **(13)** for some constant C_0.

$$\|x^* - x\|_{l_1} \leq C_0 \|x - x_k\|_{l_1} \tag{12}$$

$$\|x^* - x\|_{l_2} \leq C_0 \frac{1}{\sqrt{k}} \|x - x_k\|_{l_1} \tag{13}$$

If the **x** is k-sparse, then the oracle solution, $\mathbf{x}_k$, exactly matches **x**. Therefore, the solution to the **x*** also matches **x** and recovery is exact. Also, the solution recovers the sparsest solution of $\theta\tilde{x} = y$ or the solution to the l_0 problem

$$\min_{\tilde{x}\in\mathbb{R}^n} \|\tilde{x}\|_{l_0} \quad \text{subject to} \quad \theta\tilde{x} = y \tag{14}$$

While explicitly solving this is an NP hard combinatorial problem, the above shows that the solving **(14)** and the solving the l_1 problem given in **(10)** are equivalent. Explicitly,

- If $\delta_{2k} < 1$, the l_0 problem has an unique k-sparse solution
- If $\delta_{2k} < \sqrt{2} - 1$, solution to l_1 and l_0 problems match, convex relaxation is exact

Generally speaking, if $\delta_{2k} < 1$ than any k-sparse solution is unique. By contradiction, if $\delta_{2k} = 1$, this means that 2k columns of the sensing matrix, **θ**, may be linearly dependent and as a result, there is a 2k-sparse vector, **h**, such that **θh** = 0. This vector **h** can be decomposed into two unique k-sparse vectors, **h** = **g**-**g**'. It then follows that **θg** = **θg**' meaning that there is no method to recover all k-sparse vectors with the given sensing matrix, **θ**.

(2) Basis Pursuit De-Noising (BP-DN) (l_1 norm min, noisy)

Basis Pursuit De-Noising extends Basis Pursuit for more realistic scenarios where noise corrupts the measurements is given by

$$y = \theta x + n \tag{15}$$

where **n** is some noisy contribution to the measurements **y** [10] [12] [13]. This can be thought of as associated with the sensed signal, $\mathbf{x}_n$, and would be considered approximately sparse. This can be decomposed into a truly sparse noiseless signal, **x** or $\mathbf{x}_{ideal}$, and a noise component $\mathbf{x}_{noise}$ as shown in **(16)**.

$$y = \theta x_n = \theta(x_{ideal} + x_{noise}) = \theta x_{ideal} + \theta x_{noise} = \theta x + n \tag{16}$$

With noise introduced into the system, the l_1 problem is recast and given by

$$\min_{\tilde{x} \in \mathbb{R}^n} \|\tilde{x}\|_{l_1} \quad \text{subject to} \quad \|y - \theta\tilde{x}\|_{l_2} \leq \varepsilon \tag{17}$$

where ε represents an upper bound on the noisy contribution. As in the analysis of BP, the measure of quality of reconstruction is carried out by comparing the solution to **(17)**, **x***, and the

oracle solution, $\mathbf{x}_k$. In this case $\mathbf{x}_k$ refers to a signal which would result from directly measuring the k largest coefficients of the noisy signal $\mathbf{x}_n$.

Again assume a sensing matrix with isometry constant which satisfies the inequality **(18)** and a bound on the norm of the noise term given by **(19)**.

$$\delta_{2k} < \sqrt{2} - 1 \tag{18}$$

$$\|n\|_{l_2} < \varepsilon \tag{19}$$

With these assumptions, the solution to (20) obeys

$$\|x^* - x\|_{l_2} \leq C_0 \frac{1}{\sqrt{k}} \|x - x_k\|_{l_1} + C_1 \varepsilon \tag{20}$$

where C_0 is the same constant from BP and C_1 is a new constant. The reconstruction error can be expressed as the sum of two terms. The first proportional to the error associated with the ideal noiseless signal and the oracle solution. The second is proportional to the bound on the noise term. An example given in [13] states that if $\delta_{2k} = 0.2$, the C_0 =4.2 and C_1 = 8.5. Another numerical example found in [10] states that if $\delta_{2k} = 0.25$, than C_0 =5.5 and C_1 = 6.

(3) Orthogonal Matching Pursuit (OMP)

Orthogonal Matching Pursuit for recovering sparse signals discussed in [16] and provides a greedy or iterative algorithm for signal reconstruction. In contrast to BP, which attempts to converge to an overall solution, OMP solves a portion of the problem at each iteration. For very sparse vectors, this can provide a simpler and computationally less demanding reconstruction algorithm compared to BP. Following a similar constraint as given in RIP conditions for some fixed term, assumed to be $\delta \in (0, 0.36)$, and the number of measurements is chosen to be $m \geq C \cdot k \cdot \log\left(\frac{N}{\delta}\right)$, then if m vectors are drawn independently from a Gaussian Distribution, OMP can recover the k-sparse signal with probability exceeding $1 - 2\delta$. Note that C is some

positive constant term following $C \leq 20$ and N is the length of the sensed vector [16]. Other random matrices based on distributions such as Bernoulli Distributions are also applicable.

Successful recovery also depends on the coherence statistic given by

$$\mu(\theta) = \max_{j<k} |\langle \theta_i, \theta_j \rangle| \tag{21}$$

where θ_i, θ_j represent the i^{th} and j^{th} columns of $\boldsymbol{\theta}$. Applying Theorem 3.6 of [27], OMP can recover a k-sparse vector given that $k\mu < \frac{1}{2}$. Note that this result is also applicable to BP.

To recover the signal, OMP iteratively finds the locations of non-zero entries in the sparse vector, which in turn correspond to certain columns in the sensing matrix. These locations are also referred to as the support of the sparse vector. Each iteration picks the column with the strongest correlation, stores the result, then subtracts off that contribution and iterates. The resulting support is then used to reduce the overall sensing matrix into a sub matrix whose columns correspond to the support. The reader is referred to [16] for details of the algorithm.

Halting conditions can depend on either the assumed size of the support or a threshold on the resulting residual from each iteration. If the vector is known to be k-sparse, the algorithm only needs to be run k times. In an approximately sparse setting, running the algorithm k times should return the locations of the k largest coefficients of the sparse vector.

The advantages of OMP over BP lie in the speed of the algorithm and ease of implementation. Generally speaking, a similar probability of success can be achieved; however, the OMP algorithm is more straightforward to implement and, for very sparse vectors, faster. For vectors which are not particularly sparse, then OMP may be a bad choice due to the long computational

time needed to for orthogonalization, which increases quadratically with the number of iterations [16].

(4) Compressed Sensing Matching Pursuit (CoSaMP)

A variant of OMP, Compressed Sensing Matching Pursuit (CoSaMP) was developed as an OMP based algorithm for robust recovery in the presence of noise [23]. To find the locations of the largest coefficients of the sparse vector, CoSaMP uses an approach inspired by the RIP principle. One of the key assumptions of the associated proofs is that the isometry constant for a 4k-sparse vector is smaller than a bound given by $\delta_{4k} \leq 0.1$ when sensing a k-sparse vector. In comparison, RIP conditions dictate a bound on δ_{2k} for a k-sparse vector as seen in equations (**11**) and (**18**).

One of the key distinctions between the CoSaMP algorithm and the OMP algorithm [16], is that for a k-sparse vector, at each iteration a support estimate of size 2k is found, then merged via union with the pervious iterations support estimate. This can result in an overall support estimate of size 3k if no terms are shared common between the two sets. The combined set is later pruned to the k largest coefficients such that the support output is only of size k. In comparison, in the OMP algorithm no pruning is necessary since each iteration only adds one value to the growing support estimate. For details on the CoSaMP algorithm refer to [23].

The sparsity of the input vector must be known before running this algorithm, since it is an input to the algorithm. Alternatively, the algorithm could be run multiple times with a varying sparsity, selecting the best approximation based on the size of the resulting error term. While this obviously increases run time, it does so by a factor no worse than O(logk) and removes the requirement that the exact sparsity be known a priori [23].

(5) Other Algorithms

Other OMP-based greedy algorithms include Stagewise OMP (StOMP) [24] and Regularized OMP (ROMP) [28], which both attempt to reduce the computational burden. ROMP is shown to recover with high probability k-sparse vectors with $O(k \ln^2 N)$ random frequency measurements and is based on RIP [28]. Other iterative methods exist, such as Ordinary Matching Pursuit and Thresholding algorithms. For an overview of these and a comparison between BP and OMP, the reader is referred to [29].

5. Example: A Simple Sinusoid

A simple example of a signal which fits the CS model is a sinusoid. In the time domain, this signal is almost always non-zero; a dense representation. However, in the frequency domain, this same signal can be described ideally by a single tone; a sparse representation. Also, since the Fourier basis is generally incoherent with random matrices, both will be used as our measurement and transform bases. Figure 12 shows **x** in both the time and frequency domain, depicting the transformation between a dense vector in the time domain and a sparse vector in the frequency domain.

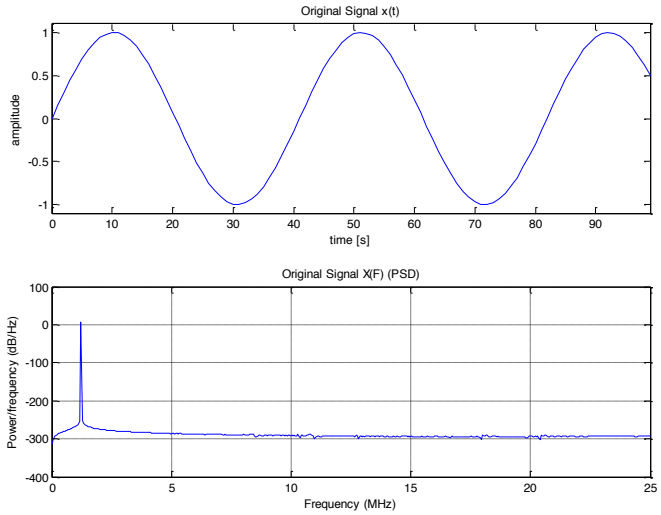

Figure 12 - vector x in time domain and frequency domain representations

For simplicity, a scenario with no noise is considered. Assume the signal of interest a sinusoid represented in the time domain by a vector **x** of length N. Our goal is to reconstruct this vector in time by only taking M << N measurements, as given in (**22**).

$$y = \Phi x \tag{22}$$

$$x = \sin(2\pi f t) \quad [Nx1] \tag{23}$$

It follows that the measurement matrix, **Φ**, must be of dimension (M x N) to reduce the size of the sensed vector to only M measurements. For the measurement operation is given by a matrix whose entries are ± 1 with equal probability, as given below.

$$\Phi_{i,j} = \begin{cases} +1 & (p = 0.5) \\ -1 & (p = 0.5) \end{cases} \tag{24}$$

The last step is to transform the basis of x from the time domain to the Fourier domain. Let **F** represent the (N x N) matrix form of an N point Fast Fourier Transform (FFT) with entries given by

$$\mathbf{F}_{i,j} = e^{-j2\pi/N(i-1)(j-1)} \tag{25}$$

and let **z** represent the (N x 1) vector which represents **x** in the Fourier domain, giving

$$z = \mathbf{F}x \quad \rightarrow \quad x = \mathbf{F^{-1}}z \tag{26}$$

To replace the vector **x** with its sparse representation **z**, the transform matrix is set to $\boldsymbol{\Psi} = \mathbf{F^{-1}}$ and substitute into (**22**).

$$y = \Phi x = \Phi\Psi z = \Phi\mathbf{F^{-1}}z = \theta z \tag{27}$$

To recover the unknown vector, the measurements **y**, along with the sensing matrix, **θ**, are passed into an L_1 solver such as CXV [30], which produces a reconstructed version of **z**, $\mathbf{z}_{rec}$. Figure 13 shows the original and reconstructed versions of **z**, reconstructed using M=256 measurements of total length of N=1024.

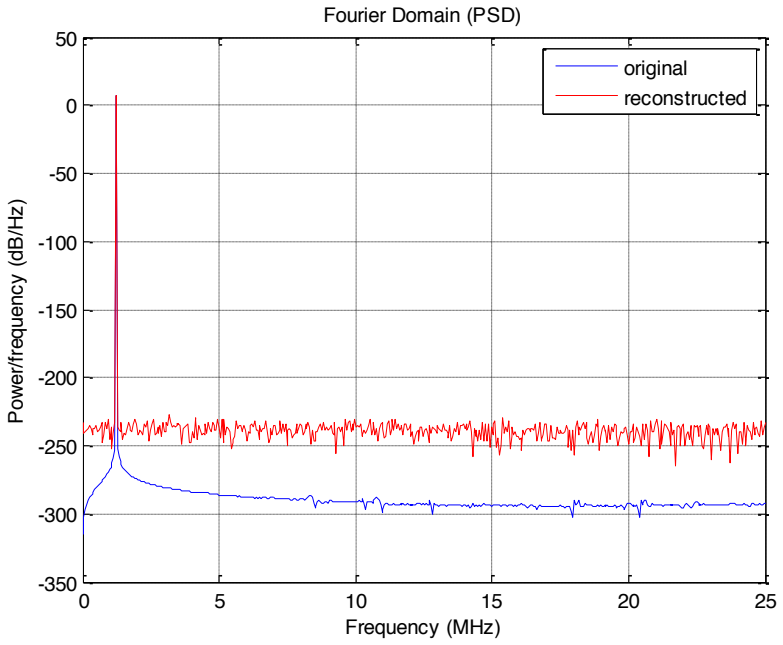

Figure 13 - Original and reconstructed versions of z (frequency)

The final step in recovering **x**, $\mathbf{z}_{rec}$ is multiplied with the transform matrix, which in our case is the inverse FFT.

$$x = \Psi z \quad \rightarrow \quad x = \mathbf{F^{-1}} z \tag{28}$$

Figure 14 shows the original and reconstructed versions of the vector x, which are indistinguishable after transformation. Thus it is shown that highly accurate reconstruction can be achieved despite taking only a fraction of the total frequency data during measurement. For a more detailed tutorial the reader is referred to [31].

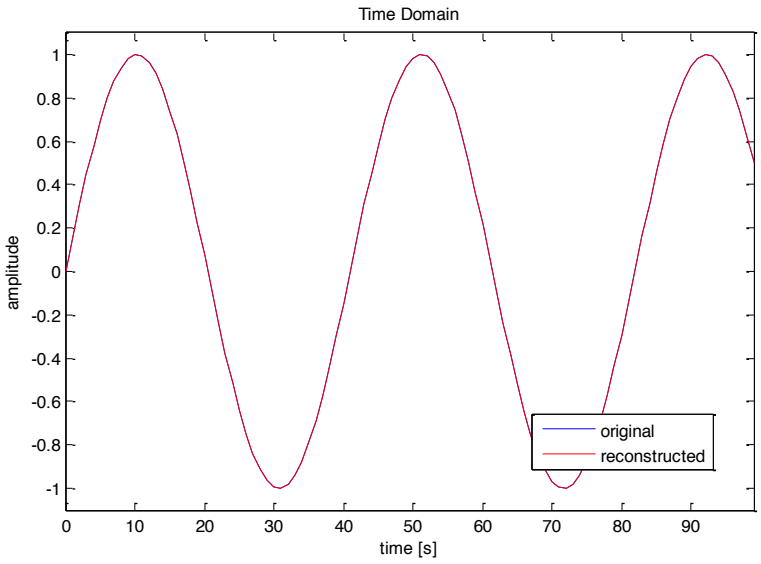

Figure 14 - Original and reconstructed versions of x (time)

C. Xampling and the Modulated Wideband Converter

Xampling is a framework for implementing Sub-Nyquist sampling systems which aim to alleviate requirements on system hardware and computational loads, particularly sampling rate [4]. The MWC [2] is a system which was designed with the Xampling framework in mind, along with CS techniques.

1. Multiband signal model

The MWC [2] [32] targets a signal scenario in which there are multiple narrowband signals, concurrently transmitted with different carrier frequencies. This is formally defined by the following.

1) It is assumed that for any portion of the signal, x(t), is contained in the multiband model, $\mathcal{M}$, such that all frequency components lie within the spectrum bounded by some f_{max}, such that $\mathcal{F} = [-f_{nyq}/2, +f_{nyq}/2]$. Here $f_{nyq} = 2f_{max}$ corresponds to the sampling frequency required for direct Nyquist rate sampling. The spectrum of x(t) is formally defined by

$$X(f) = \begin{cases} \int_{-\infty}^{+\infty} x(t)\, exp(-j2\pi f t)\, dt, & f \in \mathcal{F} \\ 0, & f \notin \mathcal{F} \end{cases} \tag{29}$$

2) There are no more than N, disjoint bands in $\mathcal{F}$. Additionally, N is even since each band corresponds to a real signal and therefore has conjugate symmetry
3) Each of the N bands has a bandwidth that does not exceed B

As long as these three criteria are met, the signal model is appropriate for treatment with the MWC. Note that there is no restriction on the location of each of the N bands, as long as none of the above criteria is violated. Table 1 summarizes the above signal model parameters. This

system is meant to treat signals whose actual spectral occupancy is relatively low, that is, NB << $2f_{max}$. For example if twice the total occupied bandwidth, 2NB, happens to be greater or equal to the Nyquist rate, then no rate improvement is possible [2].

Table 1 - Summary of Signal Model Parameters

Symbol	Definition
$\mathcal{M}$	Multiband model, signal x(t) is contained in this model
x(t)	Input signal to be sampled
X(f)	Continuous time Fourier Transform of x(t)
$\mathcal{F}$	$[-f_{nyq}/2, +f_{nyq}/2] = [-f_{max}, +f_{max}]$, wideband spectrum of input signal
N	Number of information bands in $\mathcal{F}$
$f_{nyq} = 1/T$	Nyquist rate of the overall signal x(t)

2. System overview

The MWC [2] [5] can be divided into two main blocks: an Analog Front End and a Digital Board. Each is described in more detail in the remainder of this section. Figure 15 depicts a high level block diagram of the system.

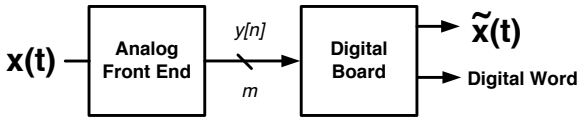

Figure 15 - High level block diagram of MWC

First the Analog Front End takes in input signal x(t) and splits it into m parallel channels, mixes it with a periodic sequence, passes it through a low pass filters and samples the mixture to produce samples, y[n]. Here, m corresponds to the number of channels present in the Analog

Front End, which directly corresponds to the number of rows in y[n]. The output of the Digital Board provides both a reconstructed version of x(t), denoted by a tilde, and a low rate, digitized version of x(t) which can lead into any standard DSP block [33]. The later is a result of an extension of the basic MWC configuration and the reader is referred to [5] and [33] for more details.

3. Analog Front End and Signal Sensing

a) Overview

The Analog Front End [2] [5] treats the signal x(t) by passing it to m parallel channels, mixing with a periodic sequence, low pass filtering the mixture and sampling at a relatively low rate. Figure 16 depicts a simplified, system level diagram of this portion of the system.

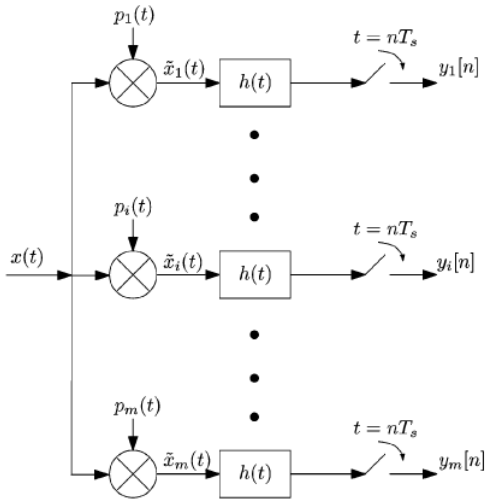

Figure 16 - Simplified version of analog front end of MWC [2]

First, the signal x(t) is mixed with a periodic signal, $p_i(t)$. The purpose of this mixing is to scramble the spectrum of x(t) and intentionally alias the signal such that all frequency components from $\mathcal{F}$ appear at baseband. Next, each channel is low passed filtered by h(t) which also provides anti-aliasing for the ADC, which is the last portion of the Analog Front End.

b) Periodic Mixing Sequence: $p_i(t)$

The periodic mixing sequence which is mixed with the Nyquist rate signal, x(t) and is periodic with period, T_p. A separate sequence is defined for each channel and $p_i(t)$ refers to the periodic mixing sequence of the i^{th} of m channels. After mixing, the periodicity of $p_i(t)$ provides the spectrum scrambling and aliasing, ensuring that all portions of the spectrum are aliased within $\pm 1/2T_p$ [4]. While periodicity is the only time domain requirement for the periodic mixing sequence, the resulting sensing matrix must satisfy certain properties for stable reconstruction using CS algorithms [14].

For the later numerical simulations and those presented in [2], $p_i(t)$ is chosen to be sign alternating function such that each entry is equal to +1 or -1 with equal probability. Within one period, this signal alternates M times which results in a clock rate of M/T_p. This signal is depicted in Figure 17. As seen in [5] the time domain shape of this signal is not important as long as periodicity holds. Mathematically, this signal is given by (**30**) as given in [2].

$$p_i(t) = \alpha_{ik}, \qquad k\frac{T_p}{M} \le t \le (k+1)\frac{T_p}{M}, \quad 0 \le k \le M-1 \tag{30}$$

However, since $p_i(t)$ is periodic, it can be represented by its Fourier Expansion. This is given in (**31**) and (**32**) and will be used to examine the frequency domain representation of the mixture of x(t) and $p_i(t)$.

$$p_i(t) = \sum_{l=-\infty}^{+\infty} c_{il} e^{j\frac{2\pi}{T_p}lt} \tag{31}$$

Where,

$$c_{il} = \frac{1}{Tp}\int_0^{T_p} p_i(t) e^{-j\frac{2\pi}{T_p}lt} dt \tag{32}$$

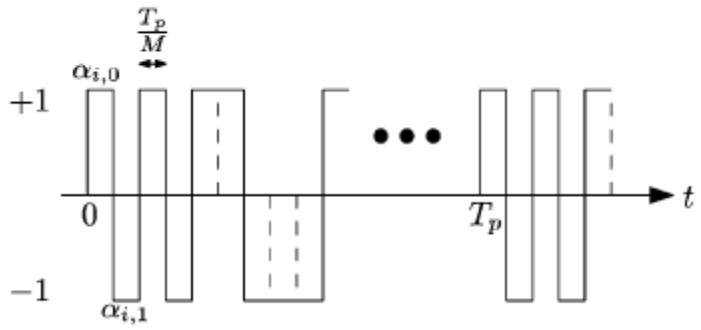

Figure 17 - Example of a periodic mixing sequence, $p_i(t)$ [2]

The frequency and hence period of this signal is not chosen arbitrarily. Essentially, the spectrum will be sliced in intervals of width $f_p = 1/T_p$, therefore, f_p is chosen to be greater or equal to B, or the maximum bandwidth of any band of interest. This ensures that each band is either wholly contained in a single slice or, at most split once [32]. L is defined as the number of spectrum slices which exist due to mixing with pi(t). If each slice has some arbitrary width f', then

$$L = \frac{f_{nyq}}{f'} \rightarrow f' = \frac{f_{nyq}}{L} = \frac{1}{LT_{nyq}} \tag{33}$$

If this f' is chosen to be less than B, it is possible that a single band, centered in one slice, will span three or more slices. However, if f' is limited to be greater or equal to B, then each band is only split at most once, if at all. The frequency of the periodic mixing function sets our width, thus, resulting in the relationship

$$f_p = \frac{f_{nyq}}{L} = \frac{1}{LT_{nyq}} \geq B \tag{34}$$

The number of alterations per period is also not arbitrary and is set by the following relation, given in (**35**).

$$M \geq 2\left\lceil \frac{f_{nyq}}{2f_p} + \frac{1}{2} \right\rceil - 1 \qquad (35)$$

This condition is required for exact blind spectrum recovery in the case that f_p is chosen to be minimum, that is $f_p = B$ [2]. Taking a simple approximation, the alternation rate of $p_i(t)$, given by M/T_p, must be equal or greater than Nyquist rate. This shows that although the ADC can sample at a low rate, f_s, the preceding mixer and low pass filter must be able to handle the Nyquist rate.

When the signal x(t) enters the MWC and is split into concurrent channels, ideally each channel should have a unique sequence, $p_i(t)$. Intuitively this maximizes the diversity of information about x(t) relayed in each mixed signal channel [5]. On the other extreme, if all channels had the same periodic sequence, then all channels past the first would be redundant and contribute no new information about x(t).

c) Mixer

The symbol $\tilde{x}(t)$ represents the product of analog multiplication of the input signal x(t) and the periodic function $p_i(t)$. Using (**36**) for the Fourier expansion of $p_i(t)$, Fourier Transform of this mixture is given below.

$$\tilde{x}_i(t) = x(t) \cdot p_i(t)$$

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

$$\tilde{X}_i(f) = \int_{-\infty}^{\infty} \tilde{x}_i(t) e^{-j2\pi ft} dt$$

$$= \int_{-\infty}^{\infty} x(t) \left(\sum_{l=-\infty}^{\infty} c_{il} e^{j\frac{2\pi}{T_p} lt} \right) e^{-j2\pi ft} dt$$

$$= \sum_{l=-\infty}^{\infty} c_{il} \int_{-\infty}^{\infty} x(t) e^{-j2\pi \left(f - \frac{l}{T_p}\right) t} dt$$

$$= \sum_{l=-\infty}^{\infty} c_{il}\, X(f - lf_p) \tag{36}$$

Effectively, the analog multiplication of x(t) and $p_i(t)$ results in linear combination of f_p shifted copies of the Fourier Transform of x(t), weighted by the Fourier coefficients of $p_i(t)$, c_{il}. Notice that this is not band-limited, therefore the low pass filter h(t) is needed to enable low rate sampling at the ADC. This mixing is also shown graphically in Figure 18.

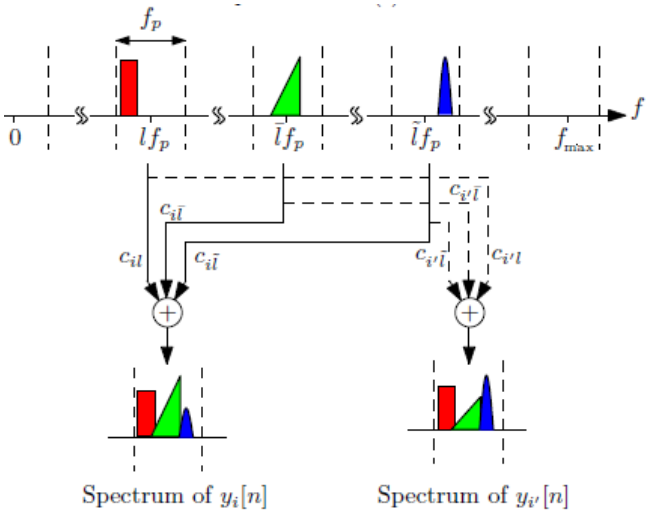

Figure 18 - Effect of mixing of x(t) with $p_i(t)$ and $p_{i'}(t)$ [33]

d) Low Pass Filter

The low pass filter treats the scrambled spectrum mixture, and acts as the anti-aliasing filter prior to the ADC. The cutoff of this filter corresponds to $1/2T_p$, which filters out unwanted spectrum contents, since all portions of the spectrum were assumed to be aliased into this band. Figure 19 provides a depiction of the frequency response of this filter. Since $f_s \geq f_p$, this filter also effectively provides an anti-aliasing filter for the sampling stage.

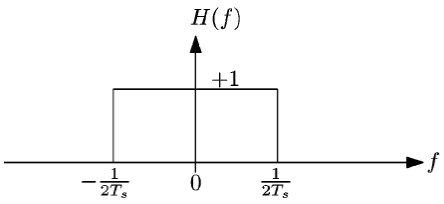

Figure 19 - Frequency domain representation of low pass filter h(t) [2]

Recall that the output of the filter given in (**36**) is not band-limited, however after filtering the summation limits are no longer infinite since it is assumed no signal exists past $f_p/2$. The output of the filter, $\widetilde{Y}_i(f)$ is given by

$$\widetilde{Y}_i(f) = \sum_{l=-L_0}^{L_0} c_{il}\, X(f - lf_p) \tag{37}$$

where L_0 is a constant chosen such that all spectral components within $\pm f_p/2$ are contained. This will be explicitly defined later.

e) ADC and sampling

The low pass filter h(t) ensures that the sampled output only contains frequencies below $f_s/2$, since the cutoff of h(t) is $f_p/2$ and $f_s \geq f_p$. Since the clock rate of $p_i(t)$, M/Tp, must run at or above Nyquist rate, this corresponds to a $f_p \geq f_{nyq}/M$, where again M is the number of alternations per period of $p_i(t)$. While f_s must be above this rate, this typically leads to $f_s << f_{nyq}$, since M is relatively large. For example, in [2], M = 195 is chosen.

One goal of the Xampling framework and MWC is to achieve the lowest sampling rate possible while still retaining the ability to reconstruct the signal exactly [4]. In [34], Landau presents theoretical minimum sampling rate if band locations and number of bands is known, with this rate being equal to the number of bands times the maximum bandwidth, NB [32]. Also in [32] the authors show that in a situation where the system is blind to the spectrum and band locations are unknown, then the minimum sampling rate becomes twice that set by the Landau Theorem or

2NB. In the sampling stage presented for the MWC, the overall sampling rate is given by the number of channels times the sampling rate of that channel, mf_s. In numerical simulations, f_s is set using its required relationship to f_p, while the number of channels is set to ensure that $mf_s \geq 2NB$. Given $f_s = f_p = B$, this results in the condition, $m \geq 2N$, however, it is later shown that the Continuous-to-Finite (CTF) Block requires $m \geq 4N$ and for use of CS algorithms, this condition is raised to $m \geq 4N \log(M/2N)$ [2]. This increases the overall sampling rate to more than twice the theoretical minimum, however, a sampling rate much lower than Nyquist is still achieved.

f) Other considerations

The mixer must handle the full bandwidth, and $p_i(t)$ must have a clock rate at least the Nyquist rate. Although there is no time delays needed, a prohibiting factor in high speed systems such as Time-Interleaved ADC, these components must work at very high speeds which can be difficult. Also, mismatch in delays between channels can degrade performance as seen in [6]. While, each ADC samples at a low rate, it also must resolve entire dynamic range of the mixture, $x(t) \cdot p_i(t)$.

g) Analog Front End Summary

To summarize, the signal x(t) was split into m different channels, where the i^{th} channel mixes x(t) with the periodic mixing sequence, $p_i(t)$. This mixture, which contains all portions of the spectrum weighted by the Fourier coefficients of $p_i(t)$ and is not band limited. However, aliasing ensures that all spectrum components will be present at baseband such that they appear within $\pm 1/2T_p$. T_p must be chosen such that no information band is split more than once by aliasing, which results in the condition $1/T_p \geq B$. This guarantees that sparsity associated with the sensed signal is at most 2N and is discussed later. The number of alternations per period, M, must satisfy the condition that the frequency of alternation, M/T_p, is at least the Nyquist rate of the wideband input signal. This mixture is then low pass filtered to band-limit the output of the

mixer and sampled at a low rate, $f_s \geq f_p$ with $f_s << f_{nyq}$. The output of this stage is m parallel channels with output $y_i[n]$ or the sampler output of the i^{th} channel. Table 2 provides a summary of system parameters as well as the corresponding requirements on their values.

Table 2 - MWC System Parameters and Requirements

Symbol	Requirement	Definition	Source
$p_i(t)$	periodic	Periodic mixing sequence to be mixed with input, aliases spectrum	[5]
T_p	$1/T_p \geq B$	Period of $p_i(t)$ ($f_p = 1/T_p$), ensure each band split no more than once	[32]
M	$M \geq 2\left\lceil \frac{f_{nyq}}{2f_p} + \frac{1}{2} \right\rceil - 1$	Number of alterations within one period of pi(t), Nyquist rate captures information	[2]
$f_s = 1/T_s$	$f_s \geq f_p \geq B$	Sampling rate of the ADC, must be larger than f_p to ensure all information aliased to base band by $p_i(t)$ is sampled	[2]
m	$m \geq N$ (NBSR) $m \geq 2N$ (BSR) $m \geq 4N$ (CTF) $m \geq 4N\log(M/2N)$ (CS)	Number of Channels. Various parts of the MWC system impose difference constraints on this value with the use of CS algorithms demanding the highest number of channels, since generally M is significantly larger than N	[2]
L_0	$L_0 = \left\lceil \frac{f_{nyq}+f_s}{2f_p} \right\rceil$-1	Sets length of sum in DTFT of uniform sequence $y_i[n]$	[2]
L	$2L_0+1$	Approximately the number of slices which spectrum is divided into, which is strictly set by Tp	[32]

4. Digital Processing and Signal Reconstruction

a) Overview

Once the Analog Front End processes the continuous time signal x(t), the m parallel channels of sampled output, $y_i[n]$, are passed to the Digital Board which represents the digital processing portion of the system [2] [32]. The main function of this board is reconstruction of the analog signal, but can also be used to provide low rate or baseband representation of the various information signals contained in x(t) [4] [5]. The first step in signal reconstruction is to determine the support of the spectrum, as defined previously. Here sparsity is determined by how

many of L total spectrum slices of length f_p contain portions of the input signal. The support corresponds to the locations of the input signal within this vector of length L. Once this support is found, reconstructing the signal is essentially the same problem faced if the carrier frequencies were known [32]. The DTFT of these signals, $y_i[n]$ can be found using (**38**).

$$Y_i\left(e^{j2\pi fT_s}\right) = \sum_{n=-\infty}^{\infty} y_i[n]e^{-j2\pi fnT_s}$$
$$= \sum_{l=-L_0}^{L_0} c_{il}X\left(f - lf_p\right), \quad f \in \mathcal{F}_s \tag{38}$$

This equation relates the known DTFT of the output of the ADC, $y_i[n]$, to the unknown X(f) and the unknown signal x(t). The summation in the DTFT is reduced from $\pm\infty$ to $\pm L_0$ due to the fact that $y_i[n]$ only contains frequency components from $\pm f_p/2$ or $\mathcal{F}_p = [-\frac{f_p}{2}, +\frac{f_p}{2}]$. L_0 is defined by (**39**).

$$L_0 = \left\lceil \frac{f_{nyq} + f_s}{2f_p} \right\rceil - 1 \tag{39}$$

Since the input to this stage is composed of several parallel channels, it is convenient to write his result in matrix form

$$\mathbf{y}(f) = \mathbf{A}\mathbf{z}(f), \quad f \in \mathcal{F}s \tag{40}$$

Where $\mathbf{y}(f)$ is a vector of length m, where the ith entry is equal to the DTFT of $y_i[n]$ or $Y_i(e^{j2\pi fTs})$. In this form, $\mathbf{A}$ represents the Fourier Coefficients of $p_i(t)$. The vector $\mathbf{z}(f)$ is an unknown vector which is related to the input signal as it contains the slices of spectrum $\mathcal{F}$ arranged in a vector. This vector has length $L = 2L_0+1$, since this corresponds to the total number of spectrum slices. This vector is shown in Figure 20 and equation for the i^{th} entry of this vector,

where each entry corresponds to a shifted slice of X(f), is given by (**41**). Note that the choice of $f_p \geq B$ results in each band contributing at most two non-zero entries to $\mathbf{z}(f)$ [2]. This sparsity is later leveraged by CS algorithms.

$$\mathbf{z}_i(f) = X(f + (i - L_0 - 1)fp, \quad 1 \leq i \leq L, f \in \mathcal{F}s \tag{41}$$

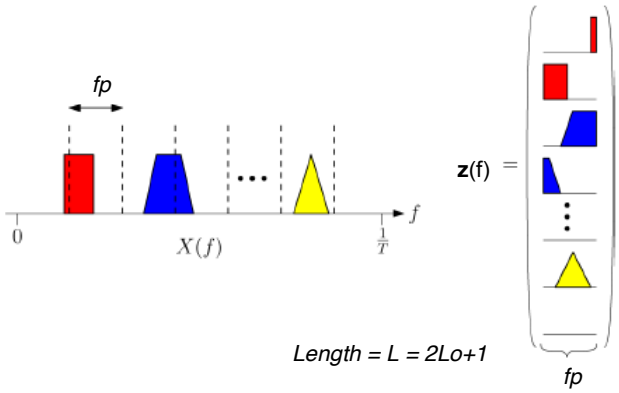

Figure 20 - Vector z(f) and its relation to spectrum on sensed signal x(t) (based on figure in [32])

A is a (m x L) matrix which contains the conjugate of the Fourier coefficients of $p_i(t)$ such that, the i,l[th] entry is given by (**42**).

$$\mathbf{A}_{il} = c_{i,-l} = c_{il}^* \tag{42}$$

Each entry c_{il}, can be calculated using (**43**).

$$c_{il} = \frac{1}{T_p}\int_0^{\frac{T_p}{M}} \sum_{k=0}^{M-1} \alpha_{ik} e^{-j\frac{2\pi}{T_p}l\left(t+k\frac{T_p}{M}\right)} dt$$

$$= \frac{1}{T_p}\sum_{k=0}^{M-1} \alpha_{ik} e^{-j\frac{2\pi}{M}lk} \int_0^{\frac{T_p}{M}} e^{-j\frac{2\pi}{T_p}lt} dt \tag{43}$$

A can also be decomposed into a multiplication of three matrices which explicitly relate back to the MWC system. This decomposition is given by (**44**).

$$\mathbf{A} = \mathbf{SFD}, \quad f \in \mathcal{F}s \tag{44}$$

Here, **S** is a m x M matrix which corresponds to the coefficients of the sign alternative function given by $p_i(t)$. Each entry $S_{ik} = \alpha_{ik}$ as defined in (**30**). Each channel's periodic mixing sequence contributes one row to **S**.

D is a (L x L) diagonal matrix with entries begin defined by dl. As defined in (**45**) the magnitude of d_l decays as $l > 0$. This is a direct result of the choice of sign alternating function for $p_i(t)$ [2].

$$d_l = \frac{1}{T_p}\int_0^{\frac{T_p}{M}} e^{-j\frac{2\pi}{T_p}lt}\,dt = \begin{cases} \frac{1}{M} & l = 0 \\ \frac{1-\theta^l}{2j\pi l} & l \neq 0 \end{cases} \tag{45}$$

F is a (M x L) matrix which holds L columns, where each column corresponds to a reordered column of the (M x M) matrix $\bar{\mathbf{F}}$, which is the Discrete Fourier Transform (DFT) matrix [2]. This matrix is defined by the following equations.

$$\theta = e^{-j2\pi/M} \tag{46}$$

$$\bar{\mathbf{F}}_i = \left[\theta^{0\cdot i}, \theta^{1\cdot i}, \dots, \theta^{(M-1)\cdot i}\right]^T \tag{47}$$

$$\mathbf{F} = [\bar{\mathbf{F}}_{\mathbf{L_0}}, \dots, \bar{\mathbf{F}}_{-\mathbf{L_0}}] \tag{48}$$

With these definitions in place, the coefficients of the sensing matrix **A** can be redefined in the terms given below.

$$c_{il} = d_l \sum_{k=0}^{M-1} \alpha_{ik}\theta^{lk} \tag{49}$$

In summary the relationship between the known vector **y**(f) and the unknown vector **z**(f) can be expressed using various parameters of the Analog Front End and is expressed in [2] (**50**).

$$\underbrace{\begin{pmatrix} Y_1(e^{j2\pi fT_s}) \\ Y_2(e^{j2\pi fT_s}) \\ \vdots \\ Y_m(e^{j2\pi fT_s}) \end{pmatrix}}_{\mathbf{y}(f)} = \underbrace{\begin{bmatrix} \alpha_{1,0} & \cdots & \alpha_{1,M-1} \\ \vdots & \ddots & \vdots \\ \alpha_{m,0} & \cdots & \alpha_{m,M-1} \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} | & \cdots & | & \cdots & | \\ \bar{\mathbf{F}}_{L_0} & \cdots & \bar{\mathbf{F}}_0 & \cdots & \bar{\mathbf{F}}_{-L_0} \\ | & \cdots & | & \cdots & | \end{bmatrix}}_{\mathbf{F}} \underbrace{\begin{bmatrix} d_{L_0} & & \\ & \ddots & \\ & & d_{-L_0} \end{bmatrix}}_{\mathbf{D}} \underbrace{\begin{bmatrix} X(f - L_0 f_p) \\ \vdots \\ X(f) \\ \vdots \\ X(f + L_0 f_p) \end{bmatrix}}_{\mathbf{z}(f)}$$

[2] (50)

Given the sensing matrix **A**, the problem of recovering unknown input signal involves finding the sparsest solution for the unknown vector **z**(f) for the system in (**40**) for every f ∈ $\mathcal{F}$s. This falls into the main problem addressed in mainstream CS and therefore these algorithms are applied in this stage. However, mainstream CS addresses problems where sparse vectors have a countable number of non-zero entries, while in the case of a continuous analog signal, there is an uncountable number of frequencies present for a finite bandwidth [32]. While the length of the vector **z**(f) has a finite length L, each spectrum slice contained in an entry has a continuous bandwidth. This leads to an Infinite Measure Vectors (IMV) problem, since $\mathcal{F}s$ has infinite cardinality. Thus the Continuous-to-Finite block reduces this IMV problem to a Multiple Measure Vectors (MMV) problem, which can be solved with CS algorithms.

b) Continuous-to-Finite (CTF) Block

The overall goal of the CTF block is to reduce the IMV problem for a MMV problem and solving for the spectral support [32]. The support of a vector (i.e. **w**) is the set supp(**w**), which denotes the indices of all non-zero entries of **w**. First a frame is constructed from the samples at ADC output. Then CS algorithms are applied to find sparest matrix corresponding the sensing matrix and the constructed frame. In [2] it is shown that solving for the support of the solution the CS problem can be used to recover the support, supp(z($\mathcal{F}$p)), where $\mathcal{F}\mathrm{p} = [-\frac{\mathrm{fp}}{2}, +\frac{\mathrm{fp}}{2}]$. A block diagram of the CTF can be found in Figure 21.

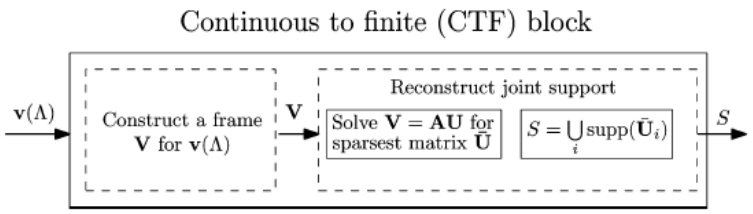

Figure 21 - Block Diagram of CTF Block, solving the joint support of u, S [2]

The first step of this block is to construct a frame, **V**, for the measurement set, y($\mathcal{F}s$), which is denoted in the block diagram as v(Λ). This is done by first computing **Q**, given by (**51**).

$$\mathbf{Q} = \int_{f\in\mathcal{F}_s} y(f)y^H(f)df = \sum_{n=-\infty}^{+\infty} y[n]y^T[n] \tag{51}$$

While theoretically this involves an infinite summation, y[n] has length m for a given n and, as shown in [4], rank(**Q**) ≤ m therefore m linearly independent vectors is. Note that the sparsity of **z**[n] at most 2N, and, generally, m > 2N, therefore m is a sufficient number of linearly independent vectors. Next this matrix **Q** is decomposed according to

$$\mathbf{Q} = \mathbf{V}\mathbf{V}^H \tag{52}$$

Where **V** is a matrix with size (m x rank(**Q**)) and has rank(**Q**) and orthogonal columns. Once **V** is found, the MMW problem in (**53**) is solved through CS sub-optimal algorithms to find the sparsest $\mathbf{U_0}$

$$\mathbf{V} = \mathbf{A}\mathbf{U} \tag{53}$$

Once the support of this vector is found, it can be applied to solve for the original signal x(t) as shown in [2]. Solving the support is not closed form when using sub-optimal algorithms, so there is a probability of success. However, once the support is found, the signal recovery becomes a linear, closed form problem.

An algorithm for the CTF is found in [32], which also has a flag to indicate successful recovery of the original spectral support. This flag is implemented by examining the cardinality of the spectral support set. As stated in [32], once the support set is found, there is essentially no difference between blind and non-blind reconstruction, since the signal locations in the spectrum are known.

c) CTF compared to Discretization [32]

An alternate method for reducing the IMV problem to a MMV is direct discretization of the frequency band. While this is more intuitive and more straightforward, it is shown in [32] that for a comparable run time, discretization yields significantly worse results. For comparable results, the grid in which the continuous band would have to be discretized to would be so large, that the run time would greatly exceed the CTF algorithms presented in the same work. Essentially, the CTF block allows the system to treat a window of time at once, reducing the computational complexity, yet still recovering the input signal support.

d) Stability of CS problem

Since the CTF employs a sub-optimal, CS based algorithm for solving the sparsest vector $\mathbf{U}_0$, there are certain theoretical requirements to ensure stable recovery of the support [14]. The main concern is the sensing matrix, **A**. According to mainstream CS literature, the Restricted Isometry Property (RIP) can be used to analyze the sensing matrix [14] [8]. A sensing matrix is said to have RIP of order k if there exists some $0 \leq \delta_K \leq 1$ such that (7) is true for every u which has at most k non-zero entries, or is k-sparse.

The authors of [14] point out that finding this isomtery constant is not an easy task. Additionally, they argue that mainstream CS measures of acceptable sensing matrices would result in excessive hardware implementations. Therefore they present an alternative measure, Expected

RIP or ExRIP, to measure the sensing matrix of the MWC. One of the basic assumptions of ExRIP is that the signal model is assumed to be random. Also, through numerical simulations they concluded that random sign matrices with uniform distribution, Maximal Length Sequences, and Gold codes all resulted in a sensing matrix which provided reliable recovery.

As discussed in [2] the number of channels in the MWC affects the stability of the CTF block. For uniqueness it is stated that every 4N columns of the sensing matrix **A=SF**, must be linearly independent. Ignoring **D** in this case is inconsequential and is also explained in this work. This rises the required number of channels for blind spectrum reconstruction from m=2N to m=4N. The use of CS algorithms raises this minimum number of channels to m=4N log(M/2N), since this is directly related to the number of measurements specified in mainstream CS research. While this does increase the overall sampling rate, mf_s, this still remains well below Nyquist rate if the parameters of the MWC are well chosen.

III. Prior Art

A basic Matlab model of the MWC can be found at [35] and can be used to perform simulations. This model was used as a basis for this work, and was extended to simulate various impairments under configurable system parameters.

In [5] a hardware implementation of the MWC is presented. The authors address system requirements regarding SNDR, gain, and the Noise Figure of each block. While this work does provide detailed analysis of the hardware constructed, they do so for a fixed hardware design.

In [2], simulations of the effects of ADC quantization are briefly presented. These results are presented through simulation and a brief discussion. In [36], the effects of filtering with an elliptic filter are investigated. A condition for ideal reconstruction is proposed, and a digital compensation method are proposed and validated in simulation. While these results are presented in the context of the MWC, there are no simulations regarding the effects in practice.

In [6], simulations of the effects of channel mismatches are presented. This is done for a fixed receiver design and compared to a Direct Conversion demodulator under the same impairments. These results are validated in this work and extended to consider system designs with various channel counts, as well as other impairments.

While the aforementioned works address issues of impairments in Xampling based receivers, no one work gives a comprehensive idea of their implications on system performance. This works presents preliminary results from a simulation tool, designed to allow various impairments and system parameters to be simulated simultaneously. The simulations presented in this work are an investigation into the relationship between impairment severity and overall system performance. Since the simulations used were extended from [35], a brief overview is provided.

A. MWC Simulation Overview

The Matlab model used for numerical simulations was adapted from the model posted at [35], with certain signal and system parameters were adjusted to match those presented in [2].

1. Signal Model

The signal model used for these simulations matches that found in Table 1, Option A in [2]. Namely the number of bands (N) is equal to 6, the maximum bandwidth (B) of a narrowband signal is 50MHz and the Nyquist frequency (f_{nyq}) is given as 10GHz.

A summary of the remaining system parameters can be seen in Table 3. Note that the overall rate is defined by mf_s. For 100 channels, this corresponds to an overall rate of 5GHz, which is still only ~50% of the Nyquist rate. Therefore, for the model parameters shown below, the MWC works effectively at a sub-Nyquist rate.

Table 3 - System parameters for numerical simulations [2]

$f_p = \frac{f_{NYQ}}{195} \approx 51.3$ MHz
$f_s = f_p \approx 51.3$ MHz
$m \geq 2N = 12$
$M = M_{\min} = 195$
$M_{\min} = L = 195$
Rate $mf_s \geq 615$ MHz

2. Input Signal

The input signal to this model is given by (**54**)

$$x(t) = \sum_{i=1}^{3} \sqrt{E_i B} sinc\big(B(t-\tau_i)\big) cos\big(2\pi f_i (t-\tau_i)\big) \tag{54}$$

Where $E_i = \{1, 2, 3\}$, $\tau_i = \{0.4, 0.7, .02\}$*1us, and f_i is drawn randomly from $[0, f_{nyq}/2]$. Figure 22 shows the resulting signal plotted in Matlab in the time domain with noise. This noise was generated using the `randn` function and was scaled according the SNR designated in the input.

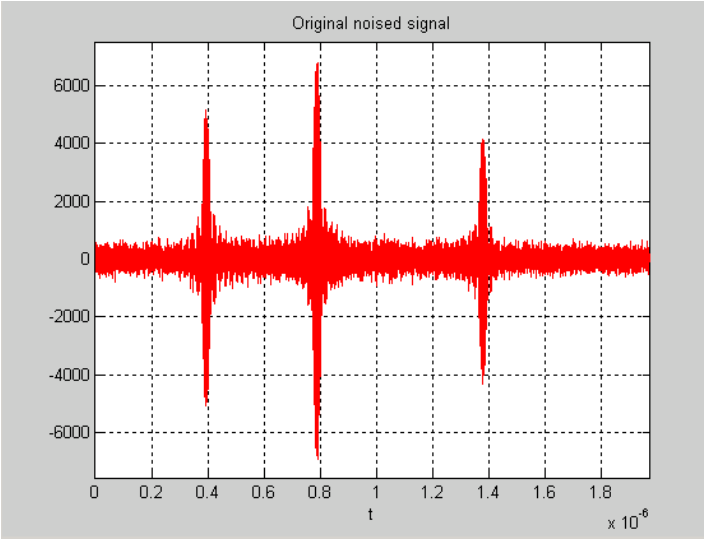

Figure 22 - Input signal in time domain, x(t)

Figure 23 shows the input signal in the frequency domain using the `fft` function. By visual inspection, it is observed that X(f) follows the narrowband, multiband model.

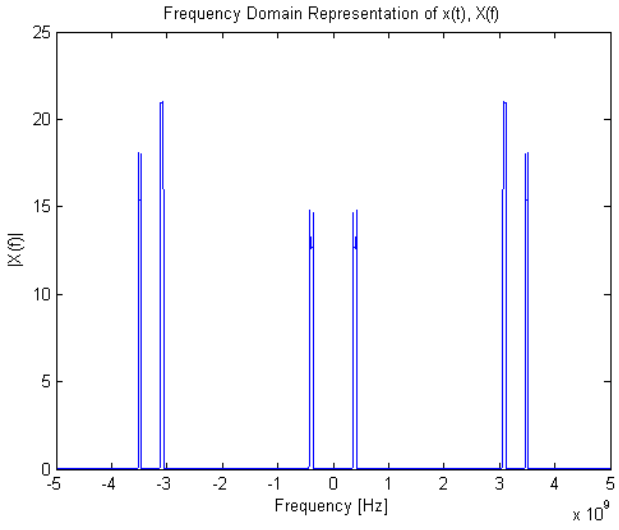

Figure 23 - Input signal frequency domain, X(f)

3. Periodic Mixing Sequences: $p_i(t)$

The periodic mixing sequences was chosen to be a sign alternating function, which alternated M times in one period Tp. This was implemented using the `randsrc` function to generate an m x M matrix whose i^{th} row contains $p_i(t)$.

4. Mixing

Mixing of the input signal x(t) and the periodic mixing function $p_i(t)$ is done by multiplication of the rows of each using the .* and iterating throughout all m channels.

5. Low Pass Filtering

To implement the ideal low pass filter, an array of zeros is created and the first element set to 1. This vector is then interpolated across the entire time axis using the `interpft` function. This effectively creates an ideal brick-wall filter with the cut off frequency set by the relative length of the original array. The FFT of the signal and the FFT of the filter are multiplied, and the inverse FFT of the product is taken. This performs the low pass filtering operation.

6. Sampling

This step implements ideal sampling by use of the `downsample` function.

7. Support Detection

Since the carrier frequencies are randomly assigned in the model during runtime and are fixed after initialization, the original support of the signal can be directly computed by iterating through the various carrier frequencies, f_i, and subtracting or adding B/2 to find the start and end of each band. This is stored and used for comparison to the recovered support to judge successful recovery.

8. Continuous to Finite Block

The matrices that make up **A**, **S F D**, are all defined in equation (50). Steps that are outlined in Section V-D and in the "SBR4" algorithm found in [32] are then used to find the matrix **Q**, construct the frame **V** and solve the sparest solution $\mathbf{U_0}$. Support of $\mathbf{U_0}$ is found using Orthogonal Matching Pursuit. Included are three variants of the OMP algorithms with the ability to reduce the number of iterations from 2N to N by forcing the selection of symmetric support and run an additional refinement step to the support estimate.

The resulting support is then tested against the original support and a decision is made on the success of recovery. Successful recovery is decided if both the original support is contained in the recovered support and the rank of the subset of **A** defined by the recovered support equal to the length of the recovered support vector [2].

9. Compressed Sensing Recovery Algorithms

Included in the demo code are two different functions which recover the spectral support through CS techniques. One function has the ability to be run in one of two modes, resulting in three total variations on the OMP algorithm. All variants require either the number of bands, N, or the assumed size of the signal support, 2N, as an input.

a) RunOMP_Unnormalized

This function takes six inputs: the frame computed by the CTF block, the overall sensing matrix A, the assumed size of the support 2N, a residual threshold, a normalized residual threshold and a true or false parameter which forces symmetric support selection. The function outputs the reconstructed signal, the recovered support, and both residuals. The algorithm is run until a halting criteria is reached, which include a limit on the number of iterations and a thresholding of the residuals computed at each iteration. The algorithm can be run in two different modes,

depending on the value of the last parameter. If true, then it is assumed that the signal has a symmetric support and adds two values to the support per iteration. In this mode, only N iterations are needed to return a support size of 2N. If this parameter is false, then no symmetry is assumed and the algorithm must run for 2N iterations to gather a support set of size 2N.

b) RunOMP_forMB

This function takes four inputs and outputs the recovered support. The inputs include the frame output by the CTF block, the sensing matrix, the number of signal bands, and a parameter specifying an additional number of iterations. As implemented, this last parameter does not affect the calculation of the support. Like `RunOMP_Unnormalized` in symmetric support mode, this algorithm only needs to iterate N times to produce a support of size 2N, since it selects the best location for a support estimate and the associated conjugate due to symmetry. However, this algorithm takes an additional step on the last iteration where it checks adjacent support values to refine the support selection.

It is important to note that OMP is used to recover the signal support, and being an iterative approach, the number of iterations is required as an input. While other halting criteria can be specified, generally the algorithm will iterate the specified number of times, returning a support of that size. Recall that the sparse vector z(f) may have at most 2N non-zero entries and as few as N. The sparsity of z(f) is equal to 2N when each band is split exactly once during aliasing and the sparsity is equal to N if none of the bands are split. However, it is given the bandwidth of each signal is set to be slightly less than the aliasing frequency, therefore it is likely that every band will be split and the true support will be of size 2N. Despite the occurrence of a “false detect”, signal reconstruction is still possible, however, the overall amount of noise in the reconstructed signal is increased.

B. Verifying Numerical Simulations

1. Simplifying the Mixing Stage

In [2] the authors proposed to reduce the amount of hardware complexity by reusing the random sequences generated in each channel. Here a new variable, r, is defined such that r defines the number of rows which have their $p_i(t)$ sequences drawn independently. If $m > r$ then the remaining $m - r$ channels have a periodic mixing function which is cyclically shifted from the sequence of another channel. That is the i^{th} row sees the periodic mixing function of the $(i-r)^{th}$ row. To achieve this, the `circshift` function was used. It is expected that with a decreasing r, the probability of success also decrease, since the relative independence of each channel decreases the more each random sequence is reused. Intuitively, a low r corresponds to a lack of diversity in the channels and can be thought of as a decrease in the effective number of channels. The number of independent channels is directly related to the ability to recover the support, as discussed earlier.

Simulations were run for various values of r, ranging from 100 to 4 and swept for several values of m, the number of channels. To calculate the probability of success, 100 trials were run for each data point with the total number of successful recoveries corresponded to the percent chance of success. Results of these simulations can be seen in Figure 24, where a SNR of 10dB was used for x(t). Figure 25 shows the same both, but for a SNR of 25dB. In both figures, the number of channels, m, appears on the x-axis, while the number of successful trials out of 100 appears on the y-axis.

In both cases, the prediction is accurate, that is, the lower the r the lower the probability of success. In the SNR=10dB case it can be seen that for an r as low as 20, there is a small degradation in performance, and about a 90% probability of success can be seen for m=50

channels. In the 25dB case there is even less degradation, with the curves corresponding to r=100, 50 and 20 overlapping.

Also in both cases, there is a steep drop in performance for r < 20. Recall that the number of channels, for a fixed f_s, is required to be $\geq$ 2N for blind support recovery, $\geq$ 4N for the CTF and $\geq$ 4Nlog(M/2N) for the CS problem. For the simulation parameters this corresponds to 12, 24 and ~29 respectively. Although for a reduced r, channels are not directly duplicated, there is still a reduction in the independence of the resulting sensing matrix.

This experiment was run to verify the results presented in [2]. Overall, the results matched well, with the results for an input SNR of 25dB, as seen in Figure 25, matching better than those for an input SNR of 10dB, as seen in Figure 24. Note that r greater than m simply means that all m channels were generated independently.

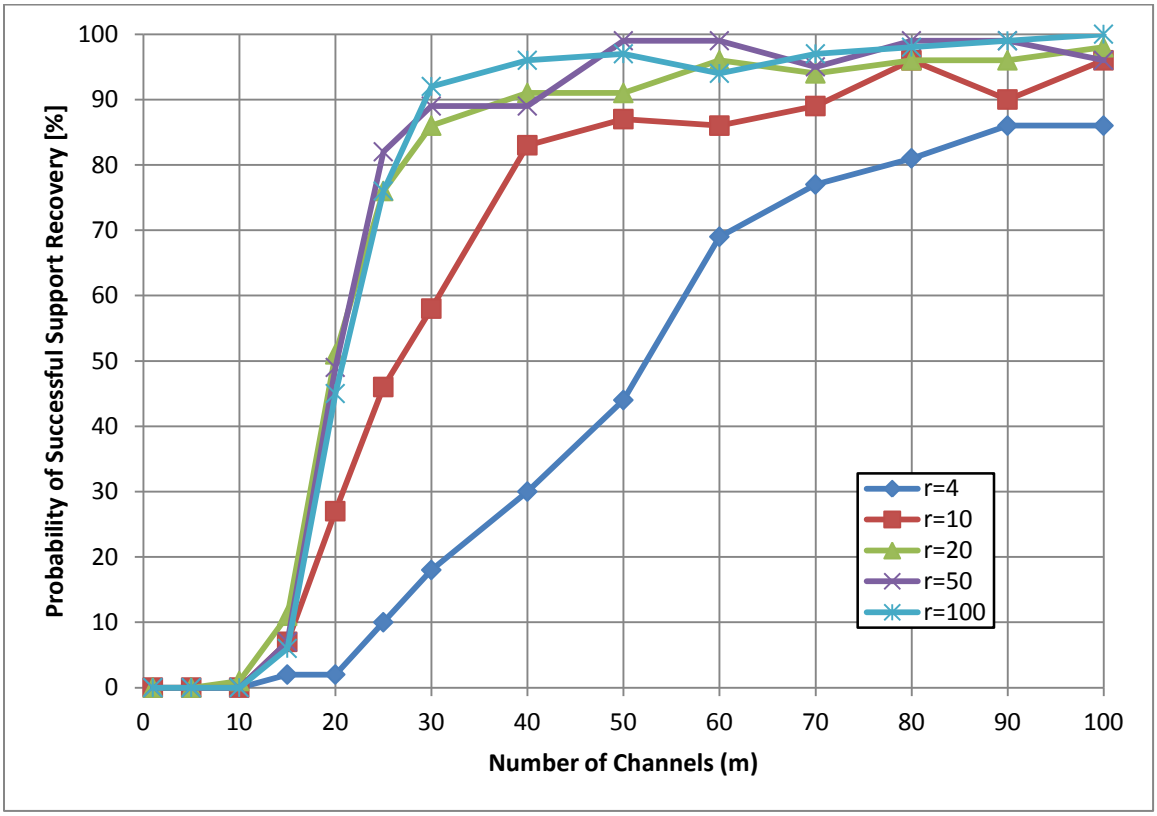

Figure 24 - Support recovery success probability for various numbers of channels, SNR=10dB

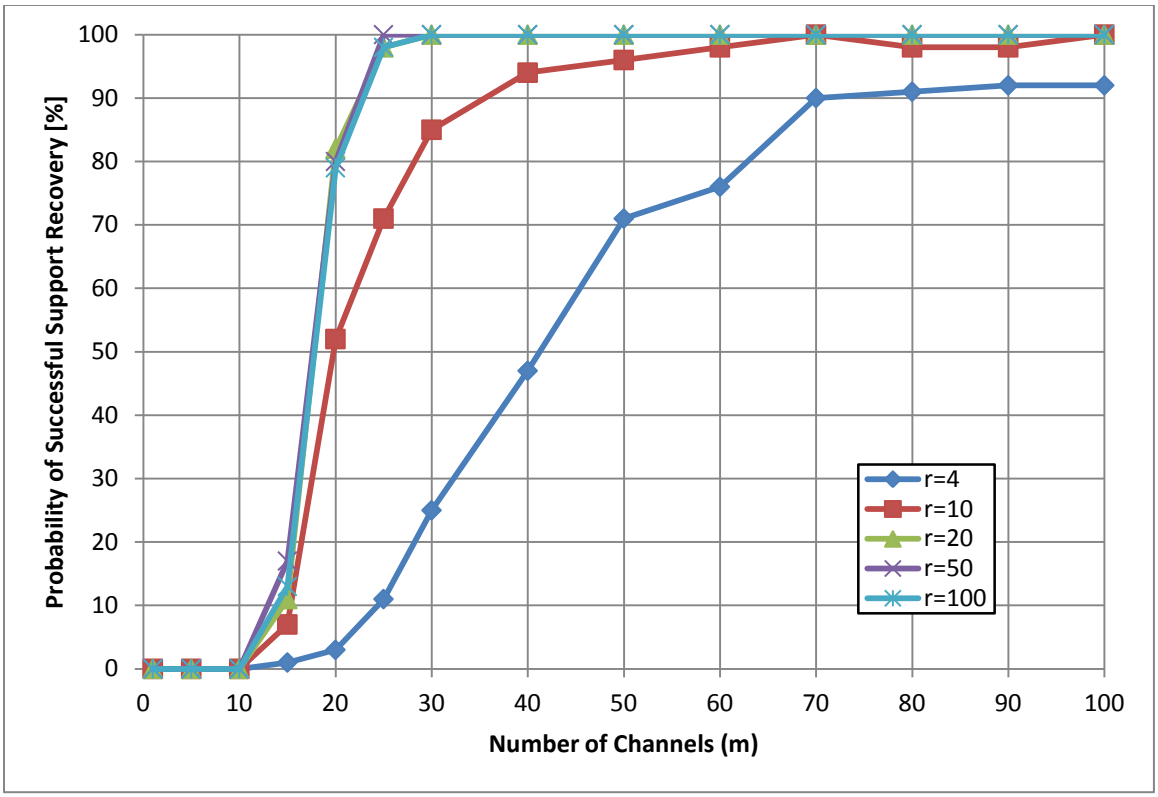

Figure 25 - Support recovery success probability for various numbers of channels, SNR=25dB

2. Signal Strength and Recovery

Another result from [2] relates the probability of successfully recovering the support with the input signal SNR. In this simulation, each channel had an independently drawn periodic mixing sequence, which was applied to an input signal x(t) with various SNRs. Results can be seen in Figure 26. As long as the SNR is above 10dB, the expected recovery rate is at least 90% for as few as 30 channels, which matches in [2]. This also matches the proposed lower limit on the number of channels, since a steep drop in performance is observed for channel counts under 30.

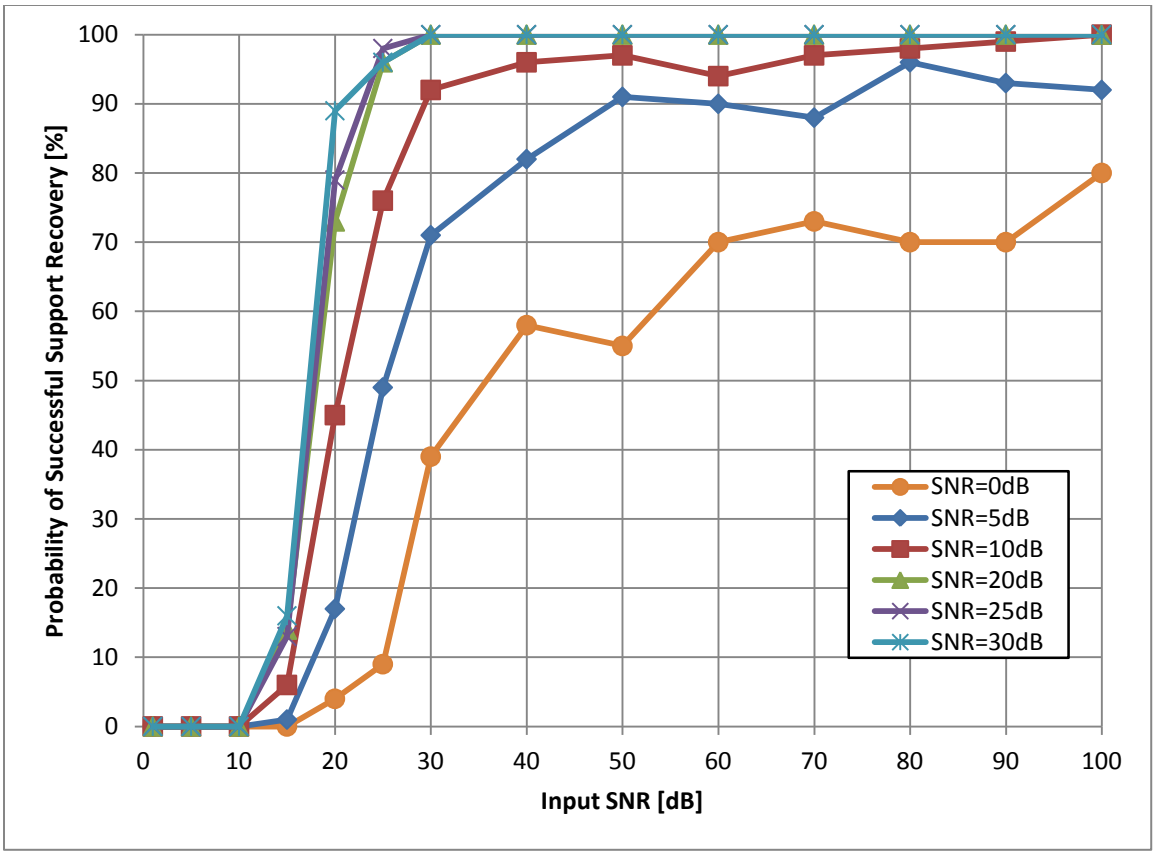

Figure 26 - Support recovery success probability for various numbers of channels, various SNR

IV. Simulation and Analysis of Impairments in Xampling Based Receivers

The MWC system and the Xampling framework have been discussed. However, before systems like these can be realized as practical sampling systems, implementation challenges such as hardware impairments and their implications must be studied. This work studies four topics related to the challenges of realizing these systems, which include simulation of hardware impairments of each channel and their effects on reconstruction, performance tradeoffs between different CS reconstruction algorithms, input signal energy dynamic range, and implications of impairments in the design of a 2.4GHz receiver.

A. Overview of Modeling Hardware Impairments

Understanding how impairments in hardware implementations of system components affect system performance is a critical step in implementing Xampling based receivers. The mixer, low pass filter and ADC found in each branch of the system were studied with various impairments and their effects on overall system performance evaluated. The following serve as a block level diagram for simulations of hardware impairments.

1. Mixer Model Impairments

The mixer is the first component found in the signal path of the receiver, as seen in Figure 16. This component must handle the full input signal amplitude and bandwidth. The impairments which were simulated in this block were delay mismatch in the mixing sequence between channels, a non-linear gain block and additive noise at the output, all of which are depicted in Figure 27.

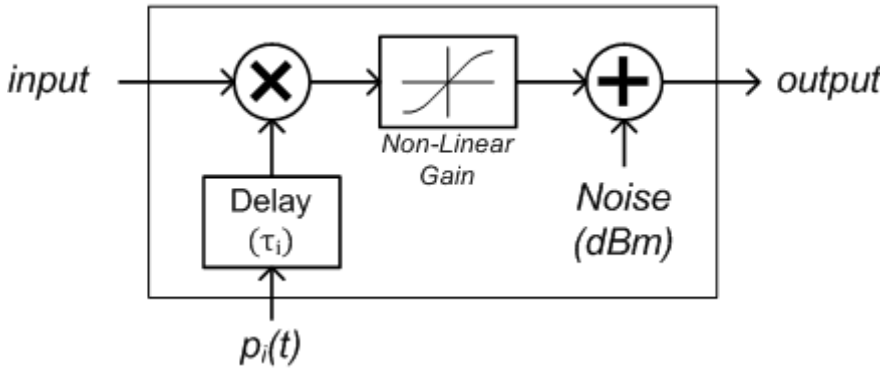

Figure 27 - Block diagram for mixer with simulated impairments

2. Filter Impairments

The low pass filter immediately precedes the mixer and is intended to suppress all components of the mixer output which are outside of the bandwidth dictated by the frequency of the periodic mixing sequence. The impairments which were simulated for this block include a random gain mismatch between system channels, a non-linear gain block and additive output noise, as depicted in Figure 28. Additionally, the various filter types were simulated, including, Butterworth, and the ideal brick-wall discussed earlier.

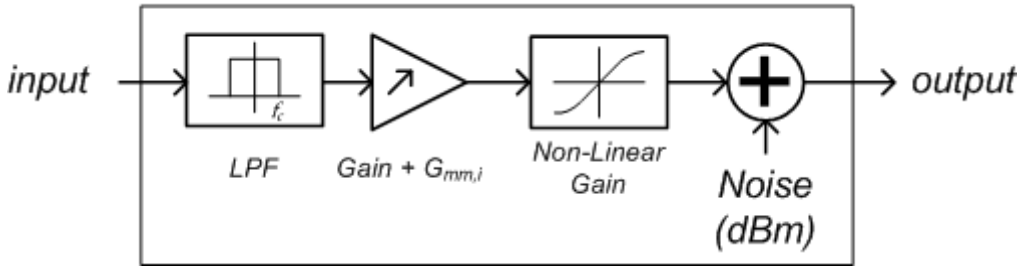

Figure 28 - Block diagram of low pass filter with simulated impairments

3. Analog to Digital Converter Impairments

The Analog to Digital Converter is the final analog block in the signal path. The sampled output of this block serves as the input to the digital portion of the system, which includes the CS

problem used to solve the signal spectral support. The impairments of this block which were simulated include a non-linear gain block, a quantizer and additive output noise, as depicted in Figure 29.

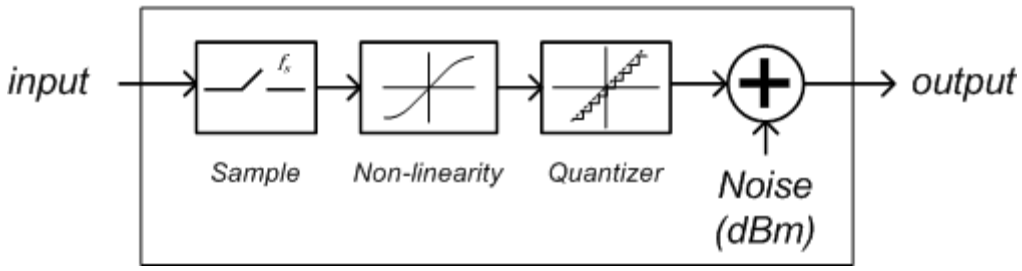

Figure 29 - Block Diagram for ADC with simulated impairments

B. Channel Hardware Impairments and Reconstruction

When the carrier frequencies are unknown prior to sampling, the MWC is dependent on recovering the spectral support through CS techniques. This means that regardless to how the signal may be affected by hardware impairments, accurate and reliable support detection is necessary to sample the signal.

In this experiment, various hardware impairments were simulated for a system designed for minimal sampling rate, as given in [2] and summarized in Table 4. These impairments include additive output noise, non-linearity, non-ideal filters, and ADC quantization. The input signal was defined by (**54**) and the carrier frequencies were chosen at randomly below half the defined Nyquist rate of 10GHz. Simulations were performed for various numbers of channels and an input SNR of 10dB and 25dB. Each impairment was simulated at various levels of severity and the probability of successful recovery of the signal support was calculated. The purpose of this experiment was to study how impairments in channel hardware affect the system's ability to successfully recover the support of the input signal. The results show if a system is designed with a sufficient number of channels, it can tolerate a certain degree of impairment and still recover the signal support with high probability.

Based on the requirements summarized in Table 4 and given in [2], this signal model is expected to require at least 30 channels for reliable recovery, therefore simulation results for systems with 100, 50, 30 and 20 channels are presented. Channel counts of 100 and 50 represent an overdesigned system; a channel count of 30 represents an optimally designed system and a channel count of 20 represents an under designed system. Also, the maximum signal bandwidth, frequency of the mixing sequence, and sampling frequency are set equal to each other, which reflects the minimal sampling rate for a given maximum signal bandwidth.

Table 4 - System parameters for hardware impairment and reconstruction simulations

N	6
fnyq	10 GHz
fc	random
B	51.2 MHz
fs	51.2 MHz
fp	51.2 MHz
L	195
M	195
Minimum Channel Requirements	
2N	12
4N	24
4Nlog(M/2N)	30

1. Mixer Impairments

The first component in the Analog Front End is the mixer, which is used to mix the input signal, x(t), with the periodic mixing sequences, $p_i(t)$, where 'i' denotes the i^{th} channel.

a) Additive Output Noise

The ideal output of the mixer is given by (**36**). The addition of noise at the output introduces an additive noise term, n(t) and N(f), as seen in (**55**) and (**56**).

$$\tilde{x}_i(t) = x(t) \cdot p_i(t) \rightarrow \tilde{x}_{i,noised}(t) = x(t) \cdot p_i(t) + n(t) \tag{55}$$

$$\tilde{X}_i(f) = \sum_{l=-\infty}^{\infty} c_{il} X(f - lfp) \rightarrow \tilde{X}_{i,noised}(f) = \sum_{l=-\infty}^{\infty} c_{il} X(f - lfp) + N(f) \tag{56}$$

This noise term lowers the overall SNR and is expected to degrade the system's ability to correctly recovery to signal spectral support. Figure 30 shows how the probability of successful

reconstruction is degraded by the addition of noise (dBm) for an input SNR of 10dB. Figure 31 shows the same plot for an input SNR of 25dB.

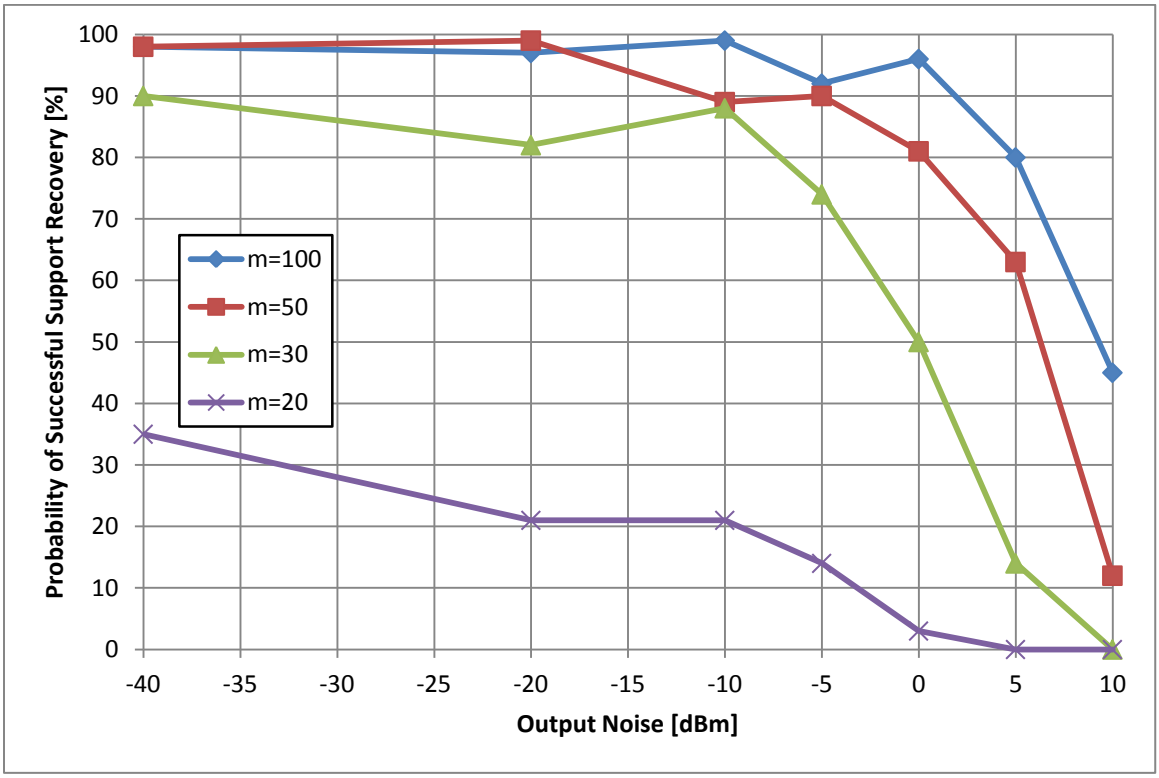

Figure 30 - Mixer output noise (dBm) and probability of successful support recovery, SNR=10dB

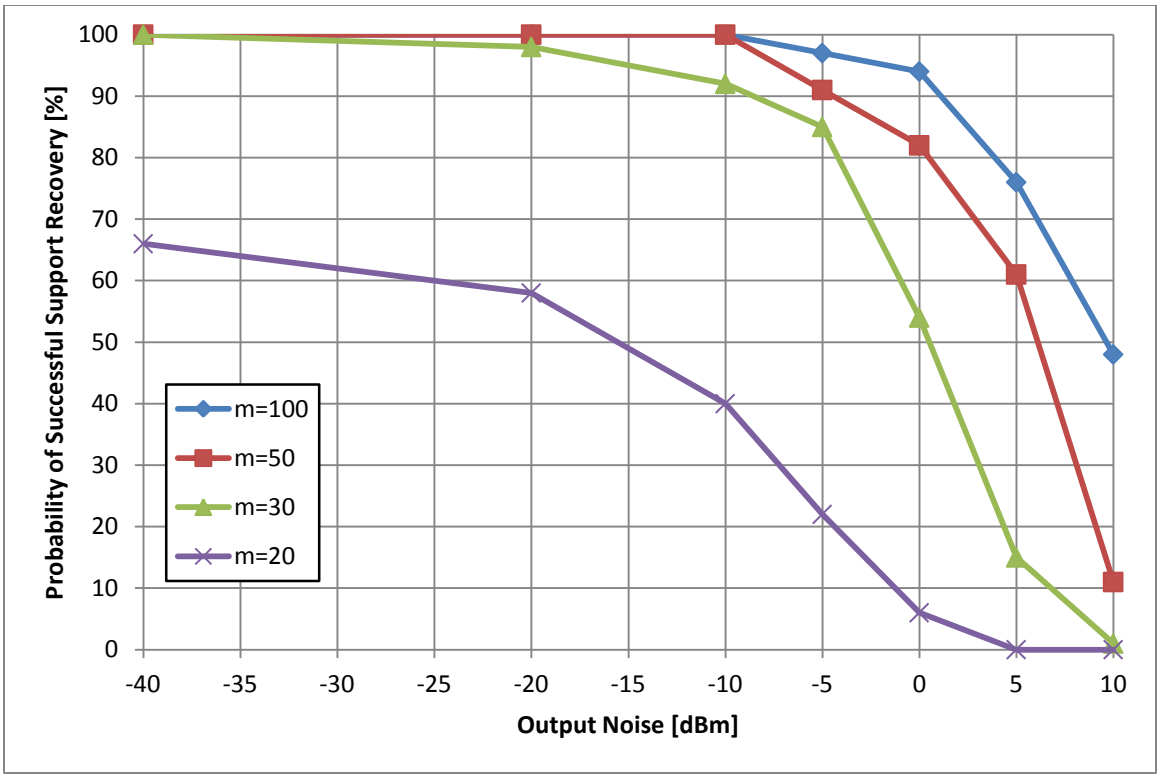

Figure 31 - Mixer output noise (dBm) and probability of successful support recovery, SNR=25dB

As expected, when the input SNR is higher, reconstruction is more robust to the additional noise; however, a steep drop in probability of success for an added noise of 0dBm or more. Also as expected, a higher channel count improves reliability, however, once the theoretical threshold of 30 channels is crossed the benefit per additional channel decreases. For example, for an input SNR of 25dB and an added noise of 0dBm, a 20 channel system has only a 6% chance of success. This probability increases to 54% for a 30 channel system, the theoretical minimum according to Table 4. This is an almost 10x improvement in performance for a 50% increase in channel count. In contrast, a 50 channel system has a 82% probability of success, while a 100 channel system has a 94% probability. This means that for a 100% increase in channel count, the improvement in performance is less than 1.25x.

b) Non-Linearity

Since the mixer must deal with the full scale signal, non-linearity near full scale is a significant impairment. The Rapp [37] model is used to model the effect of non-linearity in the signal amplitude and is characterized by a smoothness factor, r. The governing equation is given by (**57**).

$$x_{non-linear} = \frac{x}{\left(1 + \left(\frac{x}{V_{full\ scale}}\right)^{2r}\right)^{\frac{1}{2r}}} \tag{57}$$

Here $V_{full\ scale}$ represents the signal full scale voltage which is + 0.5V. This non-linearity distorts the time domain equation given in (**55**). Substituting $p_i(t)$ for the sum of its complex weighted terms, the non-linear term can be simplified to an unknown term α, as shown in (**60**).

$$\tilde{x}_i(t) = x(t) \cdot p_i(t) \rightarrow \tilde{x}_{i,non-linear}(t) = \frac{\tilde{x}_i(t)}{\left(1 + \left(\frac{\tilde{x}_i(t)}{V_{full\ scale}}\right)^{2r}\right)^{\frac{1}{2r}}} \tag{58}$$

$$\tilde{x}_{i,non-linear}(t) = \sum_{l=-\infty}^{\infty} \frac{1}{\alpha} \cdot x(t) c_{il} e^{j2\pi(lf_p)t} \tag{59}$$

$$\alpha = \left(1 + \left(\frac{1}{V_{full\ scale}} \cdot x(t) \left(\sum_{l=-\infty}^{\infty} c_{il} e^{j\frac{2\pi}{T_p}lt}\right)\right)^{2r}\right)^{\frac{1}{2r}} \tag{60}$$

The ratio $\frac{x(t)}{V_{full\ scale}}$ can be at most unity; however, generally the signal amplitude is well below the full scale value. This behavior can be observed from the time domain plot of the un-normalized input signal in Figure 22. Also if it can be assumed that the summation of the complex weighted terms of $p_i(t)$ are relatively small and that the smoothness factor is relatively high, than it is expected that $\alpha \approx 1$ with little corruption of the signal.

Figure 32 and Figure 33 show the effect of this non-linearity on the probability of successful support recovery as a function of the smoothness factor r for an input SNR of 10dB and 25dB respectively. As shown in (**57**), the amount of non-linearity is inversely proportional to the smoothness factor.

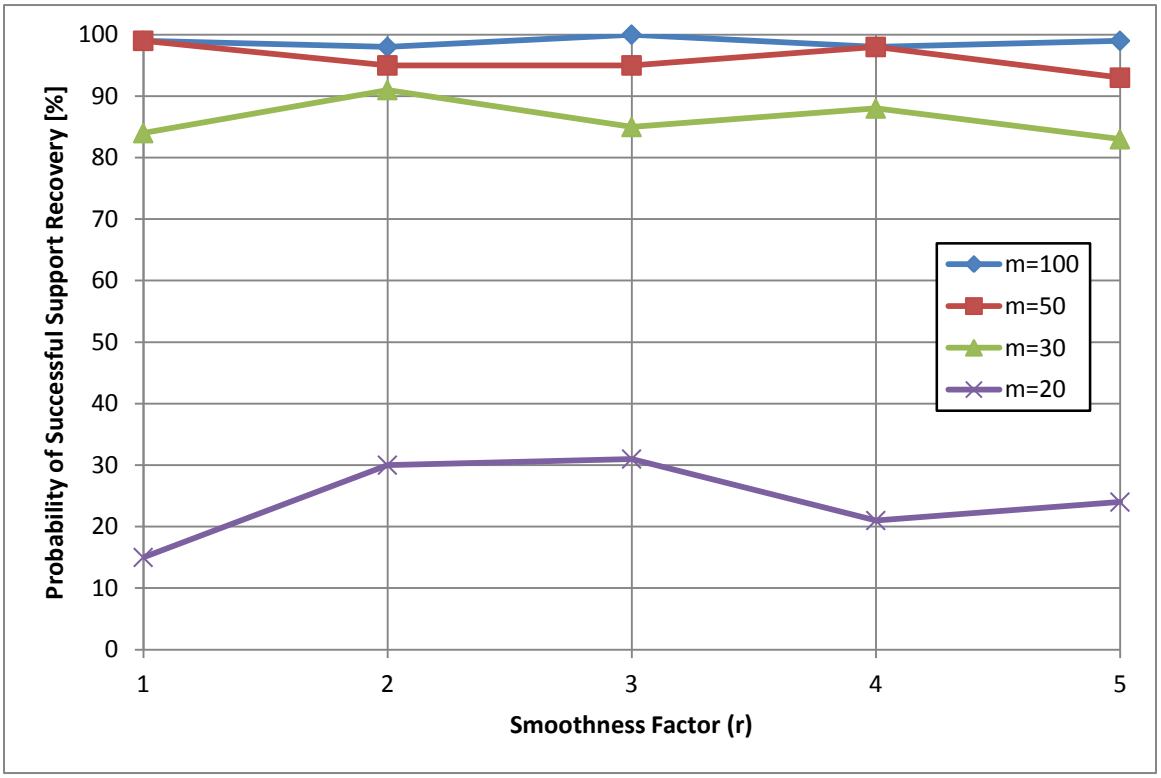

Figure 32 Mixer non-linearity and the probability of successful support recovery, SNR=10dB

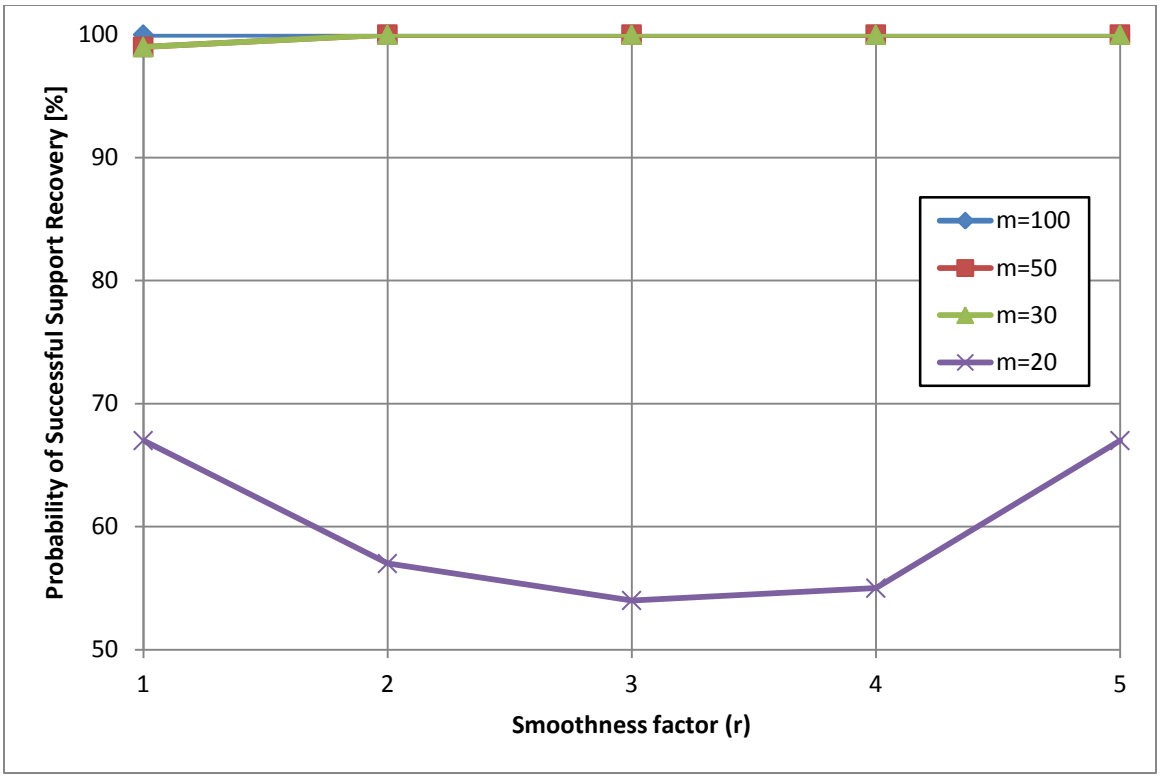

Figure 33 - Mixer non-linearity and the probability of successful support recovery, SNR=25dB

These results show that for channel counts above or equal the theoretical minimum of 30 that non-linearity in the amplitude has little effect on the probability of reconstruction. This agrees with the assumption that the non-linearity factor $\alpha \approx 1$ and that the distortion of $\tilde{x}_i(t)$ and subsequently $\widetilde{X}_i(f)$ is relatively small.

2. Low Pass Filter Impairments

The low pass filter following the mixer is a key component in the Analog Front End as it is used to remove unwanted spectral content, as well as, acting as an anti-aliasing filter for the ADC. This is especially important since the MWC intentionally spreads the signal throughout the full bandwidth and assumes perfect brick-wall filter behavior when formulating the mathematical signal analysis. The in-band portion of the signal can be corrupted by output noise, non-linearity and aliasing due to non-ideal filter roll off.

a) Additive Output Noise

The spectrum after filtering is expected to follow (**37**), such that the infinite summation is reduced such that only frequencies within $\pm$ $f_s/2$ are included. Adding a white noise term introduces a term which is not band limited, resulting in the following equations for the filter output $\tilde{y}_i(t)$ in the time domain and $\tilde{Y}_i(f)$ in the frequency domain, where $h(t)$ is the filter response in the time domain and is assumed to be an ideal brick-wall filter.

$$\tilde{y}_{i,noised}(t) = h(t) * x(t) \cdot p_i(t) + n(t) \tag{61}$$

$$\tilde{Y}_{i,noised}(f) = \sum_{l=-L_0}^{L_0} c_{il}\, X(f - lf_p) + N(f_n) \qquad f \in \mathcal{F}_s,\ f_n \in \mathcal{F}_\infty \tag{62}$$

This is expected to raise the noise floor and lower the SNR, degrading the probability of success. Figure 34 and Figure 35 show the probability of successful support recovery as a function of added output noise given in dBm for an input SNR of 10dB and 25dB, respectively.

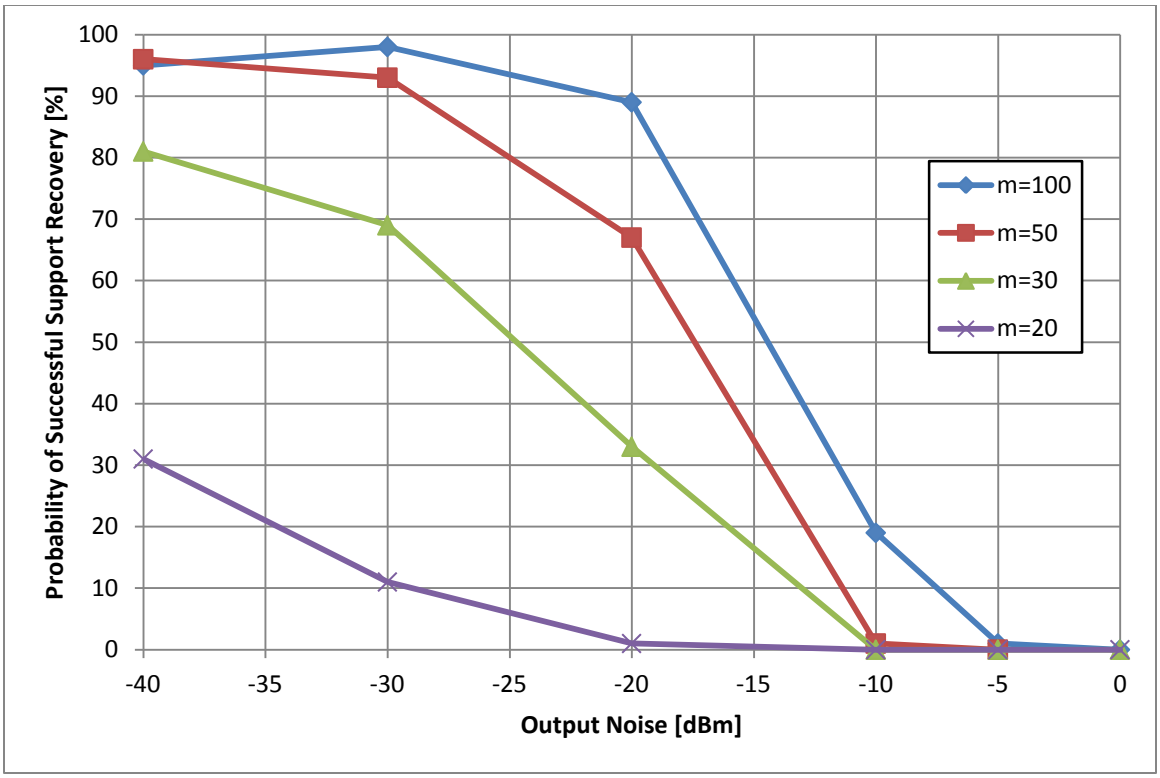

Figure 34 - LPF output noise (dBm) and probability of successful support recovery, SNR=10dB

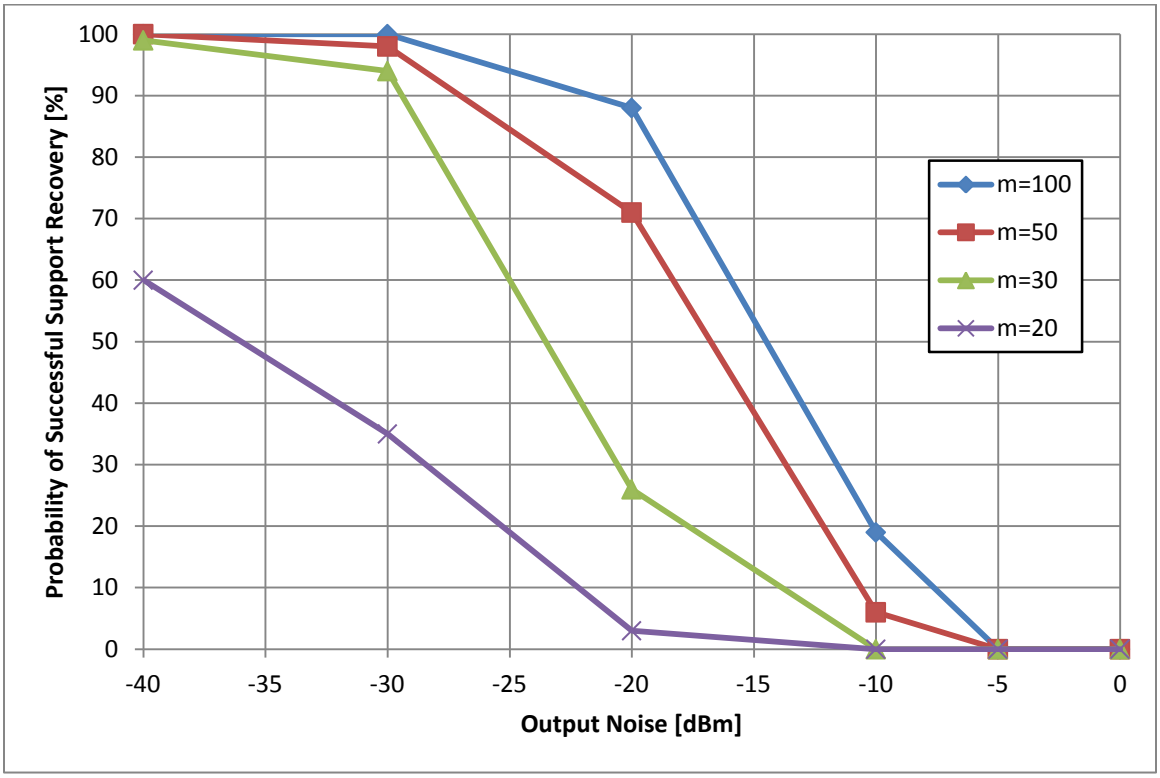

Figure 35 - LPF output noise (dBm) and probability of successful support recovery, SNR=25dB

These results show a similar behavior as those for the output noise of the mixer, namely, a higher SNR input signal is more robust to added noise and a steep drop in probability of success can be seen for a certain noise power across all channel counts. Unlike the mixer however, the signal at this point is much more sensitive to the amount of added noise power. For example, at the mixer output, a steep drop can be seen starting at a noise power of around 0dBm. In contrast, a noise power of only -20dBm is needed at the output of the filter to see a severe degradation in performance. This is likely due to the combination of the spectrum spreading which occurs during mixing and low pass filtering. Recall that mixing with the periodic mixing sequence, whose entries are randomly ± 1, spreads the signal energy throughout the spectrum. Low pass filtering then retains only a small portion of the mixture around baseband, reducing the overall remaining signal energy. Only a fraction of the signal energy, proportional to the ratio of the filter bandwidth and the Nyquist frequency, is retained after filtering. Therefore, it is expected that a proportionally small amount of noise power after the filter will cause the same SNR degradation as a given amount of noise power before the filter.

b) Non-Linearity

Since the filter also deals with the full scale signal, amplitude non-linearity is also expected to degrade the probability of success. The Rapp model is also used to model this non-linearity. Equation (**63**) shows the filter output while taking into a non-linear component, again, given by α (**64**).

$$\tilde{y}_{i,non-linear}(t) = \frac{1}{\alpha} \cdot h(t) * x(t) \cdot p_i(t) \tag{63}$$

$$\alpha = \left(1 + \left(\frac{1}{V_{\text{full scale}}} \cdot h(t) * x(t) \cdot p_i(t)\right)^{2r}\right)^{\frac{1}{2r}} \tag{64}$$

Here the product of x(t) and $p_i(t)$ are multiplied by the time domain expression for the filter h(t). Since the filter has unity gain and will be removing a large portion of the signal energy, it can be assumed that $h(t) * x(t) \cdot p_i(t) \leq x(t) \cdot p_i(t)$. Therefore the assumptions stated for the case of non-linearity of the mixer also hold and it is assumed that amplitude non-linearity will have little effect on the probability of success. Figure 36 and Figure 37 show, for an input SNR of 10dB and 25dB, the probability of success as a function of the smoothness factor associated with the Rapp model.

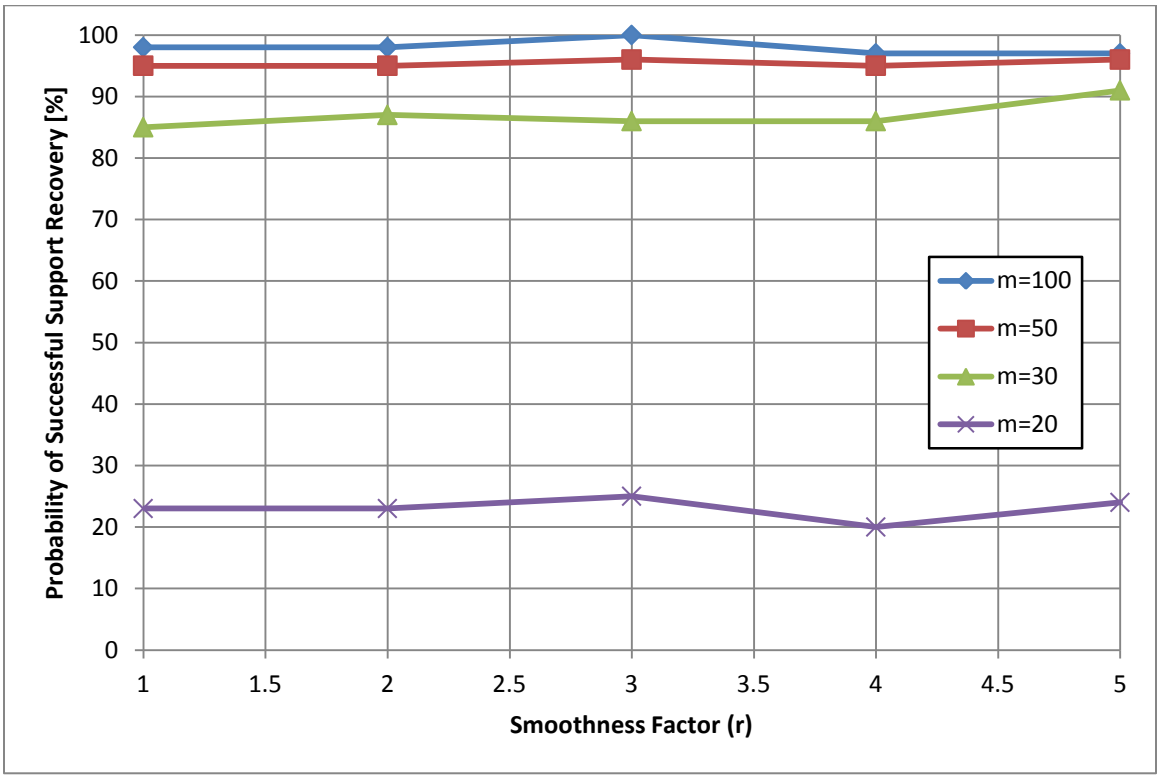

Figure 36 - Filter non-linearity and the probability of successful support recovery, SNR=10dB

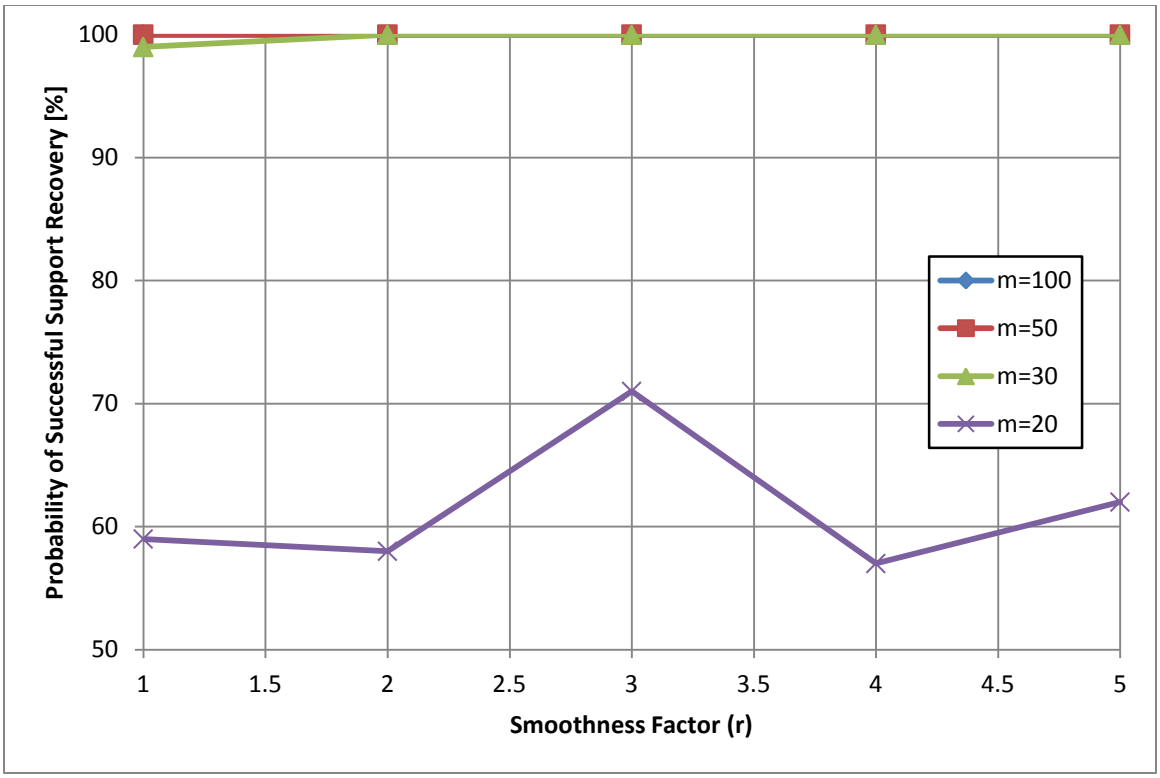

Figure 37 - Filter non-linearity and the probability of successful support recovery, SNR=25dB

As with the non-linearity at the mixer, for a system with at least 30 channels, there is little degradation in performance due to amplitude non-linearity, as expected.

c) Non-Ideal Filters

In theory, the MWC assumes a brick-wall behavior from the low pass filter, therefore it can be assumed that no signal exists past frequencies of $\pm f_s/2$ after filtering. While this simplifies the numeric analysis, realistic filters cannot achieve this kind of performance due to finite roll-off, finite suppression in the stop-band and non-uniform response in the pass-band. Figure 38 illustrates the spreading of the signal spectral content after mixing.

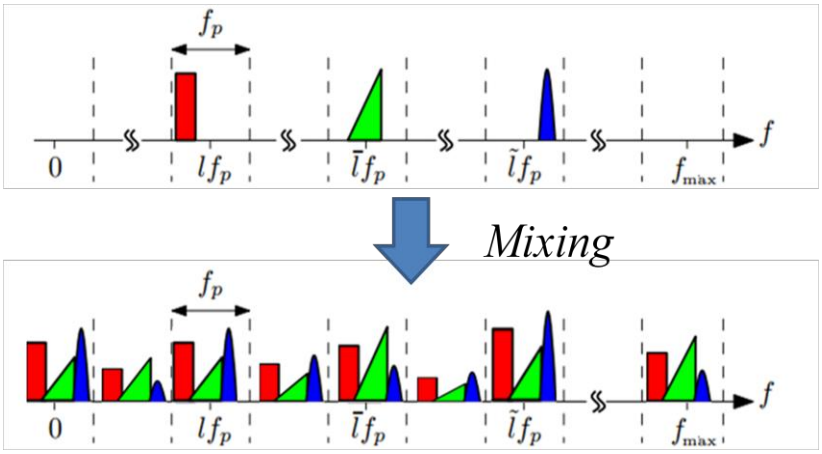

Figure 38 - Effect of mixing periodic mixing function on the signal spectrum [5]

Also it is likely that a signal will fall between two frequency slices, given f_p=B, resulting in some spectrum existing at the edge of spectrum sliced, where finite filter roll off can lead to aliasing. Figure 39 depicts ideal filtering and filtering with a finite roll off of the mixer output depicted in Figure 38.

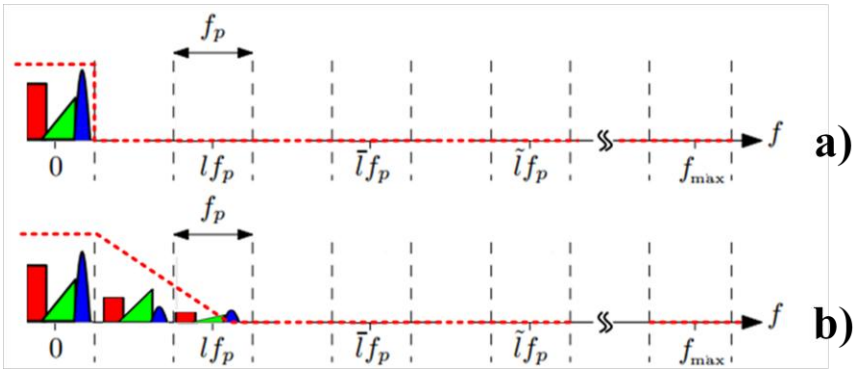

Figure 39 - Spectrum after filtering a) brick-wall filter b) filter with finite roll off [5]

In [5] two cascaded 7th order elliptic filters are used to attempt to emulate the ideal brick-wall filter. Figure 40 shows a reconstructed signal after filtering with a 7th order elliptic filter with -40dB stop-band suppression and 0.1 dB pass-band ripple, where corruption due to non-idealities in the filter can be observed. The effects of filtering with elliptic and Butterworth filters were simulated and results are given below.

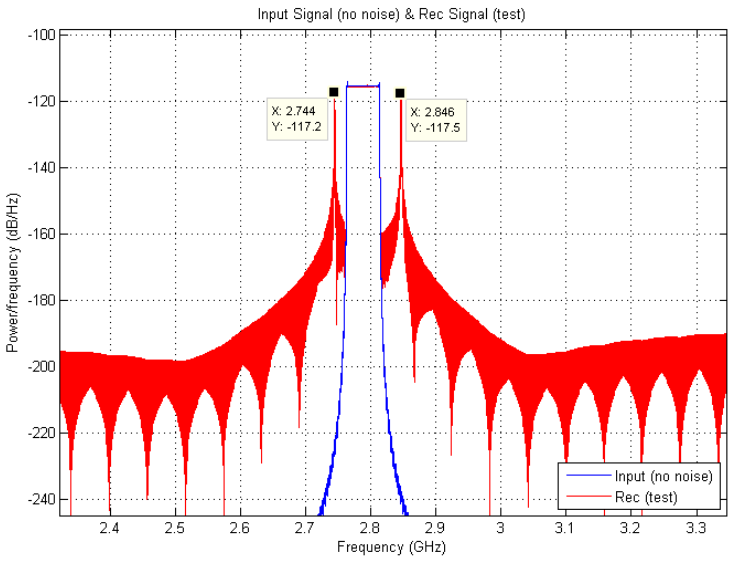

Figure 40 – Reconstructed signal after filtering with 7th order elliptic filter with stop-band supresison of -40dB

(1) Elliptic Filter

Elliptic filters are characterized by the filter cutoff frequency, pass-band ripple, stop-band suppression and filter order. While these filters can be difficult to design, they can offer a sharper stop-band slope than Butterworth filters.

Figure 41 shows the probability of success as a result of filtering an input SNR of 10dB by an elliptic filter of various orders with a stop-band suppression of -40dB and -60dB; Figure 42 shows the same, but for an input SNR of 25dB.

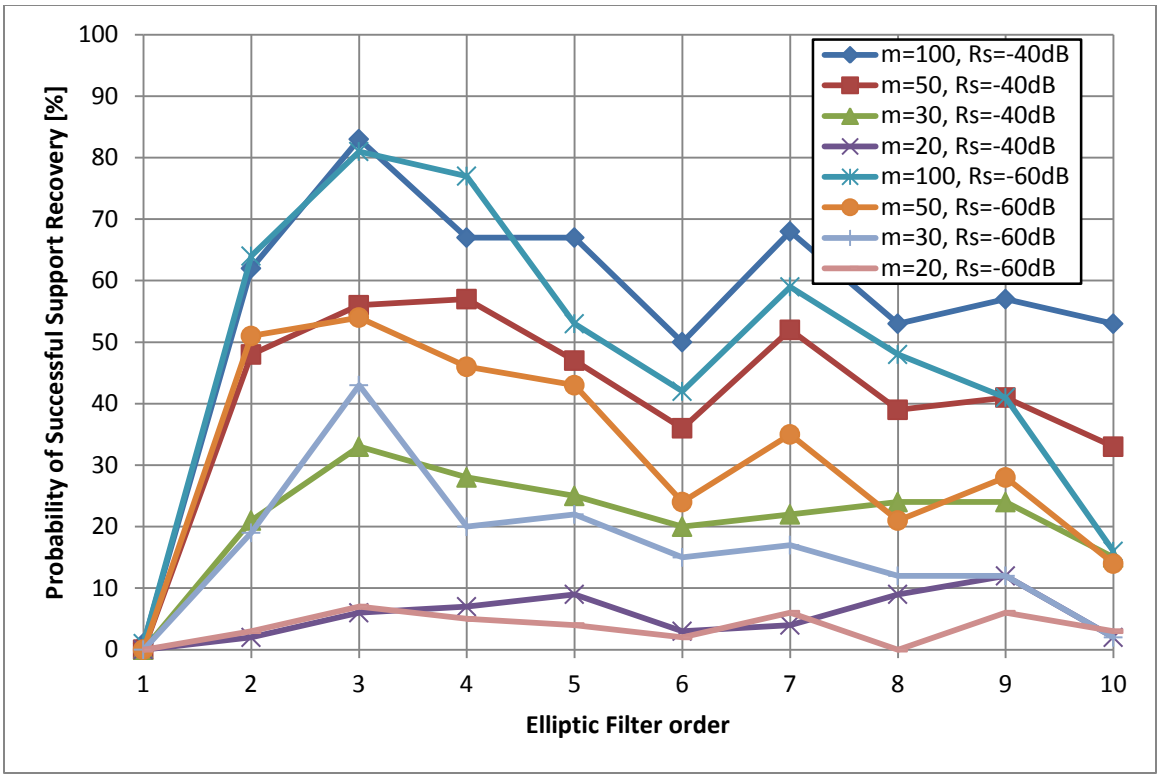

Figure 41 - Probability of successful support recovery and Elliptic Filter order with -40dB and -60dB stop-band suppression, SNR=10dB

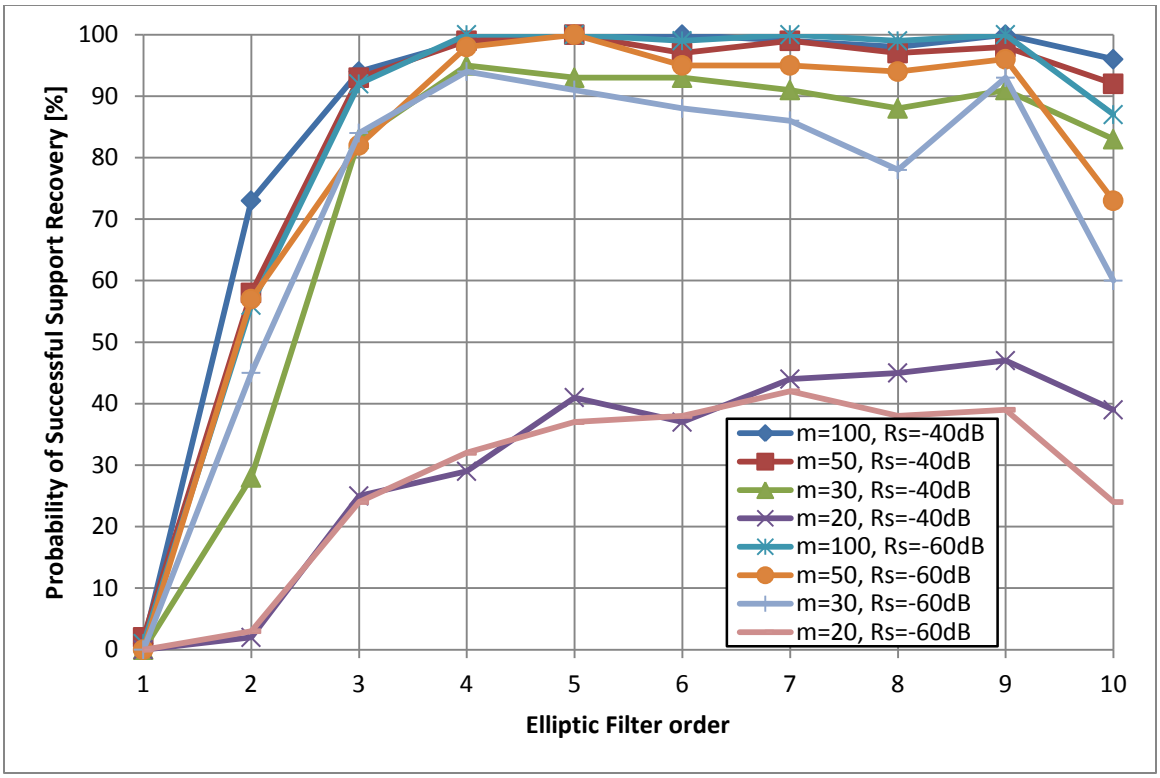

Figure 42 - Probability of successful support recovery and Elliptic Filter order with -40dB and -60dB stop-band suppression, SNR=25dB

These results show that relatively high filter order is required to maintain a high probability of success. However, a higher filter order does not necessarily result in better performance. For example in the SNR=10dB case, the best choice for filter order would be 3, while for an input SNR=25dB the best choice is an order 5 filter. This is likely due to non-ideal filter responses near the pass-band edge and within the pass-band itself [36]. Also, higher suppression doesn't necessarily yield better results. This is clear when the SNR=10dB as the -60dB curve tracks the -40% but with about a 10% degradation in performance. Again, this is may due to the non-ideal shape of the filter needed to guarantee the amount of suppression specified.

It should be noted that the filtering in this section and subsequent sections does not take into account effects of filtering on the signal phase. A given filter will have a fixed, but frequency

dependent phase offset, which would need to be corrected for. This correction will have to be designed per the desired filter. A method is proposed in [36] to correct this phase offset.

(2) Butterworth Filter

Another common filter type is the Butterworth filter, which is characterized by a cut-off frequency and filter order. The amount of stop-band suppression is dictated by the filter order, where higher order filters provide a steeper roll-off in the stop-band. Figure 43 and Figure 44 show the probability of success when filtering with a Butterworth filter of various orders for a 10dB and 25dB input SNR, respectively.

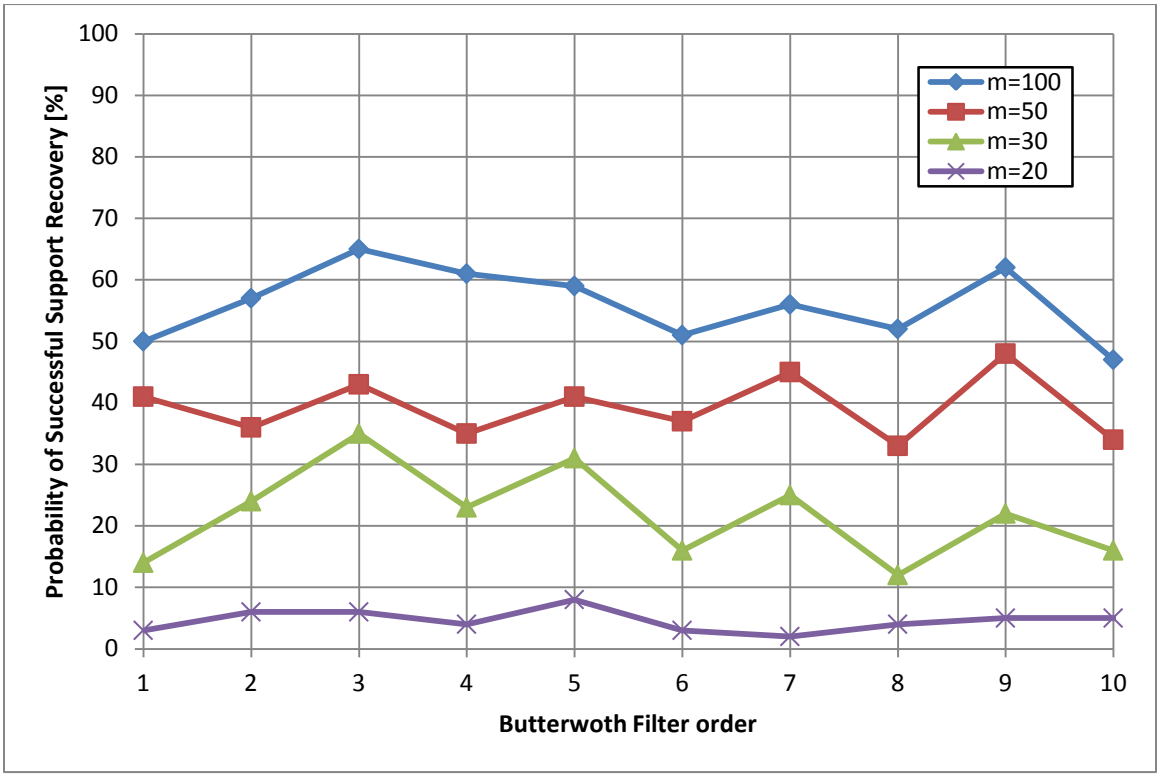

Figure 43 - Probability of successful support recovery and Butterworth Filter order, SNR=10dB

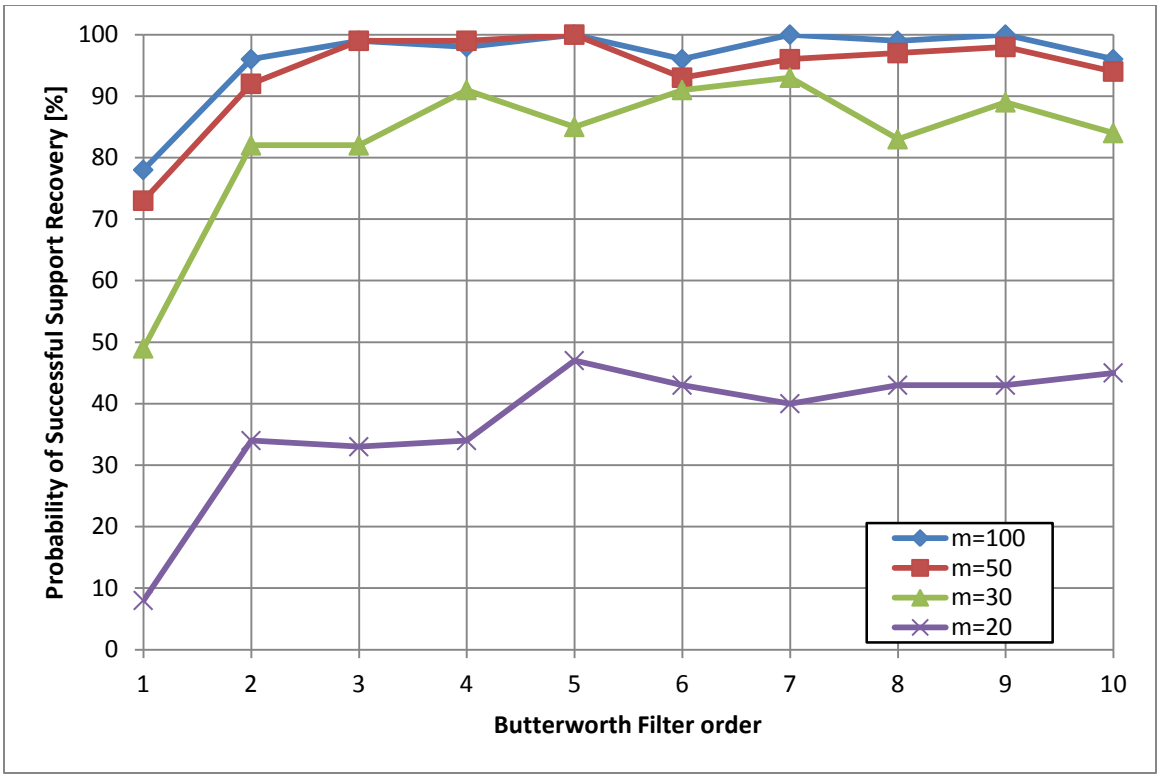

Figure 44 - Probability of successful support recovery and Butterworth Filter order, SNR=25dB

Unlike the elliptic filter, the Butterworth offers a more straightforward tradeoff between filter order and probability of success. These results verify that a higher order filter generally performs better, however, for orders higher than 3, performance may degrade and improve slightly with higher order.

Similar to previous results, the more channels a system has, the more robust the system is to the given impairment. However, filter choice can have dramatic effects on the probability of success. For example, if an elliptic and Butterworth filters was use for an input signal of 10dB SNR, none of the orders simulated can ensure even a 90% success rate.

3. ADC Impairments

The ADC is the last block of the Analog Front End. Impairments at this stage directly affect the digital samples provided to the CS algorithm and thus directly affect the recovery of the support.

While quantization is inherent in any real ADC, noise added to the output and non-linearity are also simulated.

a) Additive Output Noise

Noise added at the output of this stage directly adds noise to the digital samples which are used to compute the spectral support. In the time domain, this introduces an error on the sampled version of the filter output, $\tilde{y}_i(t)$. In the frequency domain, this introduces an error term when comparing the DTFT of the sampled output, $Y_i\left(e^{j2\pi fT_s}\right)$, to the output of the filter $\tilde{Y}_i(f)$, as shown in the equation below.

$$Y_i\left(e^{j2\pi fT_s}\right) = \tilde{Y}_i(f) \rightarrow \sum_{n=-\infty}^{\infty} y_i[n]e^{-j2\pi fnT_s} = \sum_{l=-L_0}^{L_0} c_{il}\, X\left(f - lf_p\right) \tag{65}$$

$$Y_{i,noised}\left(e^{j2\pi fT_s}\right) = \sum_{n=-\infty}^{\infty} (y_i[n] + n[n])e^{-j2\pi fnT_s} \approx \tilde{Y}_i(f) + N(f) \tag{66}$$

This analysis shows that the DTFT of the sampled output is no longer a true representation of the spectrum of the filter output given in (**38**), but an approximation. Intuitively, adding noise will lower the SNR and degrade the probability of success. Like the output of the low pass filter, the output of the ADC is more susceptible to noise than the mixer due to the significantly lowered signal energy after filtering. Figure 45 and Figure 46 show the probability of successful support recovery as a function of added noise to the output for a 10dB and 25dB signal, respectively.

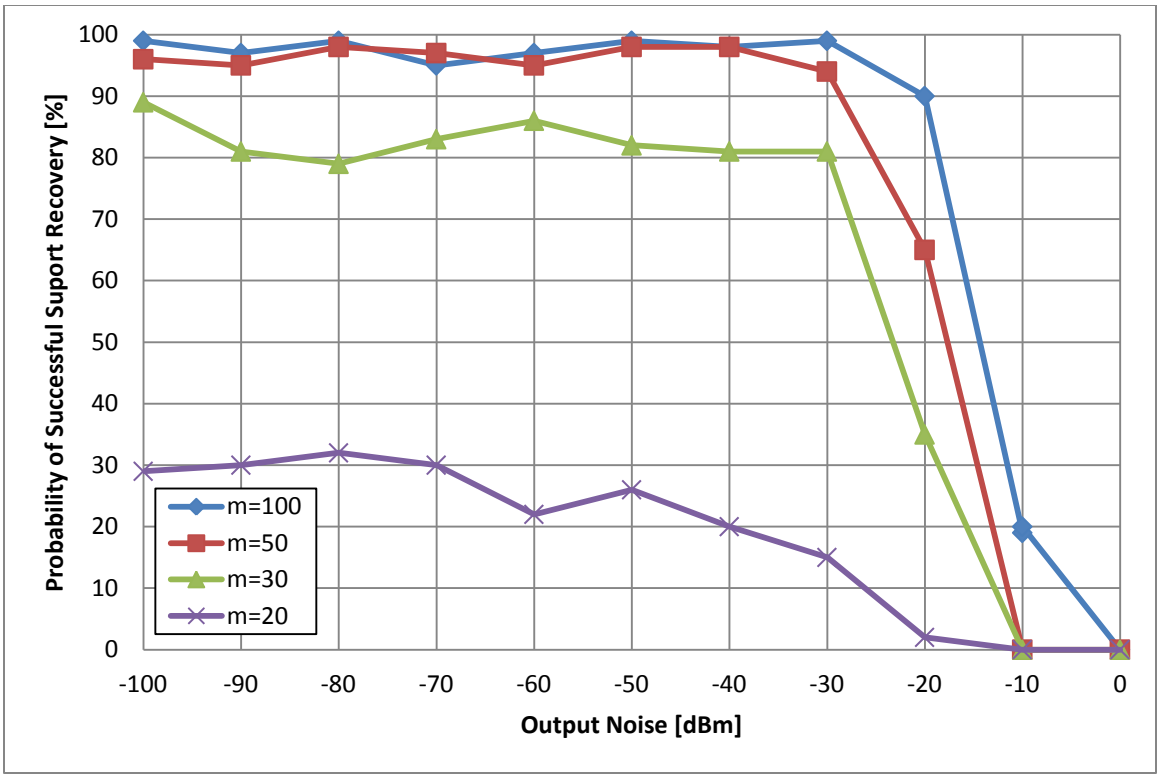

Figure 45 - ADC output noise (dBm) and probability of successful support recovery, SNR=10dB

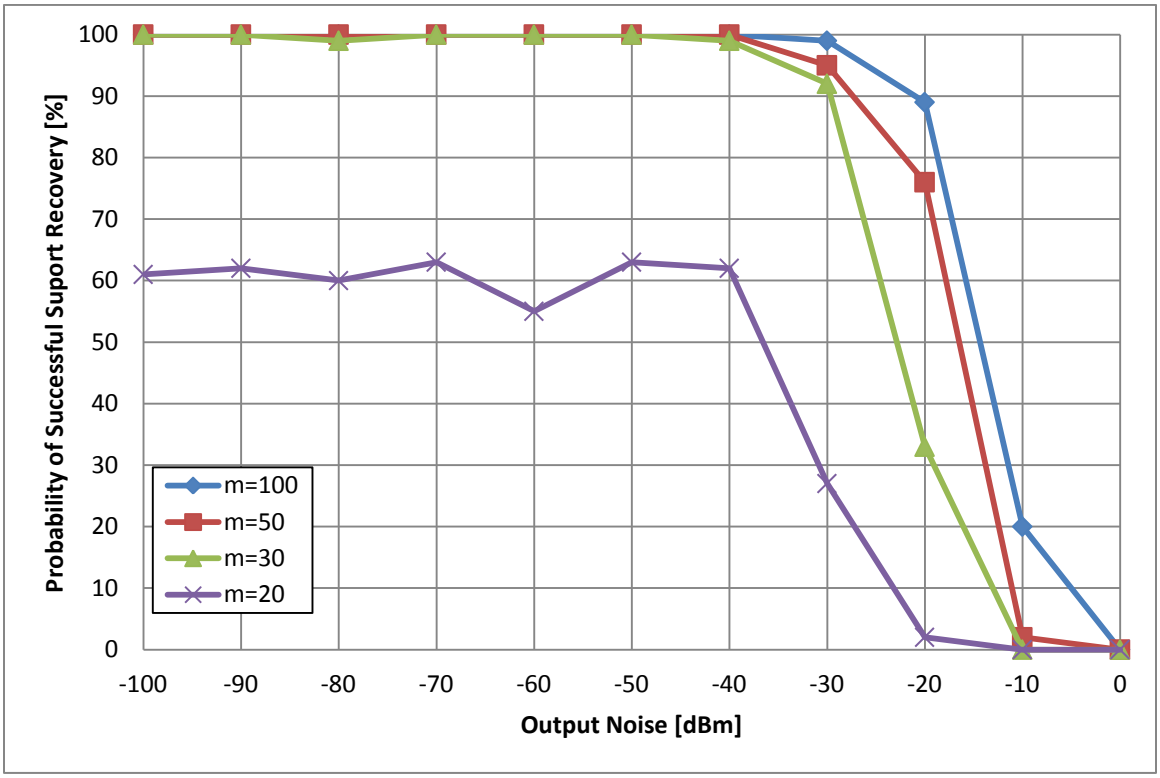

Figure 46 - ADC output noise (dBm) and probability of successful support recovery, SNR=25dB

As expected the noise performance of this block is comparable to that of the filter output and far more sensitive than the mixer output. Take, for example, a 10dB input SNR and 50 channels with an added noise of -20dBm. If this noise was injected at the output of the mixer, the expected probability of success would be 99%. If the noise was added at the output of the filter, this probability drops to 67%. Finally if the noise was added at the output of the ADC, than the probability is 65%.

b) Non-Linearity

Since the signal power and thus the signal amplitude are expected to be drastically reduced due to the mixing and filter operations, this block is not expected to be as sensitive to non-linearity. Following the same arguments used previously, it is inferred that amplitude non-linearity will have little effect on the probability of success. Figure 47 and Figure 48 show the probability of success as a function of the smoothness factor associated with the Rapp non-linearity for an input SNR of 10dB and 25dB respectively.

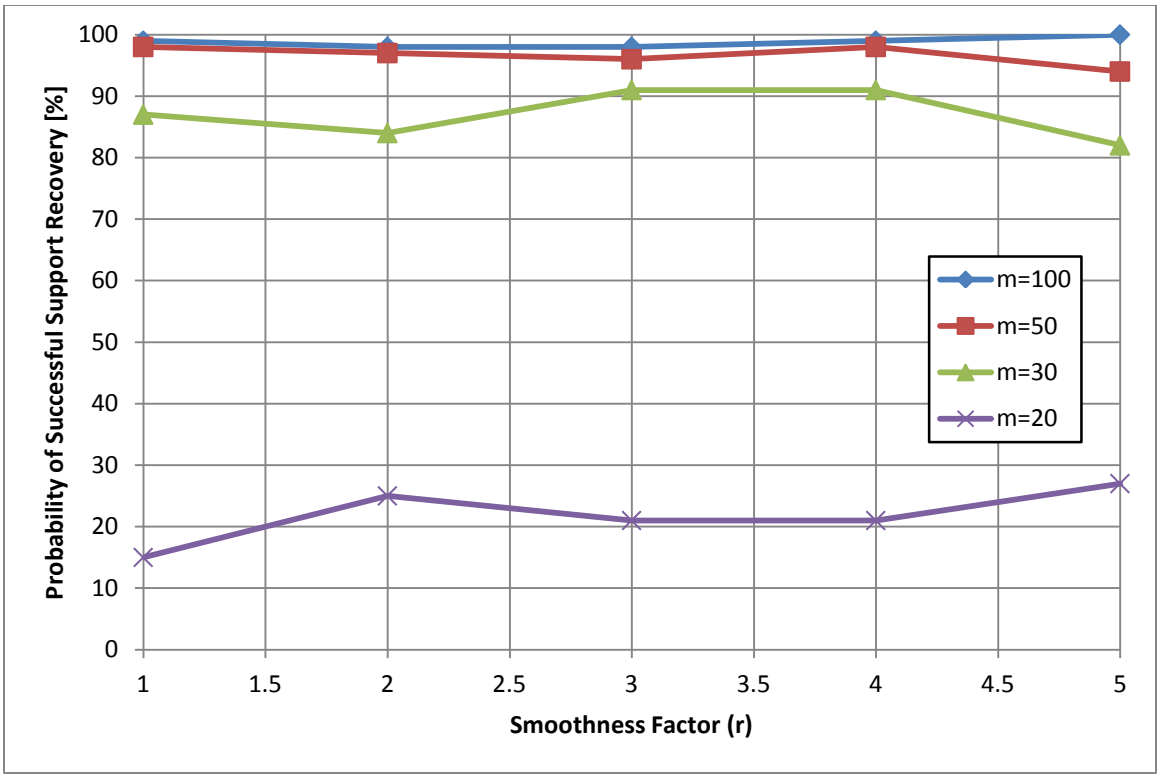

Figure 47 - Probability of success and non-linearity, SNR=10dB

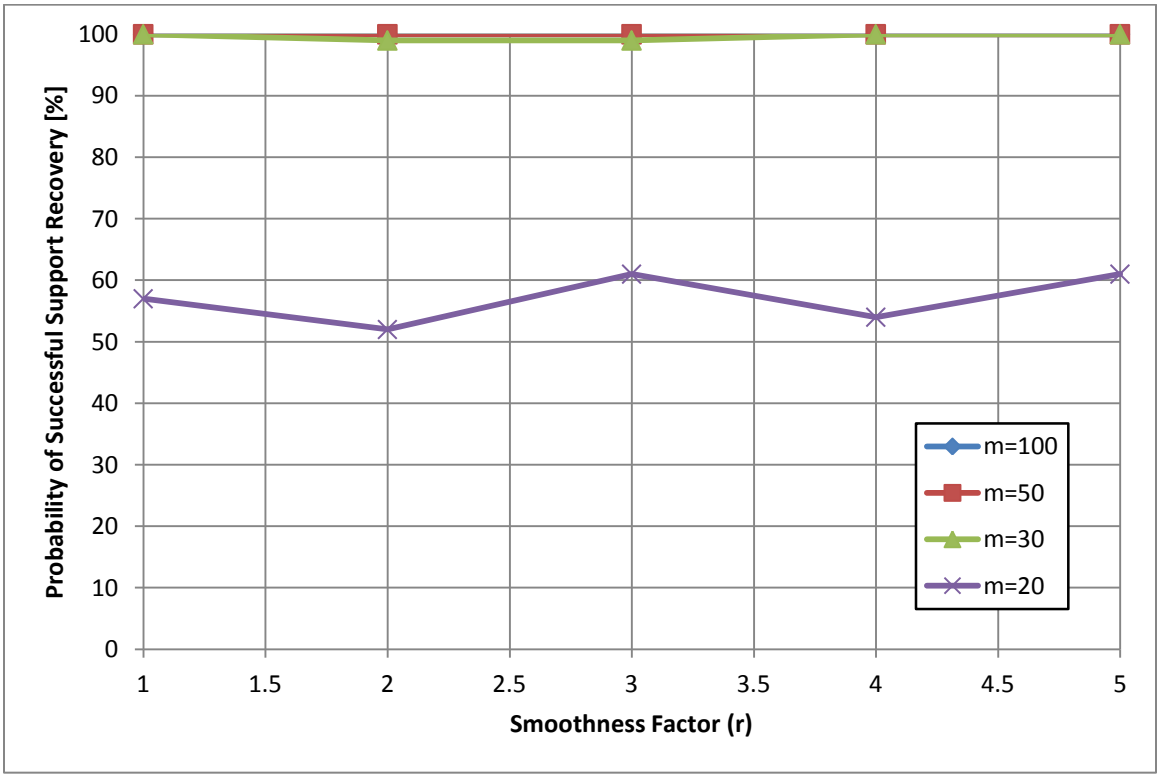

Figure 48 - Probability of success and non-linearity, SNR=25dB

As expected, this block is not sensitivity to non-linearity in amplitude. Performance in this block is about the same as other blocks. For example, for the worst case non-linearity (r=1) and 20 channels, for the mixer, the expected probability of success is 15% for a 10dB input signal and 67% for a 25dB signal. For non-linearity at the filter, these numbers are 23% and 59%, while at the ADC these numbers are 15% and 57%. Although this somewhat goes against the idea that the lowered signal power and thus signal amplitude should render the output less sensitive to the non-linearity, which affects signals close to full scale, the lower signal energy may make it more prone to error due to even small amounts of noise and distortion.

c) Quantization

Quantization is inherent in the analog to digital conversion and adds noise to the output signal. While this cannot be modeled as a simple Gaussian noise, the intuition is the same in that the noise due to quantization corrupts the sampled signal, as seen in the equation below.

$$Y_{i,quantized}\left(e^{j2\pi fT_s}\right) = \sum_{n=-\infty}^{\infty} (y_i[n] + quant(\tilde{y}_i[n]))e^{-j2\pi fnT_s} \approx \tilde{Y}_i(f) + Q(\tilde{y}(t)) \qquad (67)$$

This introduces a noise term which is dependent on the instantaneous value of the input to the ADC, $\tilde{y}(t)$. Lower resolutions result in more noise due to quantization which is expected to more severely degrade the probability of success. Figure 49 and Figure 50 show the relationship between probability of success and ADC resolution for an input of 10dB and 25dB respectively.

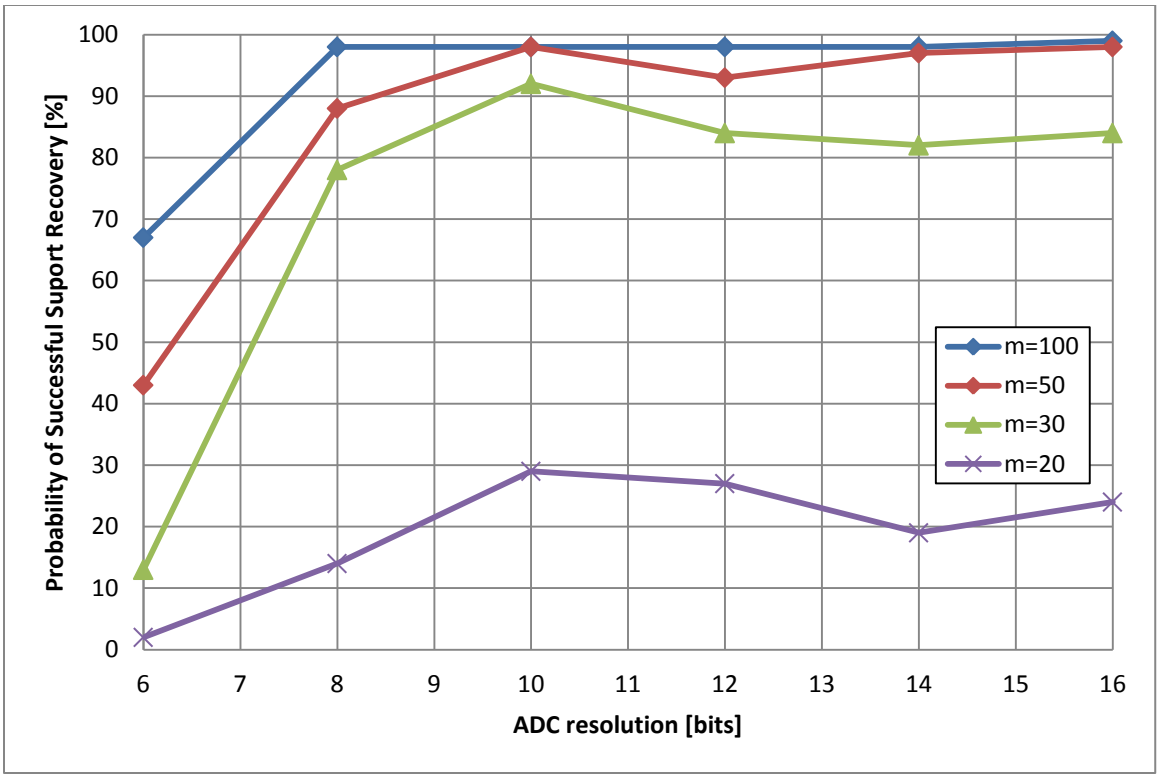

Figure 49 - Probability of success and ADC resolution, SNR=10dB

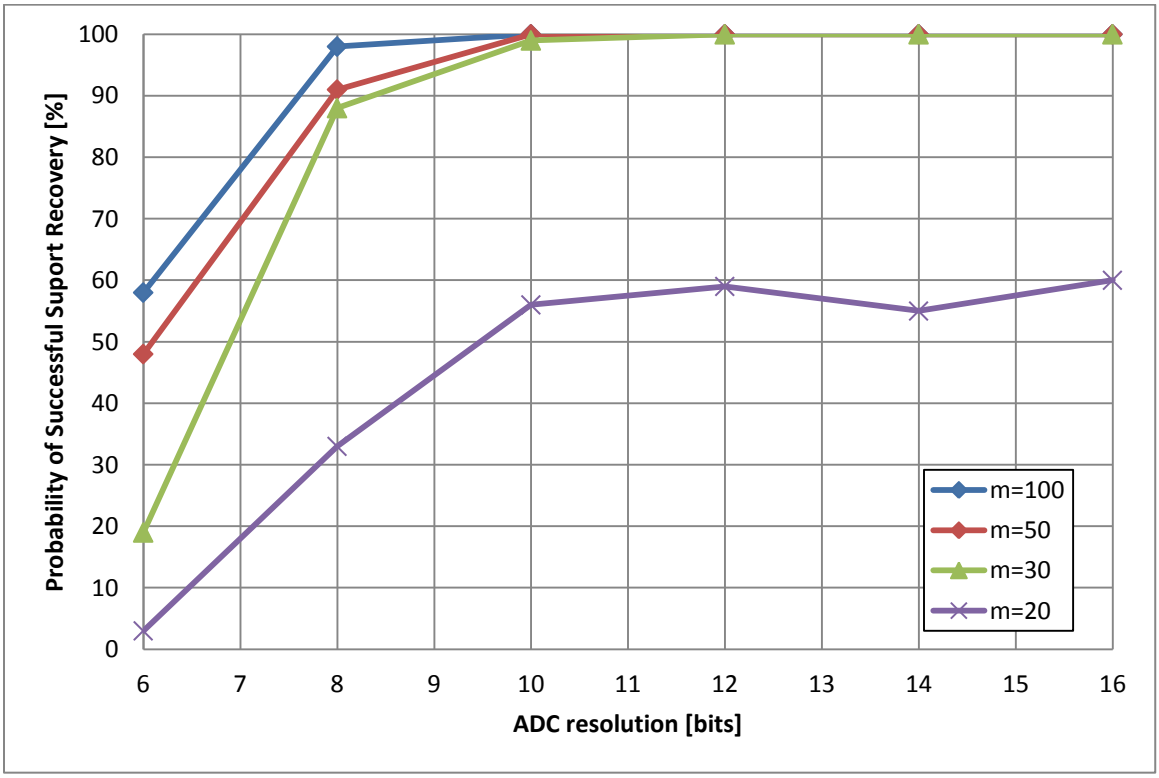

Figure 50 - Probability of success and ADC resolution, SNR=25dB

These results show that relatively high expected probability of success can be achieved with a moderate ADC resolution, although these results are more pessimistic than those found in [2]. These results also show that given an adequate number of channels, 30 or more channels, resolutions of 10 bits or higher can yield expected success rates of greater than 90% for an input SNR of 10dB or 25dB.

C. CS Reconstruction Algorithms: Stability and Runtime

The hardware impairments previously discussed were shown to have a direct impact on the probability of successful recovery of the signal support, however, the choice of CS algorithm also dictates this probability of success. While l_1 minimization techniques are better understood in the greater context of CS research, they do not seem well suited for the CS problem posed and did not perform well. It was found through simulation that popular l_1 solvers such as CVX [30] and L_1 Magic [38] were unable to reliabley reconstruct a vector in which the largest coefficients corresponded to the support of the signal. In addition to inaccurate results, the runtime of the algorithms were relatively long and unpredictable, with convergence not guaranteed for every scenario. Furthermore, L_1 Magic is not suited to deal with complex valued terms, such as the entries of the overall sensing matrix used in the MWC [31]. OMP based algorithms inherently solve the support of the signal while solving the solution vector and lend themselves to this application.

In this experiment, the system and signal models used were the same as those in Section IV.B and are summarized in Table 4. Different OMP based algorithms were used to attempt to recover the signal support for various input SNR levels. The average runtime associated with each algorithm was also calculated and considered as an additional performance metric. No impairments were simulated to ensure best case performance from each algorithm. The purpose of this experiment was to study the performance tradeoffs associated with different recovery algorithms, hardware impairments aside. The results show that OMP with an additional refinement step was the most robust to low SNR signals with a runtime which is comparable to other algorithms. The fastest algorithm was shown to be OMP under the assumption of symmetric signal support. Also, it was shown that CoSaMP, despite the ability to run the

algorithm additional iterations, did not yield better recovery performance for an increased runtime.

Figure 51 shows the probability of success for various input SNR values for four OMP based algorithms and Figure 52 shows the corresponding average runtime for each. 'OMP UnNorm' denotes the Unnormalized OMP algorithm discussed earlier and 'OMP UnNorm (sym)' denotes the same algorithm run in symmetric support mode with half the number of iterations. 'OMP fine' denotes the algorithm which refines the support estimate by inspecting adjacent bands, which was also discussed earlier. 'CoSaMP' denotes the CoSaMP algorithm, which was run 2N times such that the number of iterations would be comparable to other algorithms.

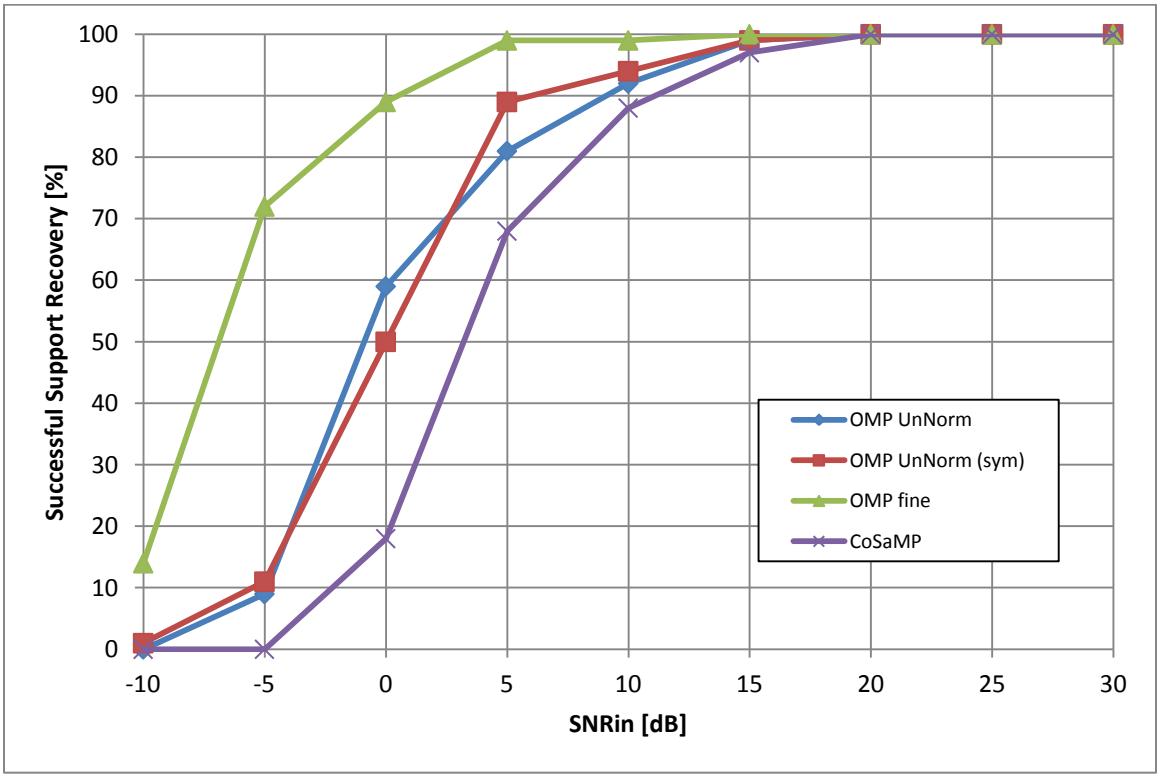

Figure 51 - Probability of successful support recovery for different CS algorithms and SNR values (m=50, N=6)

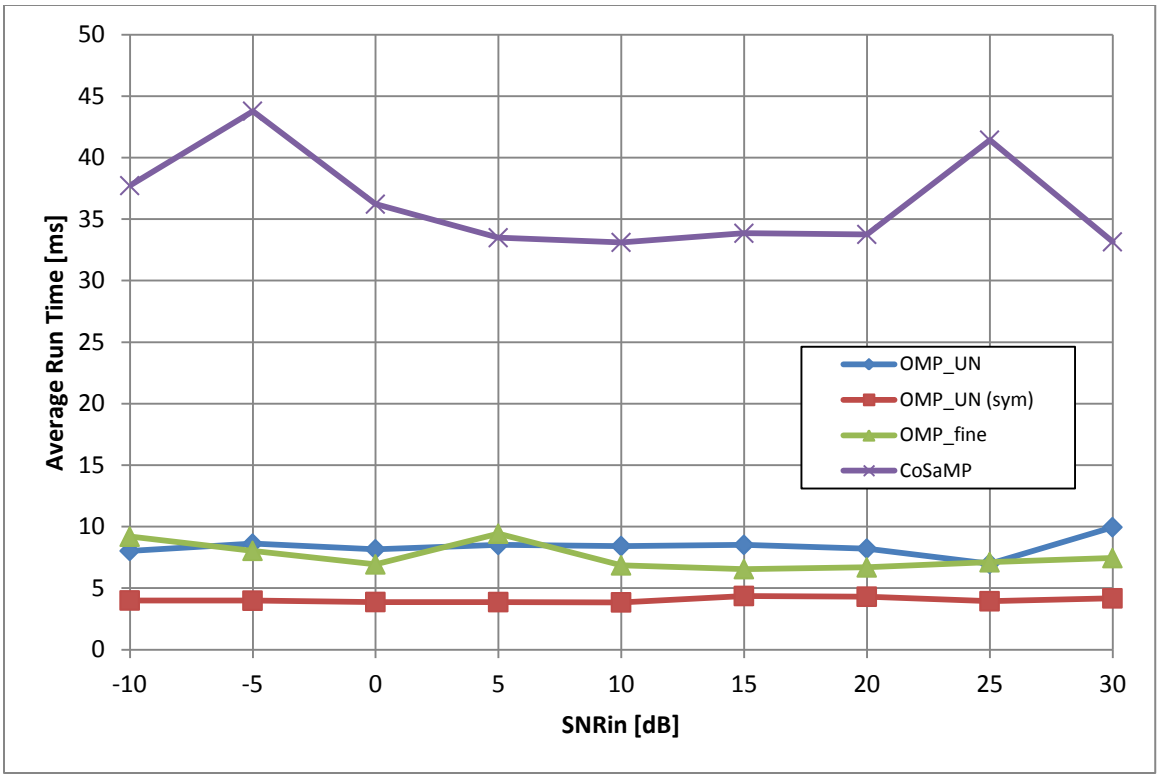

Figure 52 - Average run time for various CS algorithms across various input SNR (m=50, N=6)

These results show that for high input SNR the performance of each algorithm is identical, however, certain algorithms perform better when solving for low input SNR. From Figure 51 it is apparent that 'OMP fine' is the most robust reconstruction algorithm, performing as well as or better than other algorithms for every input SNR. Figure 52 shows that this algorithm has an average runtime of between 5 to 10 ms, which is comparable to both OMP Unnormalized algorithms. Both versions of OMP Unnormalized had comparable performance, but the average runtime of the symmetric variation was approximately half that of the basic configuration, as expected since the number of iterations is reduced to half. CoSaMP had the worst performance in both respects, that is, it was the least robust to low input SNR and had the longest average runtime. However, CoSaMP differs from the other algorithms in that the number of iterations is not fixed to the size of the expected support, therefore there is some freedom in the choice of

number of iterations. These results show that if a system is to be designed for optimal performance, 'OMP fine' would be the best choice. If a system is to be optimized for minimal computation time, one would choose OMP Unnormalized with symmetric support selection.

1. Additional Iterations in CoSaMP Algorithm

The CoSaMP algorithm has a configurable number of iterations which can specified at runtime. As seen in Figure 53, after only two iterations this algorithm nearly reaches it maximum potential for successful support recovery. Figure 54 shows the roughly linear relationship between the number of iterations and the average run time of the algorithm.

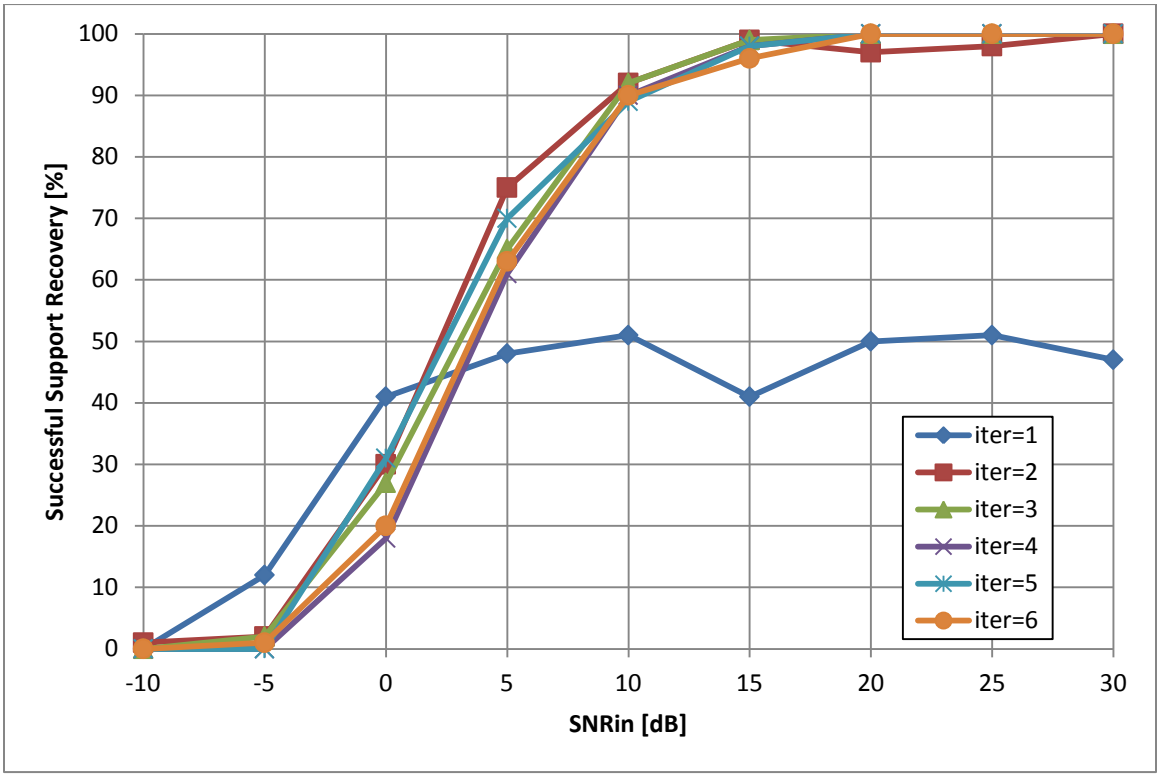

Figure 53 - SNR and probability of successful for various numbers of iterations of CoSaMP algorithm (m=50, N=6)

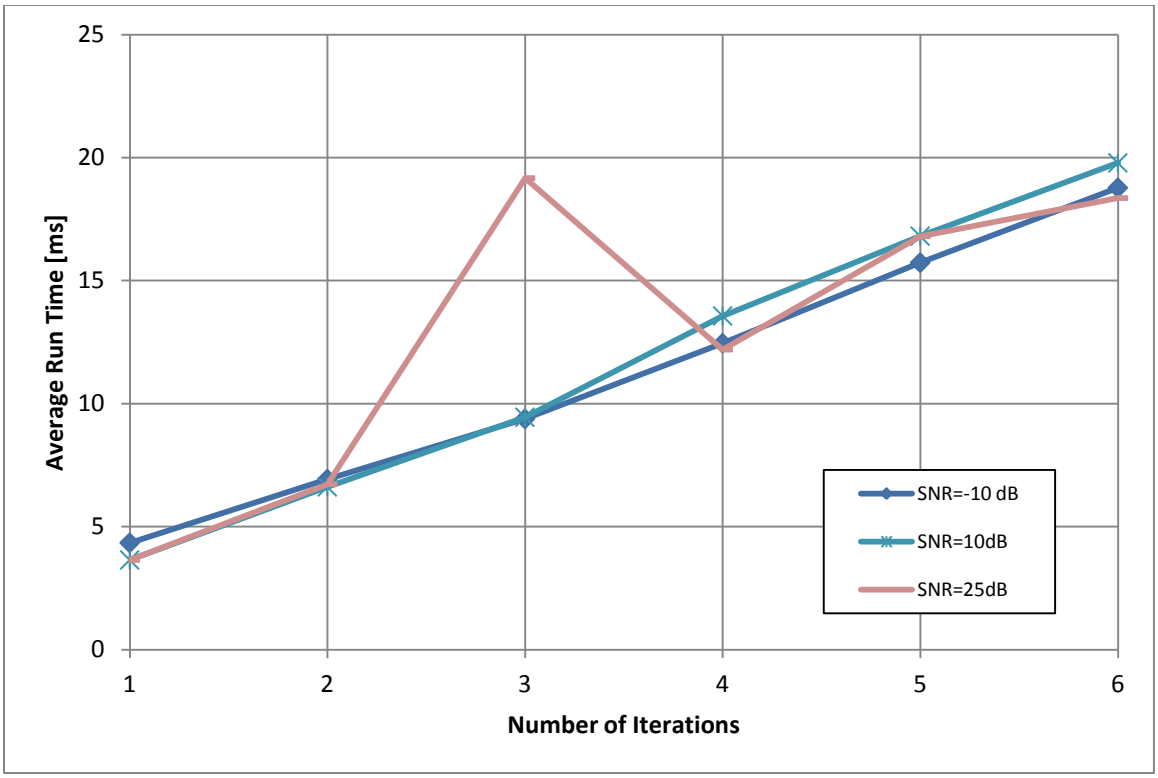

Figure 54 - Run time per iteration for CoSaMP over various SNR values (m=50, N=6)

Combining the results of these two figures, running this algorithm 2N times for N=6 is excessive and leads to a relatively long average runtime with no noticeable increase in recovery performance. At two iterations this algorithm has an average runtime of about 7ms, while for 2N iterations, this average runtime jumps to more than 30ms. However, while this result shows that the runtime in Figure 52 can be reduced to levels comparable to other algorithms with negligible impact on probability of successful support recovery, CoSaMP is still outperformed by other algorithms. Therefore, although CoSaMP has the ability to be run an configurable number of iterations, simulations results indicate that this does not yield any performance benefit over other OMP based algorithms.

D. Signal Energy Dynamic Range

So far, only signal components which have comparable energy have been considered. However, as with all receivers, there is a limit to the ratio between the smallest and largest signals which can be successfully received. To explore the effects of greatly varying signal energies and the probability of successful support recovery, the parameters in Table 4were used.

Table 5 – System parameters for energy dynamic range simulations

N	4
fnyq	10 GHz
fc	random
B	51.2 MHz
fs	51.2 MHz
fp	51.2 MHz
L	195
M	195
Minimum Channel Requirements	
2N	8
4N	16
4Nlog(M/2N)	24

In this experiment, two signal components of different energies were simulated having different frequencies and time delays. The probability of successful support recovery was calculated. The signal is defined by (**54**) with random carrier frequencies and no added noise. Also, no impairments were considered for this experiment in an attempt to isolate the effects of a large energy ratio. The energies, given by E_i in (**54**), were set such that their ratio varied from 1:1 to $1:10^4$. The purpose of this experiment is the study the limit to the largest and smallest signal which the receiver can resolve simultaneously. The results of this experiment show that if the larger of the two signals is set to full scale voltage amplitude, then smaller signal approaches the noise floor as the ratio increases. As the ratio increases, the probability of recovering the support

of both signals decreases. These results provide a best case scenario since no impairments were considered and further degradation is expected when impairments are included.

Figure 55 and Figure 56 show the input signal components with energies ratios equal to 1:1, 1:10, $1:10^2$, $1:10^3$ and $1:10^4$, in the time and frequency (PSD) domains, respectively. Table 6 gives the magnitude of the each signal component PSD for each energy ratio. To isolate the effects of the large energy difference, no additional noise is added to the input signal. Figure 57 shows the probability of successful support recovery for various energy ratios and channel counts.

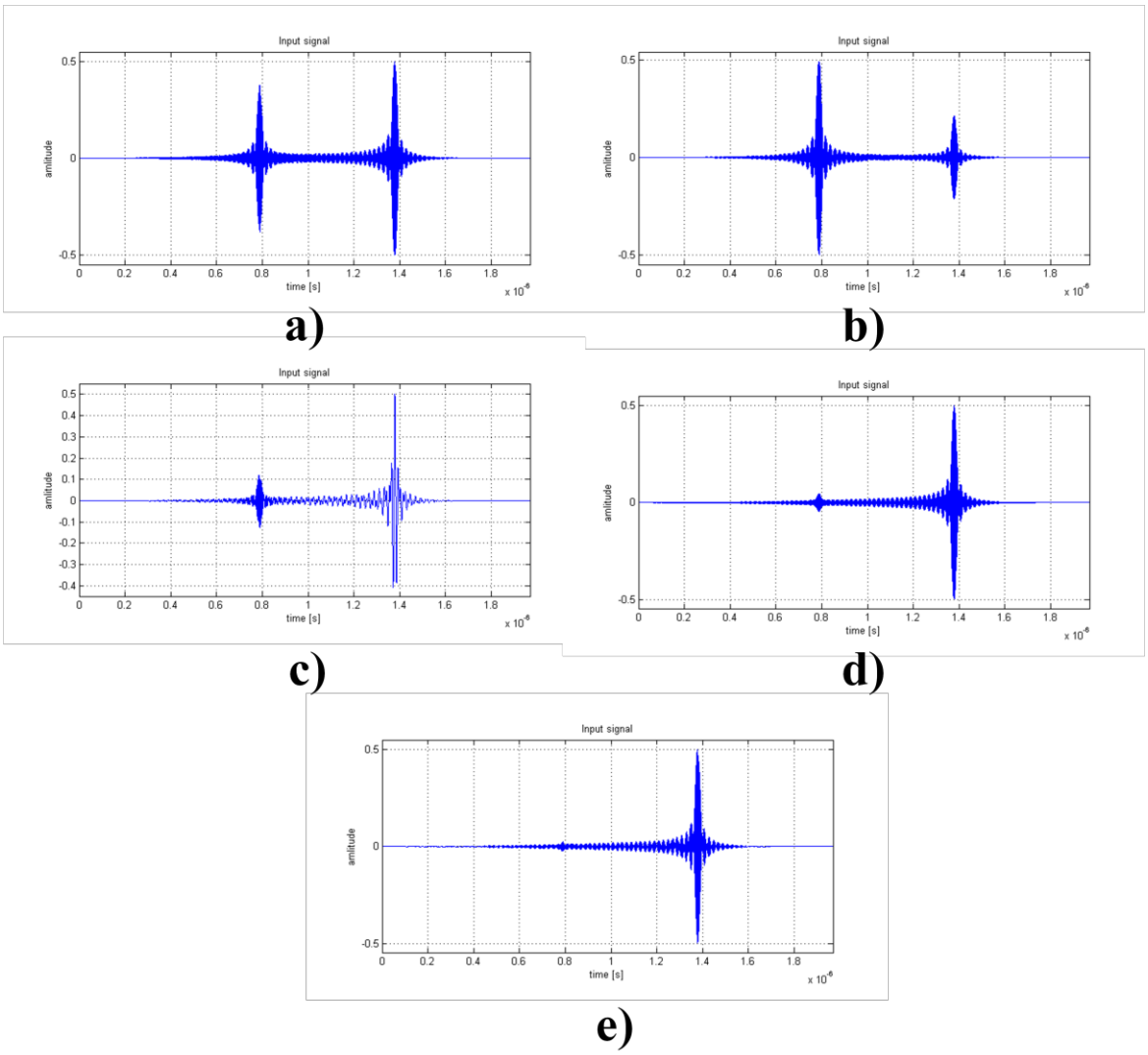

Figure 55 - Input signal (time) of various energy ratios: a) 1:1 b) 1:10 c) 1:100 d) 1:1000 e) 1:10000

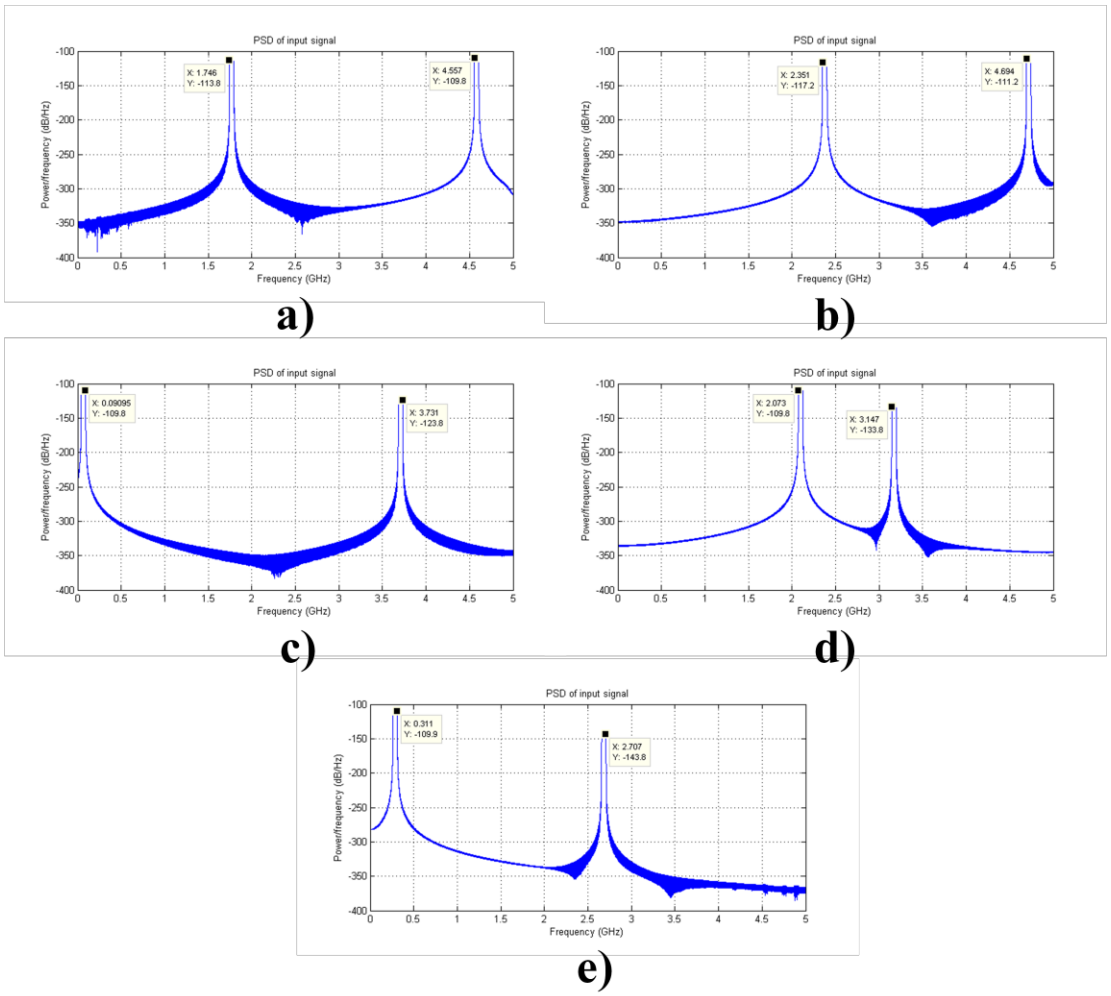

Figure 56 - Input signal (PSD) of various energy ratios: a) 1:1 b) 1:10 c) 1:100 d) 1:1000 e) 1:10000

Table 6 - Magnitude of signal component PSD for various energy ratios

Energy Ratio	|PSD| [dB/Hz]	
	large	small
1:1	-110	-113
1:10	-111	-117
1:100	-110	-124
1:1000	-110	-134
1:10000	-110	-144

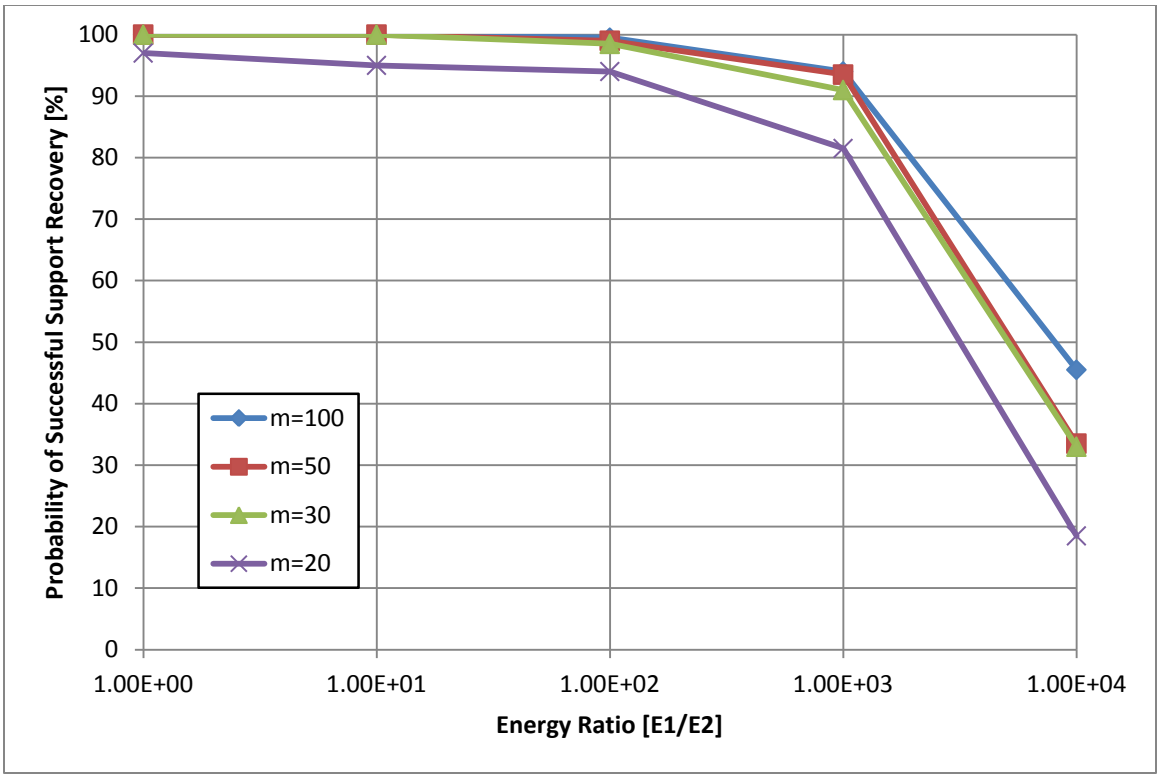

Figure 57 - Probability of successful support recovery for different ratios of energy between two signals

The results show that if the ratio of signal energies is beyond 10^4, this system is not able to reliably recover the signal spectral support. This is expected, since as the ratio gets higher, the amplitude of the smaller signal approaches a value comparable to the noise floor and cannot be reliably detected. In a realistic setting, where noise and unwanted signals may be present, this plot may offer an overly optimistic relationship between the energy ratio and reliable support recovery.

Additionally, these results show that signals which are independent in both frequency and time can still influence overall recovery of the support. This is due to the fact that the MWC system uses the Continuous-to-Finite (CTF) block to reduce the complexity of the CS problem which needs to be solved to find the spectral support. It does so by looking at a certain time window and solving for the support of all signals within that timeframe. This introduces another degree of

complexity as both time and frequency occupancy must be considered when designing the system around the assumed signal model, assuming no adaptation to the signal strength is made in this time window. Simply put, if a very large signal and very small signal occur at different times, but within the same CTF window, the system may not be able to reliable recover the spectral support of both signals.

E. Design of a 2.4GHz Receiver

In this experiment, hardware impairments and channel mismatches are considered for a Xampling based receiver designed for a 2.4GHz, 10MHz bandwidth signal, as presented in [6]. The signal is a stream of random 16-QAM bits and is given by (**68**) with no noise is added to the input signal to isolate the effects of impairments. For impairments of various severity, probability of successful support recovery and SNR as defined in [6] are calculated. The purpose of this experiment is to study the effects of hardware impairments and channel mismatches on the probability of successful support recovery and the degradation to the SNR of the reconstructed signal. The results show that if the number of channels in the system is sufficiently higher than the theoretical minimum, then the signal support can still be recovered with high probability. The results also show how various impairments degrade the SNR of the reconstructed signal, which is a measure of how the system corrupts the input signal.

Table 7 lists the system parameters in this experiment. Unlike previous system models, this one is not designed for lowest possible sampling since the signal bandwidth is only 10MHz while the sampling frequency is more than 120MHz.

Table 7 – System parameters for 2.4GHz simulations

N	2
fnyq	5 GHz
fc	2.4 GHz
B	10 MHz
fs	122 MHz
fp	122 MHz
L	41
M	41
Minimum Channel Requirements	
2N	4
4N	8
4Nlog(M/2N)	9

The input signal to this system is composed of a test vector of random data bits mapped to 16-QAM such that the complex baseband signal can be given by

$$x(t) = x_I(t) + jx_Q(t) = \sqrt{2E_{sym}B}\sum_n \left(I[n]s(t - nT_{sym}) + jQ[n]s(t - nT_{sym})\right) \quad (68)$$

where T_{sym} = 1/B and s(t) is the pulse-shaping filter impulse response. Figure 58 shows this signal in both the time domain and frequency domain (PSD). To isolate the effects of specific impairments, there is no noise added to the input signal.

In addition to the probability of reconstruction, [6] introduces an SNR based performance metric by comparing the original 16-QAM symbols and the reconstructed as given in (**69**) below.

$$\text{SNR} = 20\log\left(\frac{\|\mathbf{I} + j\mathbf{Q}\|_2}{\|(\mathbf{I} - \mathbf{I_{rec}}) + j(\mathbf{Q} - \mathbf{Q_{rec}})\|_2}\right) \quad (69)$$

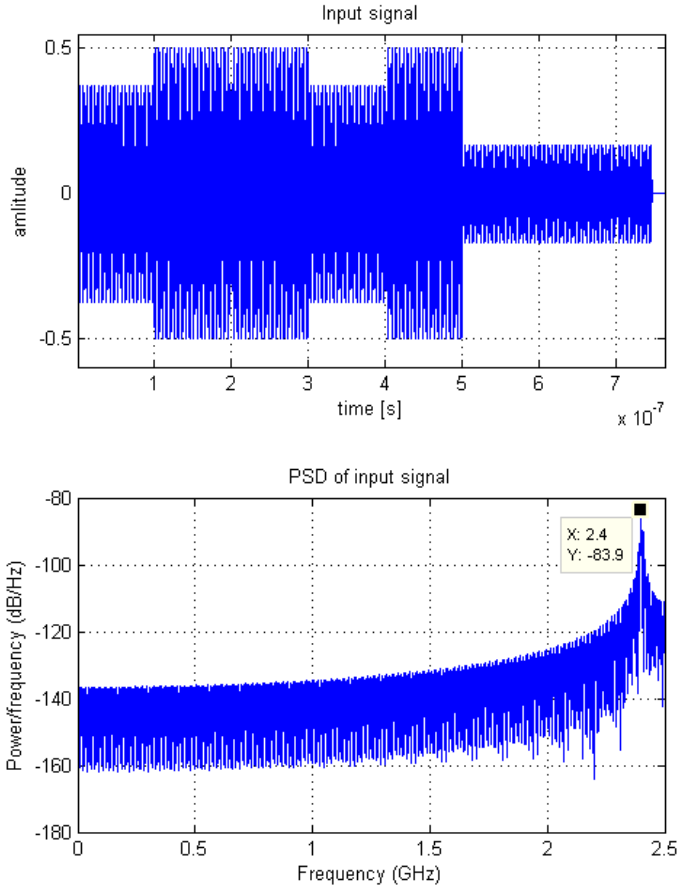

Figure 58 - Input signal in time domain and frequency domain (PSD)

1. Mixer Impairments

Added output noise and amplitude non-linearity were considered when evaluating expected mixer performance.

Figure 59 and Figure 60 show the probability of success and reconstruction SNR as a function of added output noise. Both degrade significantly with increasing noise power, with a steep drop in the probability of success for noise above 30dBm. Reconstruction SNR is observed to degrade in an linear trend until 40dBm, where the SNR levels off.

Figure 61 and Figure 62 show the probability of success and reconstruction SNR as a function of the smoothness factor associated with the Rapp non-linear model. While the probability of success is relatively unaffected by non-linearity, a clear linear trend between reconstruction SNR and the smoothness factor of the non-linearity, r, can be observed. This is in line with the previous expectation that the mixer would be sensitive to non-linearity, since it deals with the full input signal amplitude and bandwidth.

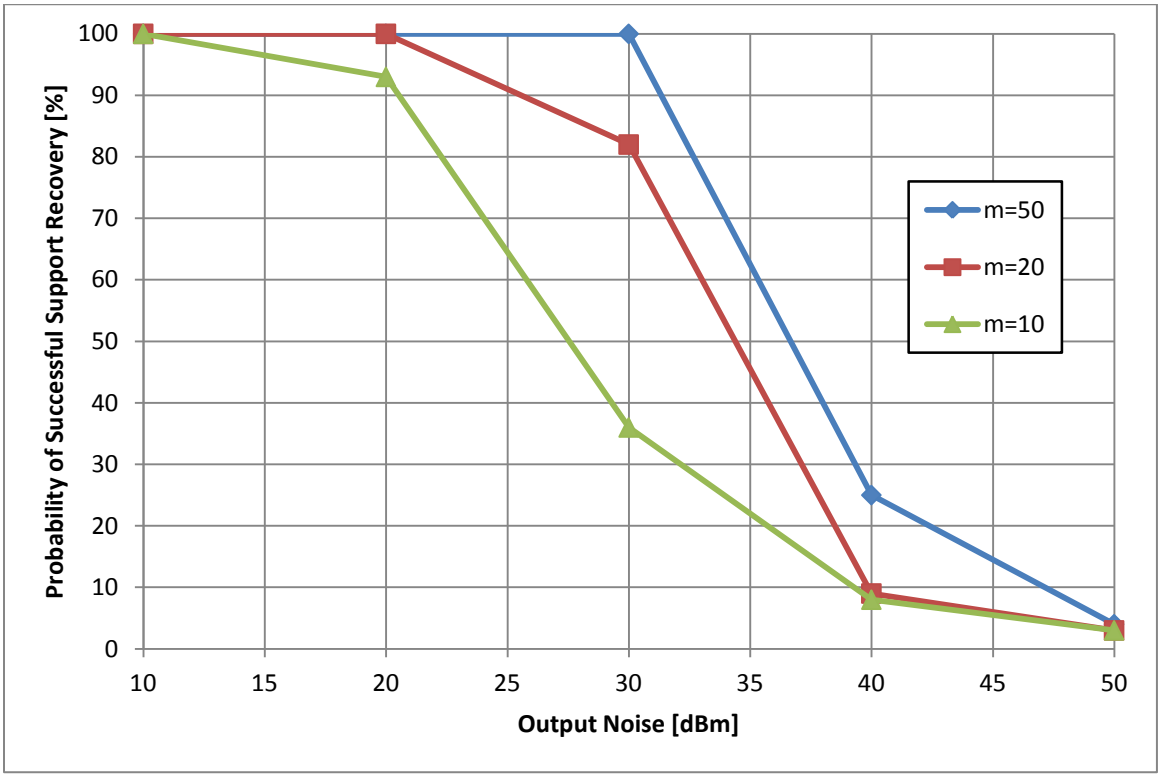

Figure 59 - Probability of success and mixer output noise

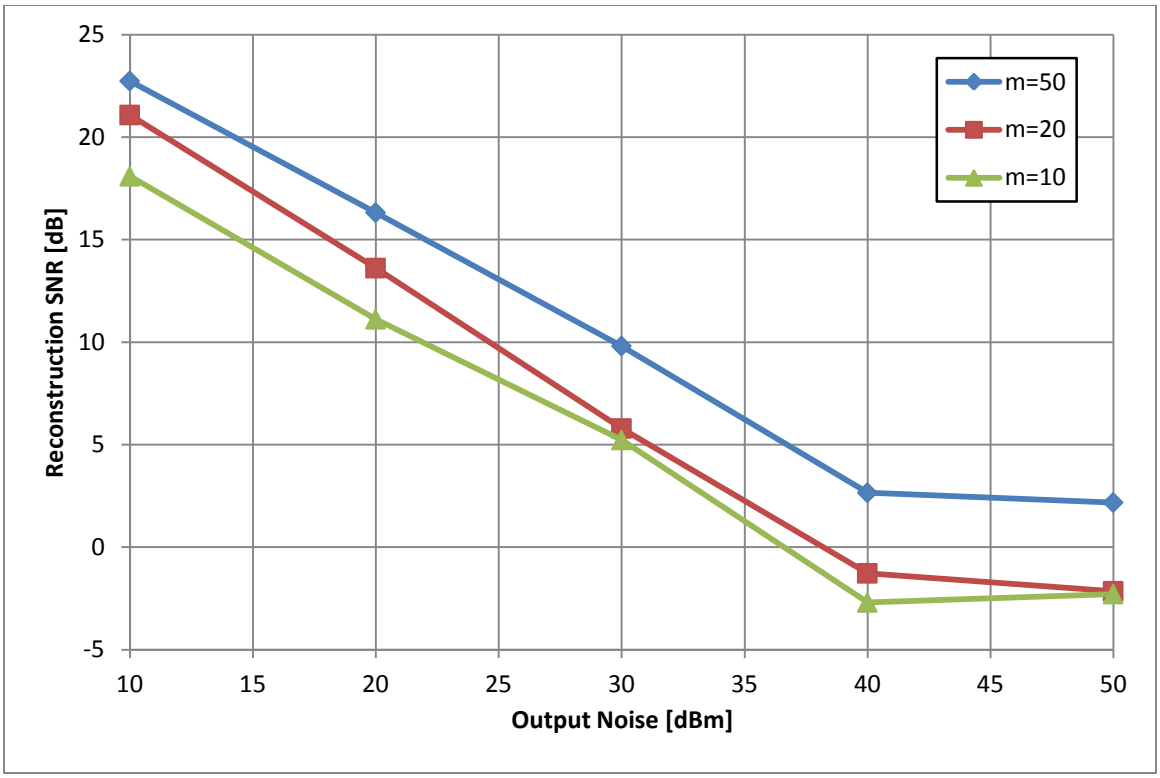

Figure 60 - Reconstruction SNR and mixer output noise

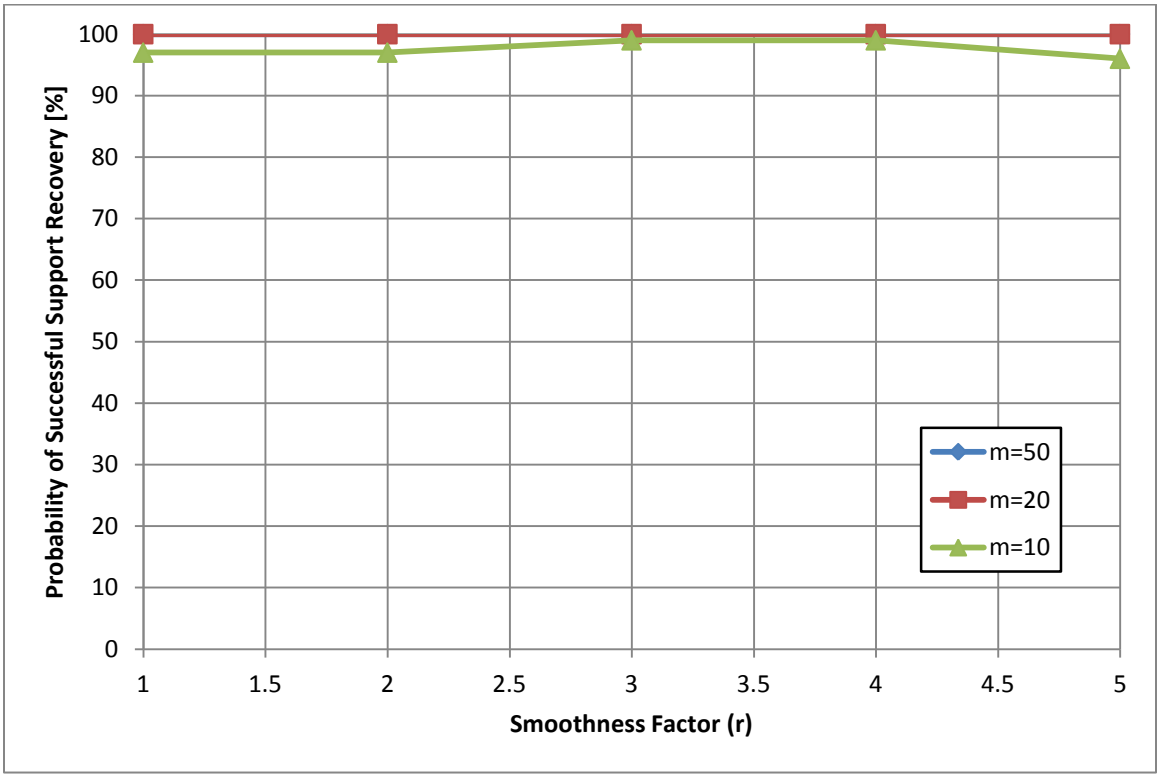

Figure 61 - Probability of success and mixer non-linearity

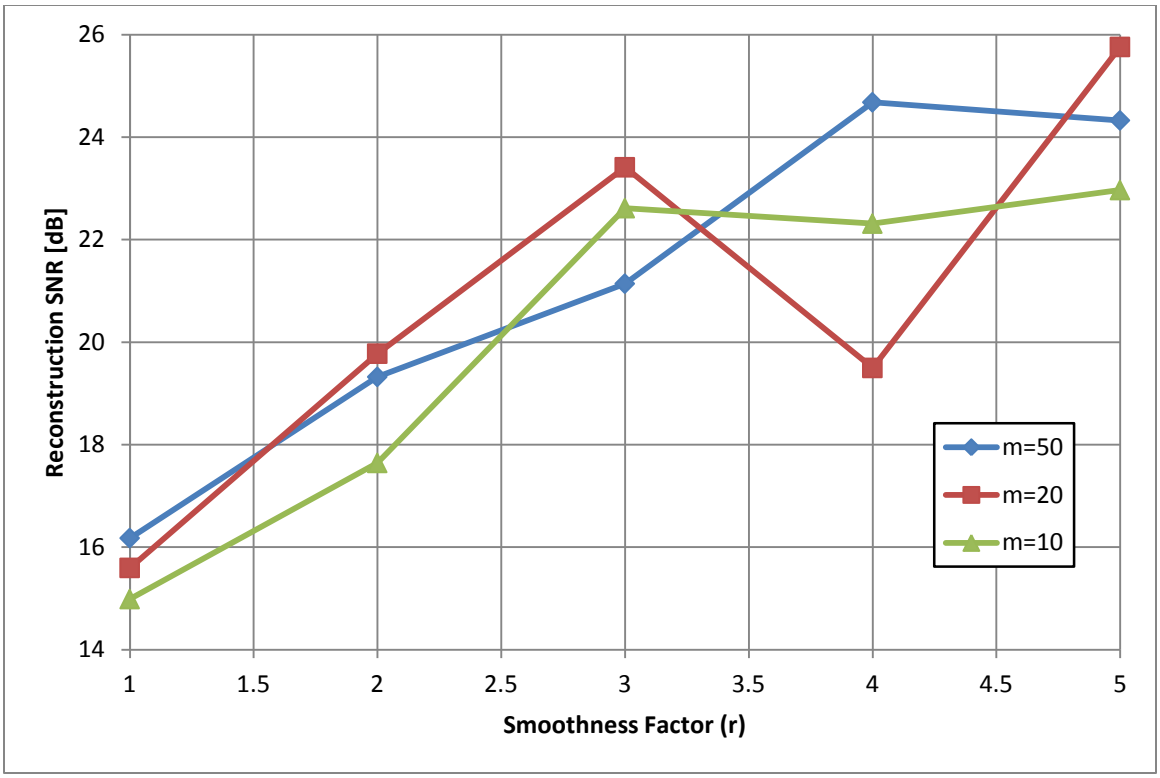

Figure 62 - Reconstruction SNR and mixer non-linearity

2. Filter Impairments

When considering the impairments of the low pass filter, added output noise, non-linearity and non-ideal filters were evaluated.

Figure 63 and Figure 64 show the probability of success and reconstruction SNR as a function of added output noise. These results show that the probability of success and the reconstruction SNR are both degraded by added noise to the output.

Figure 65 and Figure 66 show the probability of success and reconstruction SNR as a function of the smoothness factor associated with the Rapp non-linear model. These results show that the probability of success are relatively unaffected by non-linearity and that there is a slight degradation in reconstruction SNR due to non-linearity.

Figure 67 shows probability of success as a function of Elliptic filter order, while Figure 68 shows the reconstruction SNR, also as a function of Elliptic filter order. These results show that after an order of 3, the probability of reconstruction reaches its highest value. The results for reconstruction SNR are somewhat more complicated, such that for a system with 50 channels, there is a slight degradation in SNR for filter orders above 3. For a system with 20 channels there is an slight improvement past this point and for a system with 10 channels, the SNR value is relatively flat for increased filter order.

Figure 69 and Figure 70 show probability of success and reconstruction SNR as a function of Butterworth filter order, respectively. These results are similar those seen when an elliptic filter was simulated. The probability of successfully recovering the support is relatively constant throughout for a given number of channels, with a slight improvement for systems with 20 or 10 channels when increasing the filter order from 1 to 3. The reconstruction SNR follows a less straightforward behavior with the optimal filter order between 2 and 3 depending on the number of channels.

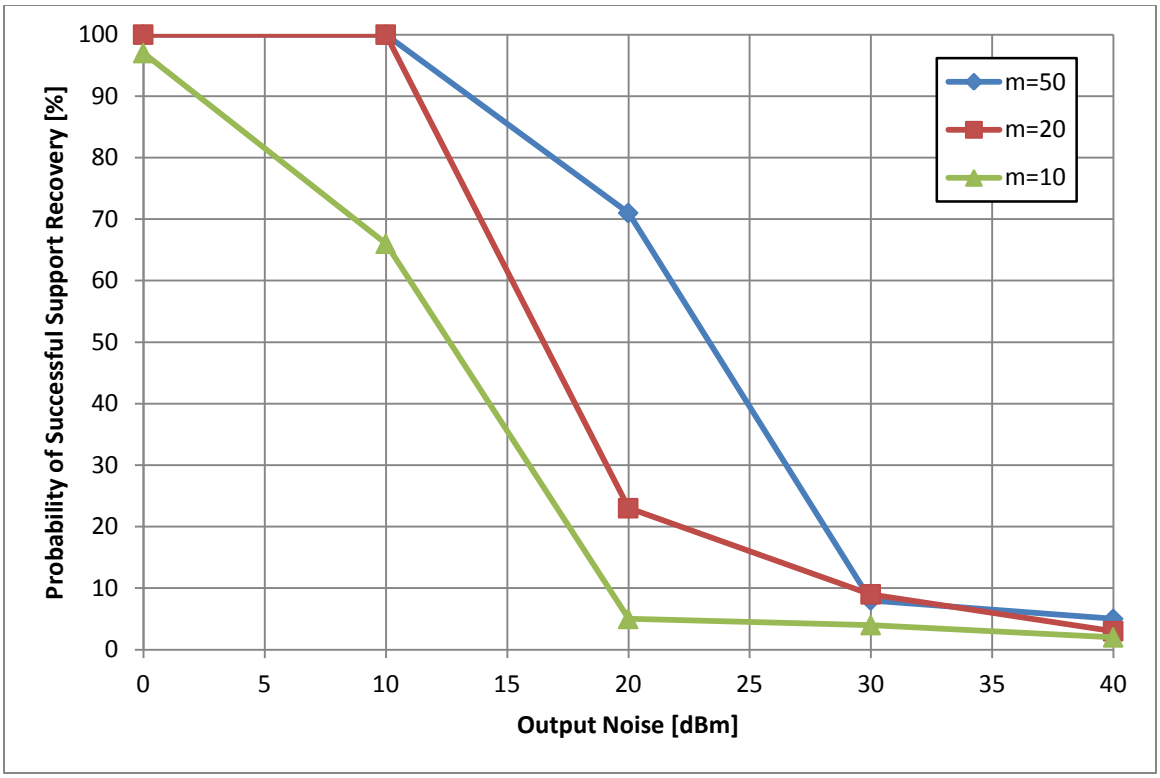

Figure 63 - Probability of success and filter output noise

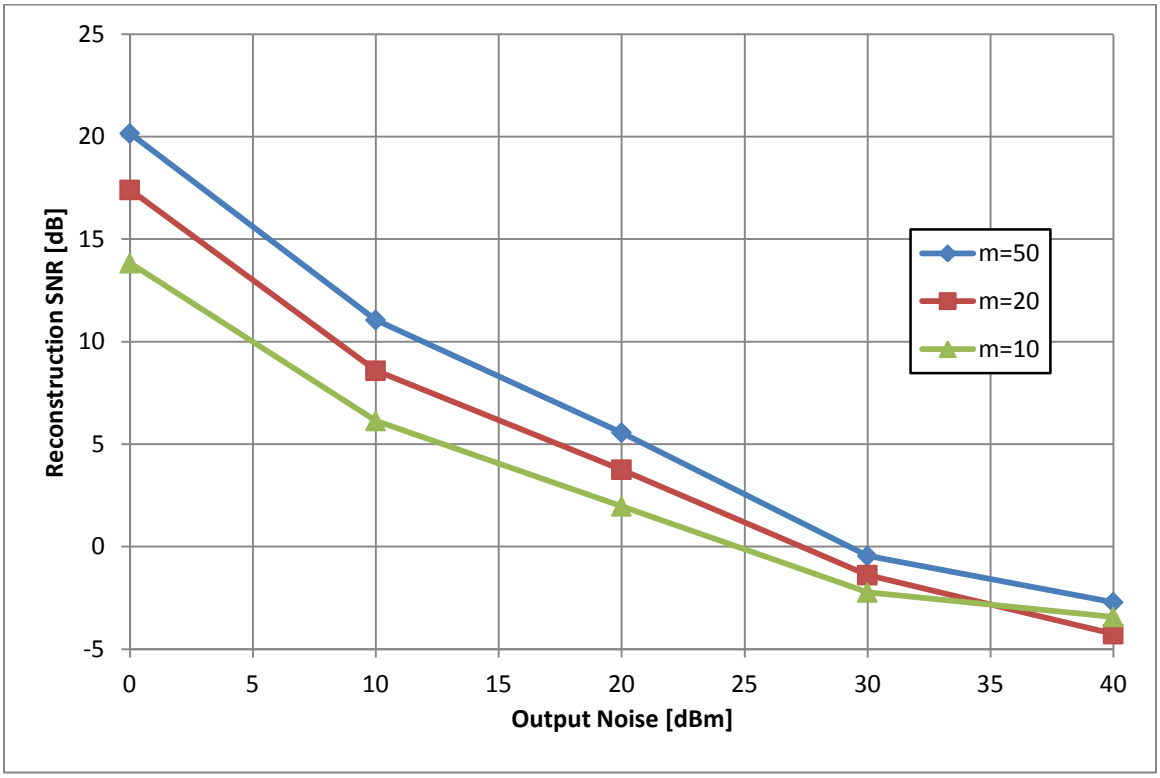

Figure 64 - Reconstruction SNR and filter output noise

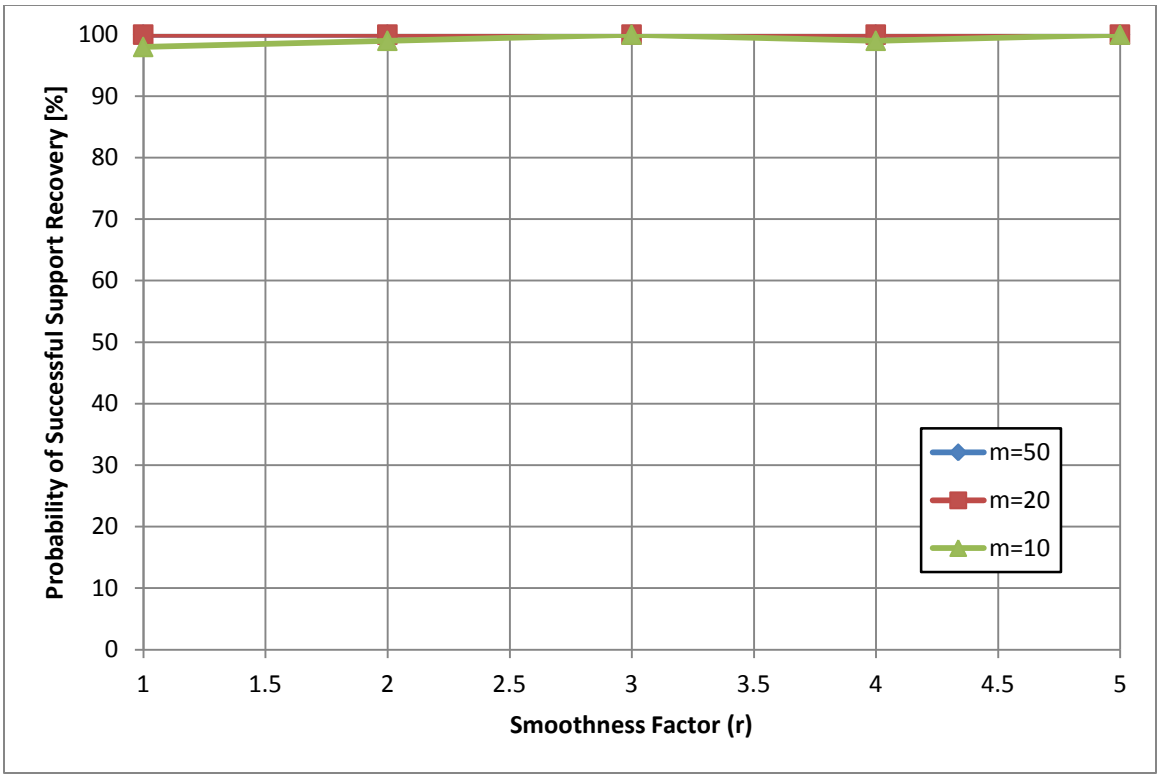

Figure 65 - Probability of success and filter non-linearity

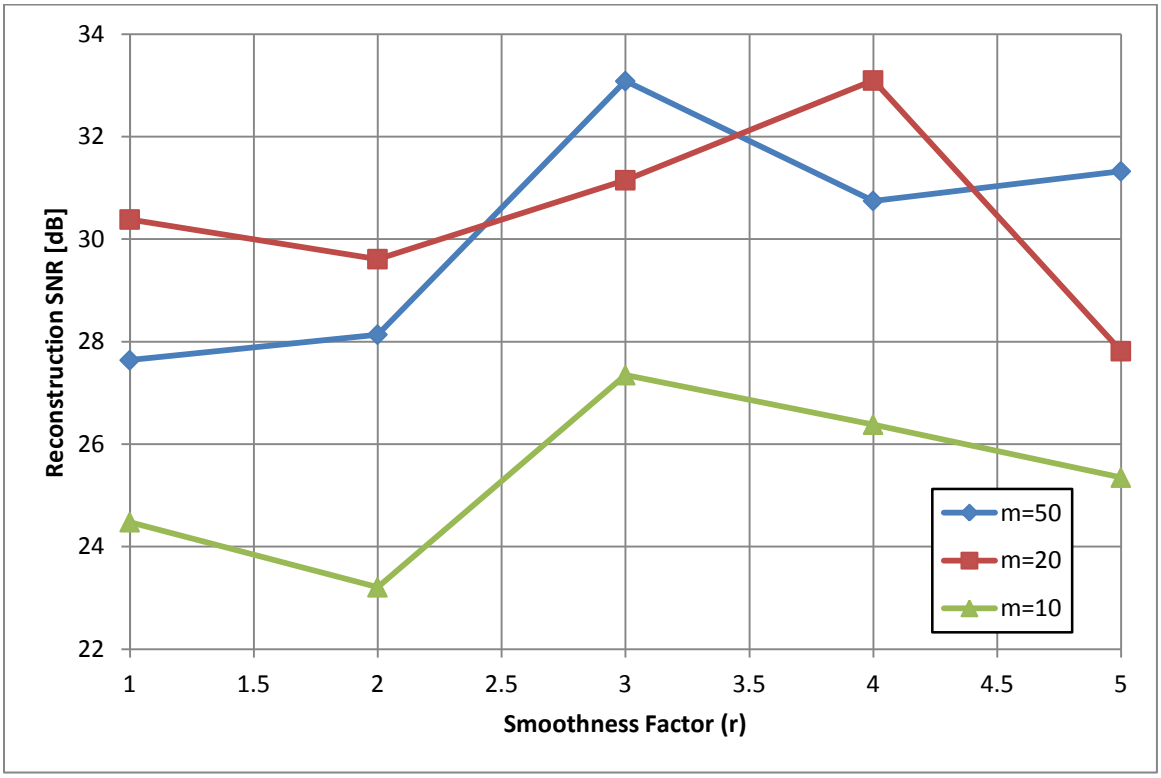

Figure 66 - Reconstruction SNR and filter non-linearity

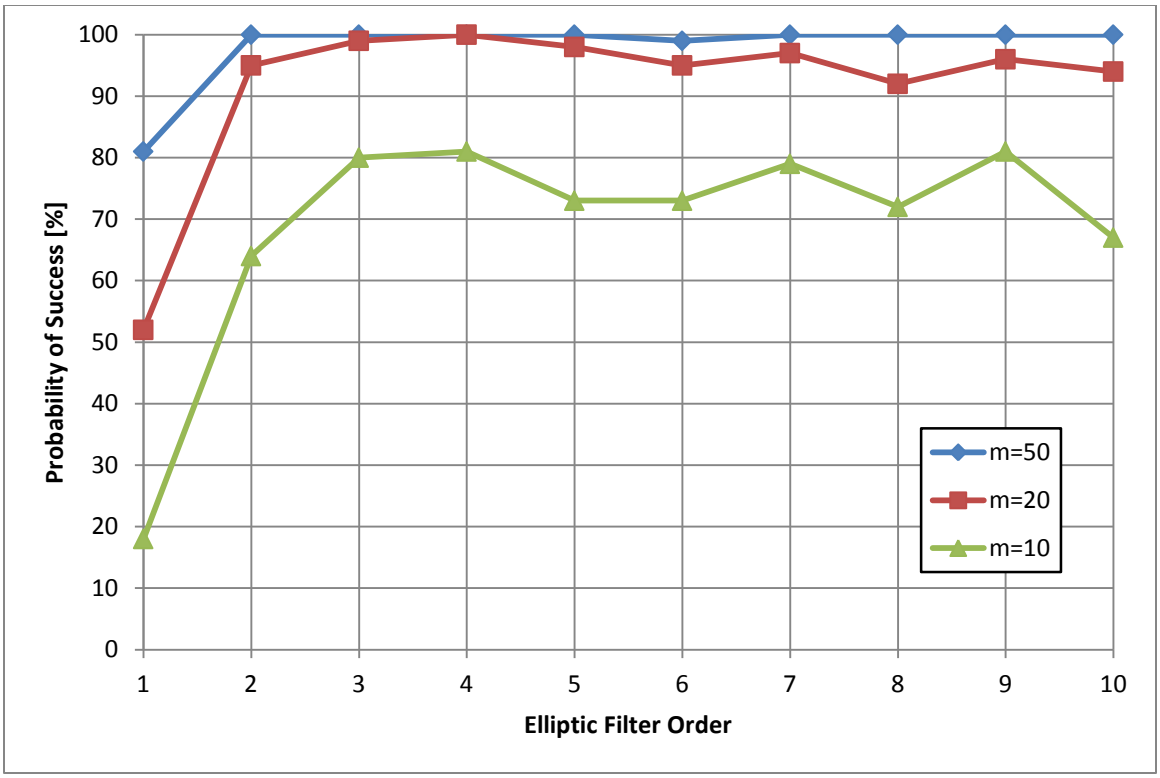

Figure 67 - Probability of success and Elliptic filter order

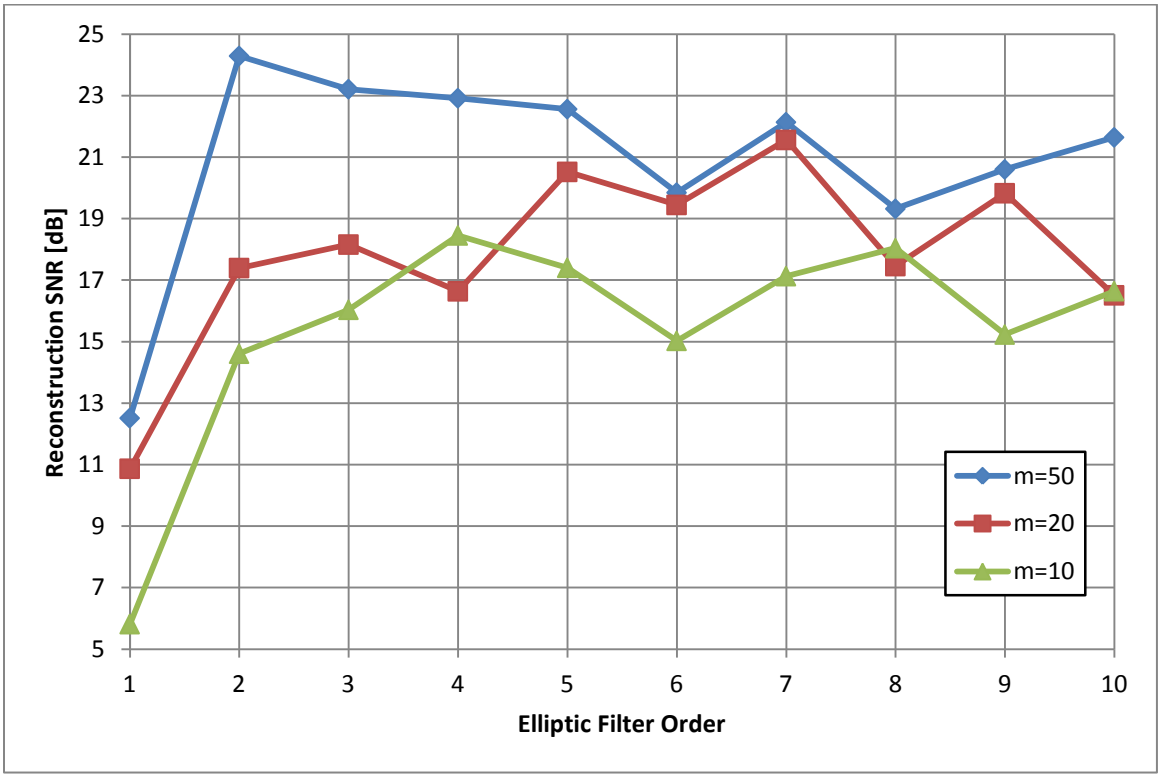

Figure 68 - Reconstruction SNR and Elliptic filter order

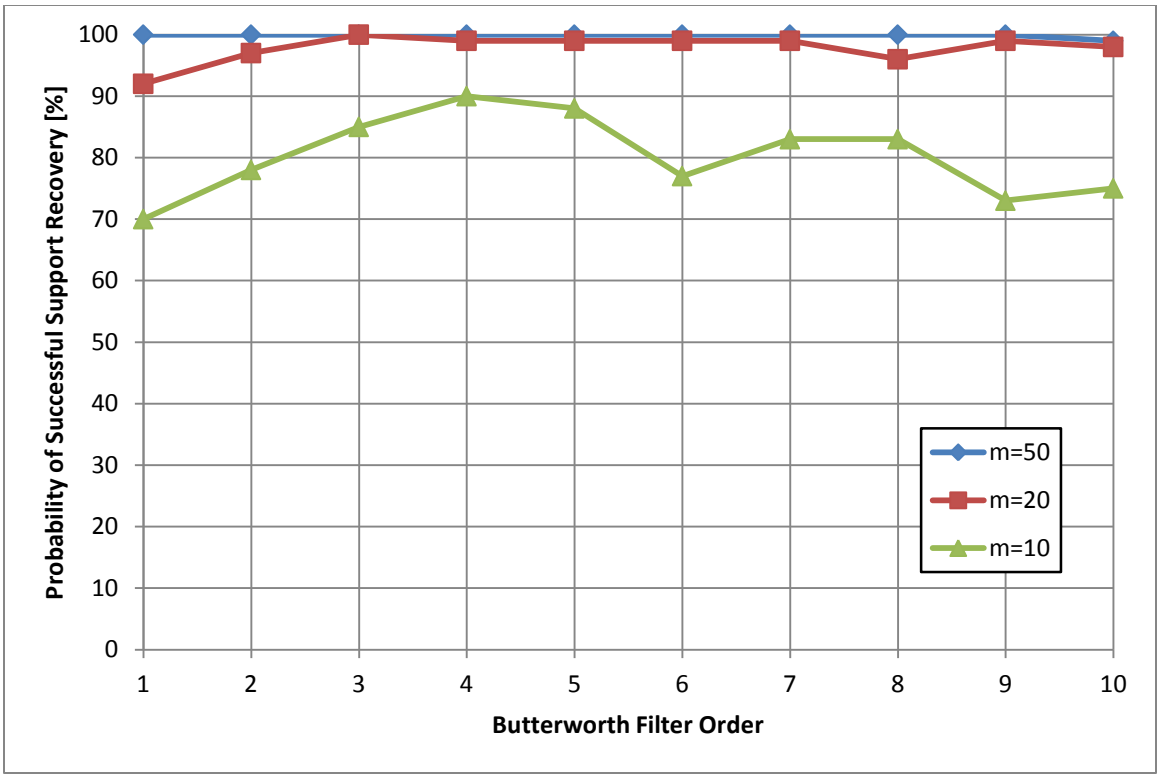

Figure 69 - Probability of success and Butterworth filter order

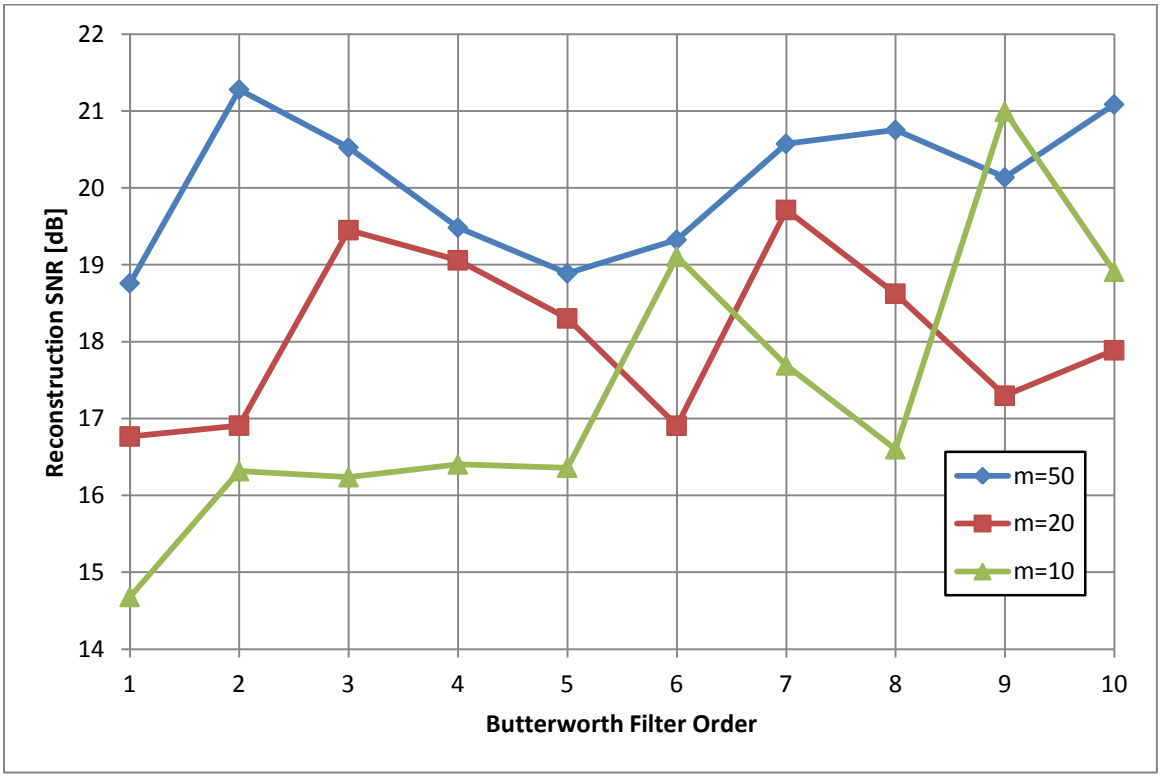

Figure 70 - Reconstruction SNR and Butterworth filter order

3. ADC Impairments

Added output noise, amplitude non-linearity and quantization were considered when evaluating expected ADC performance.

Figure 71 and Figure 72 show the probability of success and reconstruction SNR as a function of added output noise. These results show that both the probability of successful support recovery and reconstruction SNR degrade rapidly with increased noise, with the former experiencing a steep drop and the later following a roughly linear trend.

Figure 73 and Figure 74 show the probability of success and reconstruction SNR as a function of the smoothness factor associated with the Rapp non-linear model. These results show that although the probability of success is not significantly degraded by non-linearity, the reconstruction SNR is somewhat degraded with increasing non-linearity.

Figure 75 and Figure 76 show the probability of success and reconstruction SNR as a function of ADC resolution. These results show that the probability success is not noticeably affected by ADC quantization, however, a clear tradeoff between reconstruction SNR and resolution can be observed.

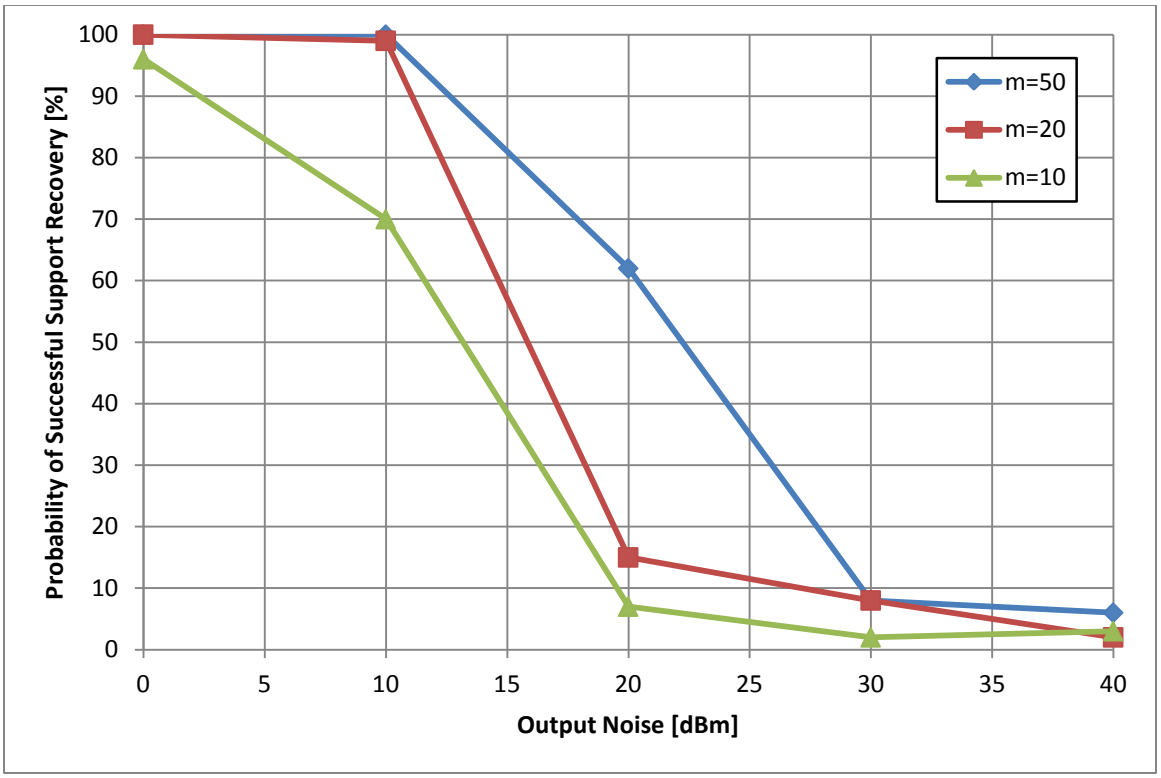

Figure 71 - Probability of Success and ADC output noise

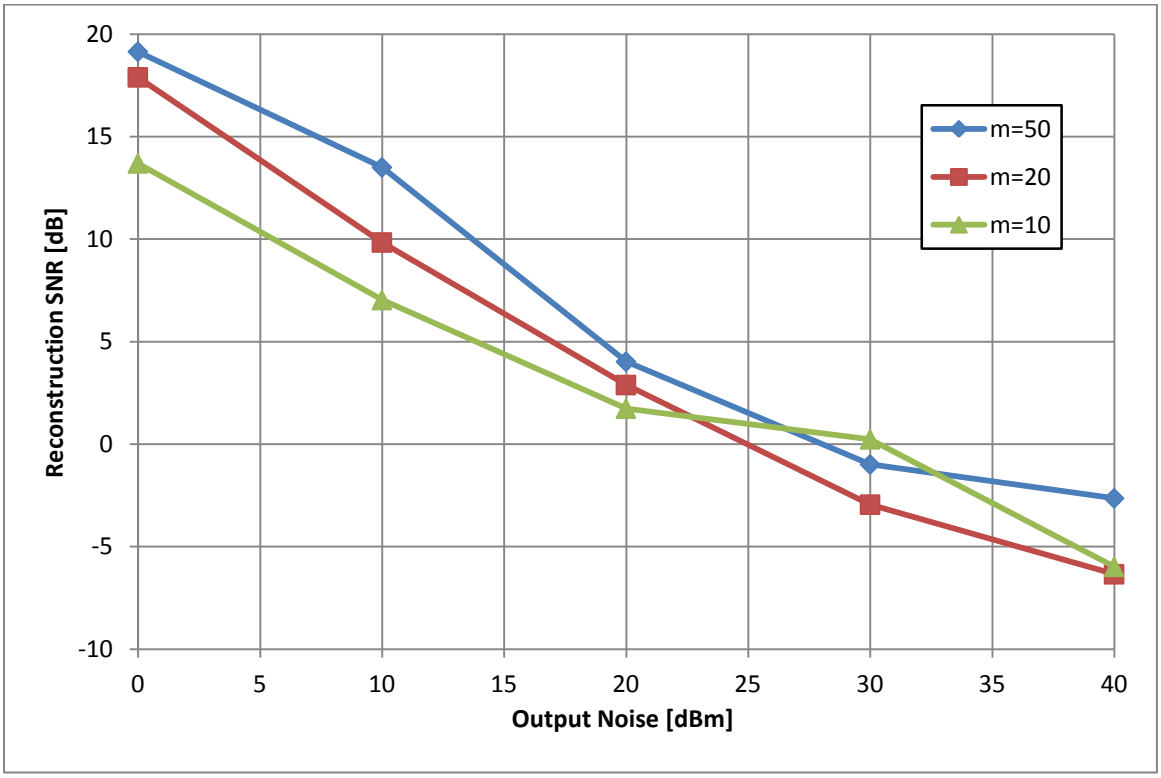

Figure 72 - Reconstruction SNR and ADC output noise

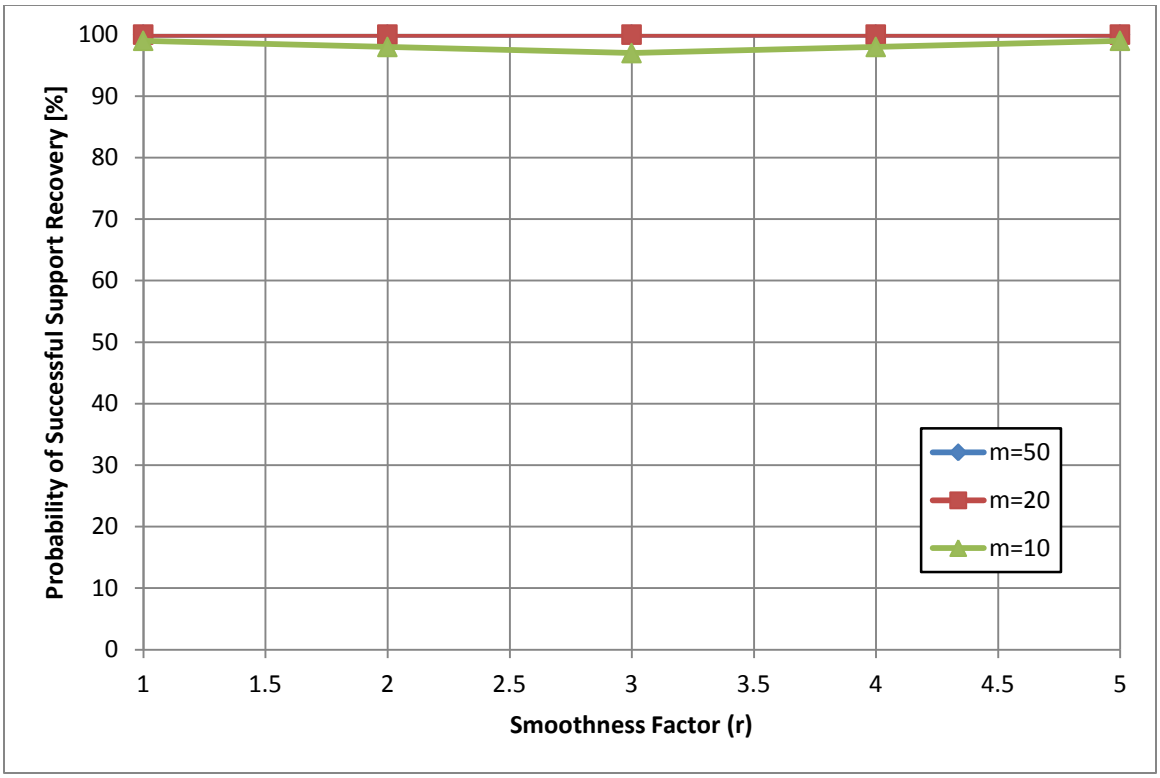

Figure 73 - Probability of success and ADC non-linearity

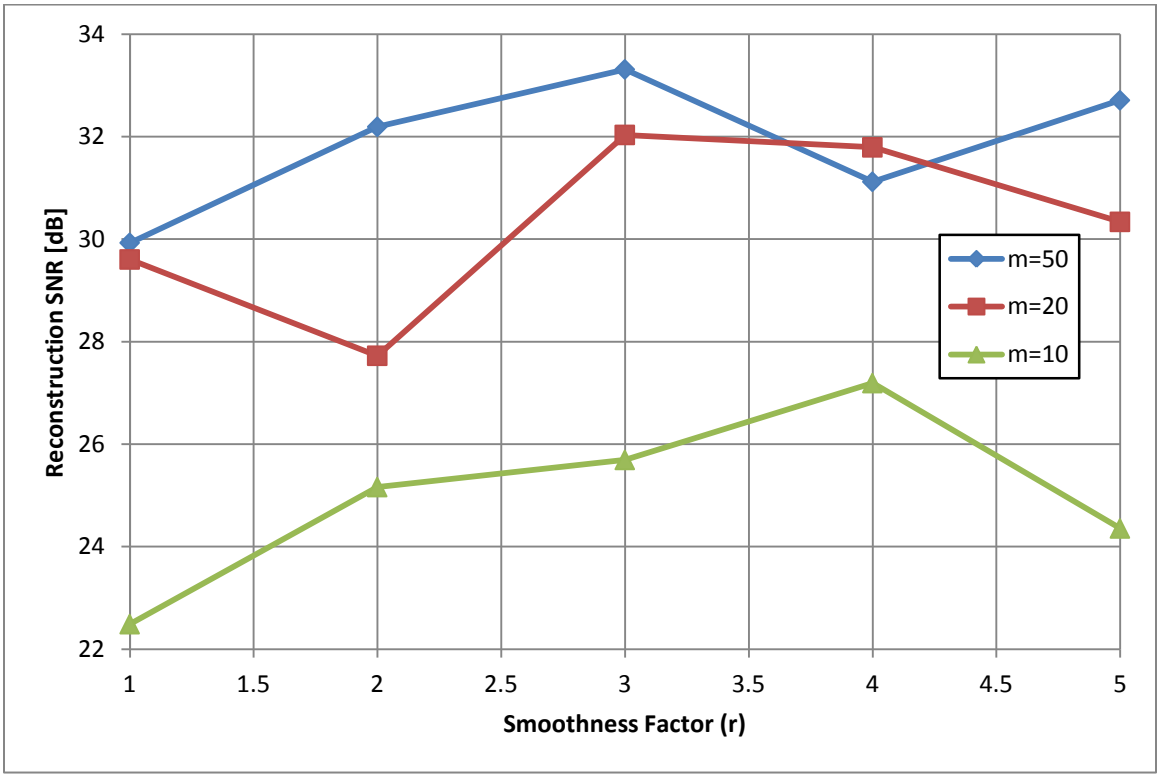

Figure 74 - Reconstruction SNR and ADC non-linearity

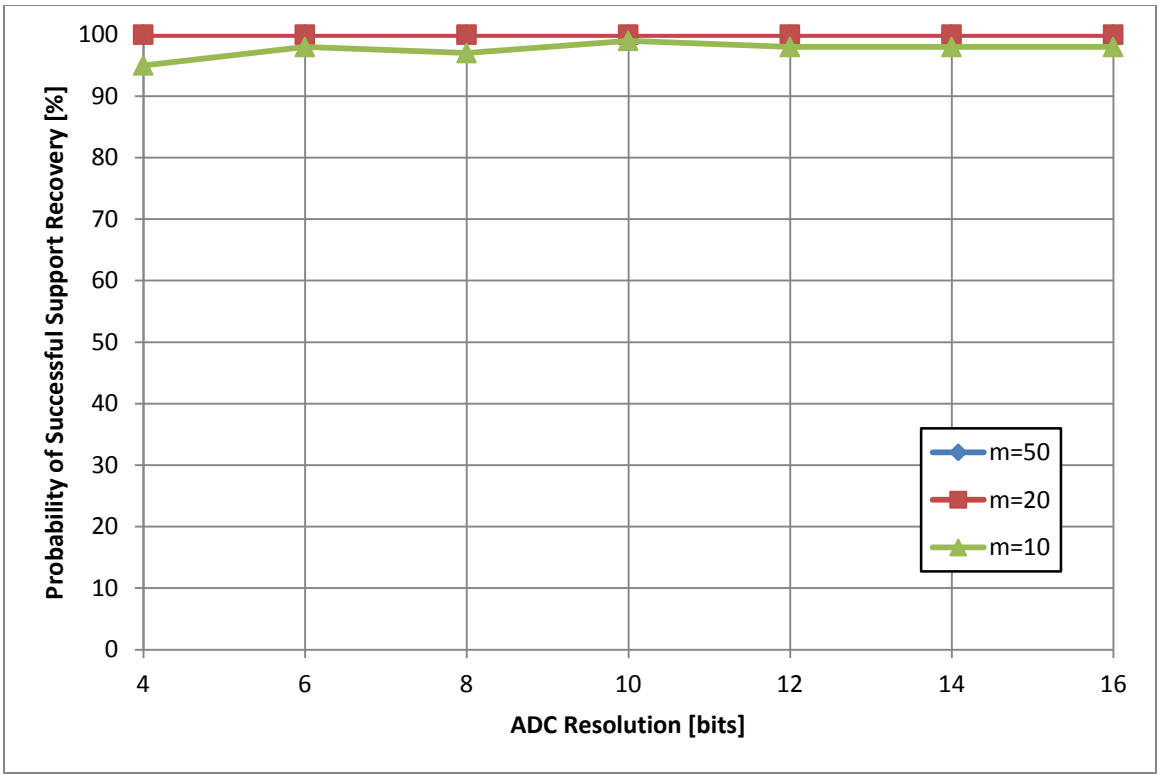

Figure 75 - Probability of success and ADC resolution

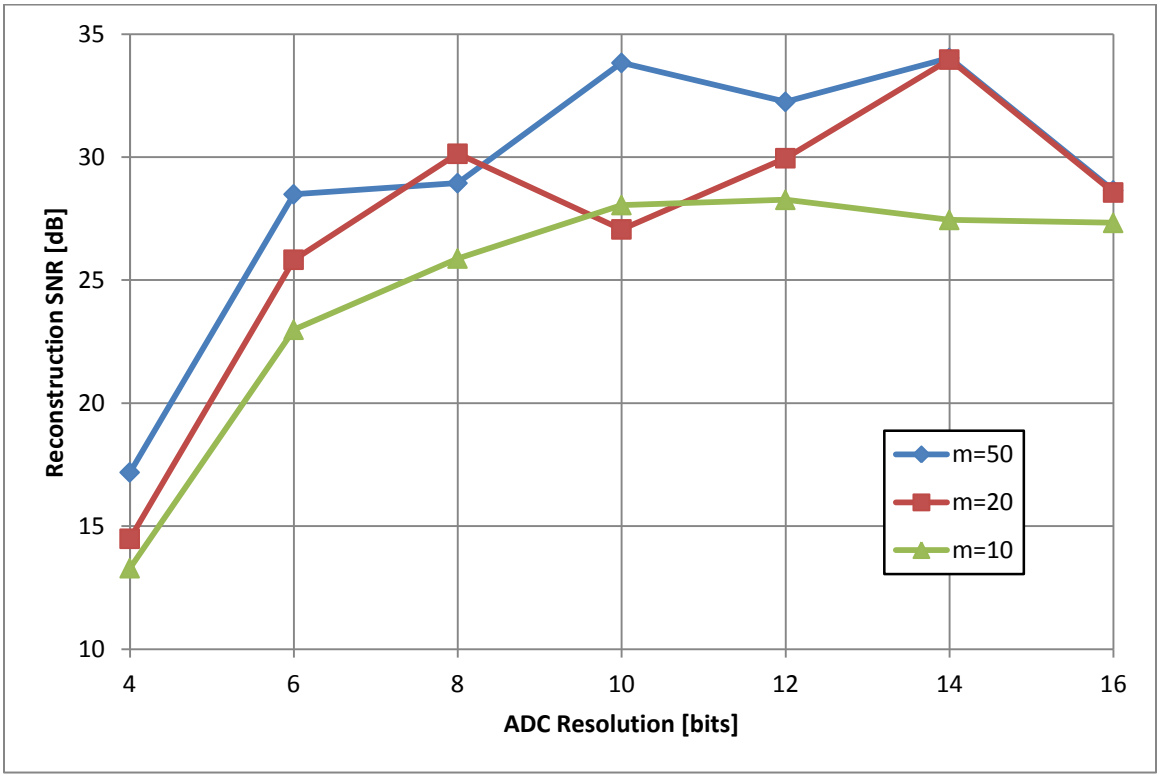

Figure 76 - Reconstruction SNR and ADC resolution

4. Channel Mismatches

Hardware impairments of each channel have been analyzed so far, however, mismatches in hardware between channels can also lead to error in signal reception. In Direct Conversion demodulators branch gain mismatch and timing delay can lead to considerable degradation in the received signal. [6] presents a Xampling based receiver architecture which is shown to be more robust to these impairments than traditional demodulator architectures. In this case the carrier frequency is assumed to be known before sampling, therefore, CS is not necessary to reconstruct the signal, since the spectral support can be directly computed from the known carrier frequency. Therefore for these impairments, probability of success was not included.

(1) Delay Mismatch

Delay mismatch is defined when periodic signals which are supposed to be synchronized may differ in phase by a fixed amount. For an IQ Direct Conversion demodulator this equates to a phase offset between the cos() and sin() used to demodulate the signal in each branch. In the Xampling based system, this equates to a random timing offset in the periodic mixing functions, $p_i(t)$, found in each branch. Figure 77 shows the average SNR as a function of the maximum timing offset defined as a percentage of $T_p/2M$, where T_p is the period of the mixing sequence and M is the number of alternations within one period of the mixing sequence.

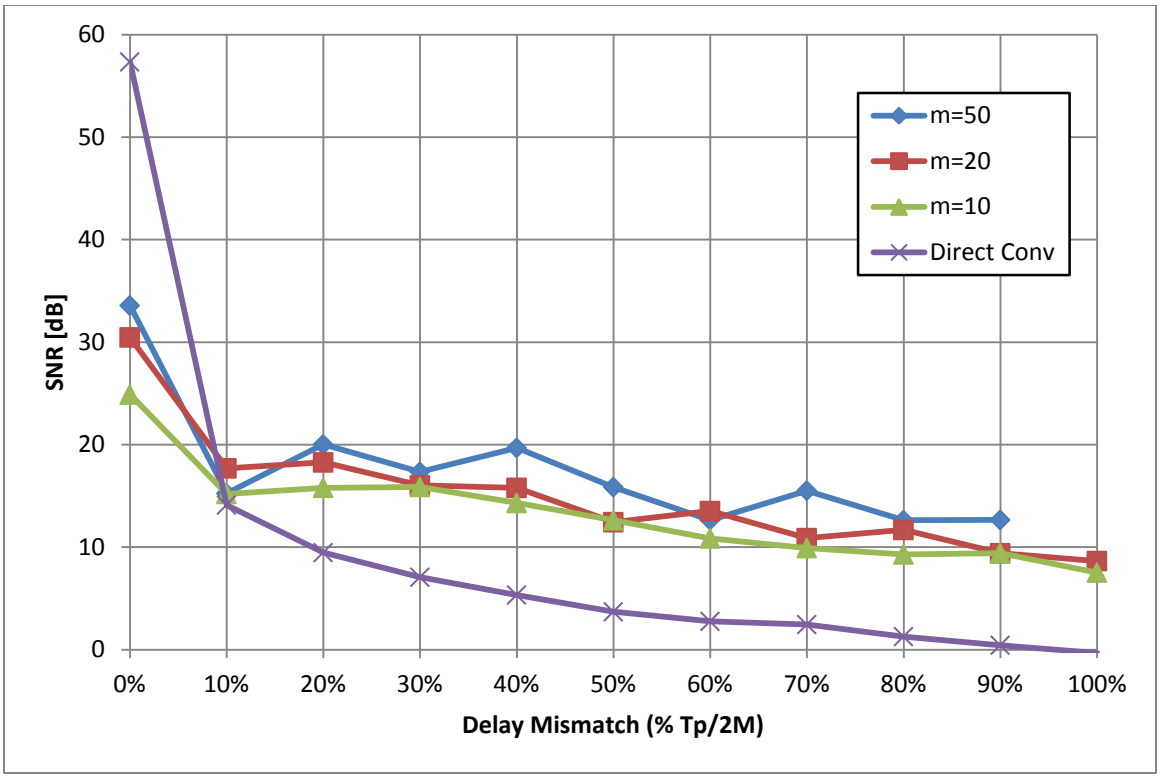

Figure 77 - SNR of IQ for different values of delay mismatch

These results are comparable with [6] and confirm that under ideal conditions, the traditional demodulation architecture yields a higher SNR. However, even under mild offset conditions, the SNR of the traditional demodulation drops off sharply while the MWC system experiences a much more gradual degradation. It should be noted that this may be due in part to the fact that the traditional demodulation is always subjected to the most severe offset, while the offset in each channel is random and within a bound dictated by the worst case. For a detailed analysis and mathematical proof of the effects of delay mismatch on the Xampling based system, refer to [6].

(2) Gain Mismatch

Another significant impairment in IQ based receivers is gain mismatch between the branches. In the Direct Conversion demodulator this is gain mismatch between the I and Q branches while in the Xampling based approach it is a random mismatch between filter gain in each channel, which

are nominally unity. Again the direct conversion system is subjected to the worst case mismatch while the each channel of the MWC approach sees a random mismatch within the bound of the worst case. Gain mismatch is given as a percentage, e.g. a mismatch of 20% would result in a ± 10% mismatch in the filter gain. The relationship between gain mismatch and average reconstruction SNR can be seen in Figure 78. For a detailed analysis and mathematical proof of the effects of gain mismatch on the MWC based system, refer to [6].

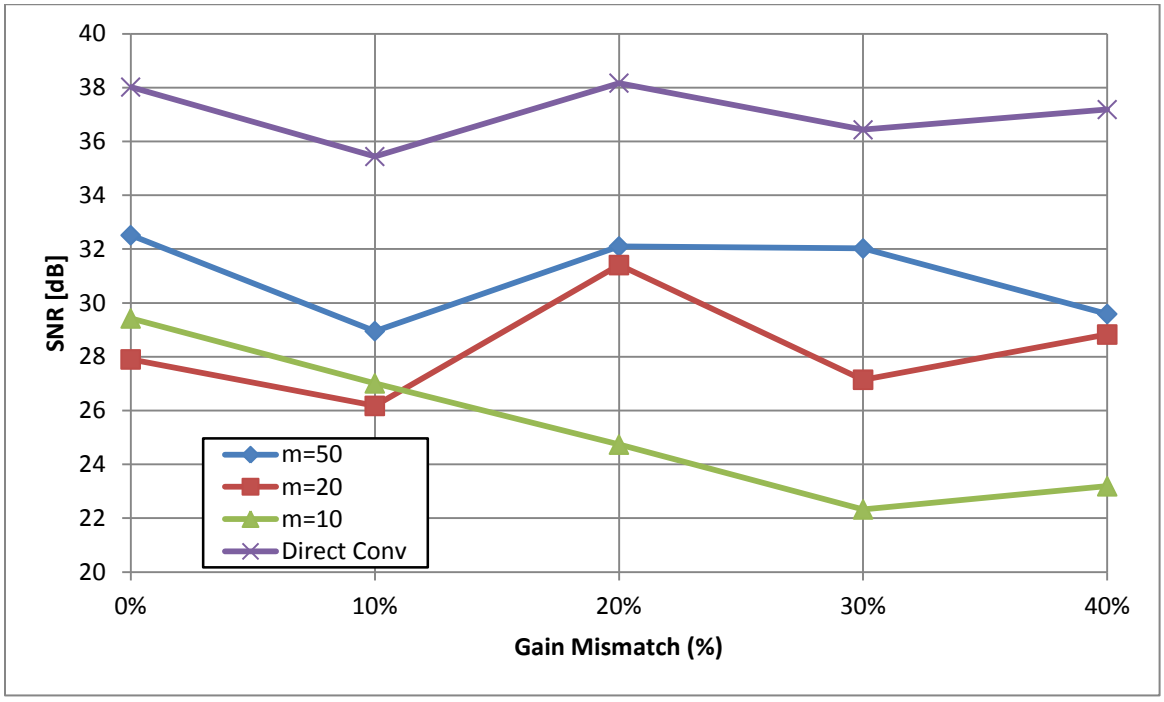

Figure 78 - Reconstruction SNR and channel gain mismatch

While these results do not exactly match those found in [6], where the SNR of both approaches were identical, they are still comparable. For example, [6] shows that a gain mismatch of between 5-10% should result in an SNR ~ 27.25 dB. The results in Figure 78 show that a system with 50 channels should see an SNR of approximately 30dB while Direct Conversion SNR should be around 35dB. These results also show that unlike timing mismatches, gain mismatches result in only a slight degradation in SNR across multiple channel counts and for the Direct Conversion receiver.

V. Conclusion

Presented were simulations related to the impact of various impairments in the Xampling based receivers found in [2] and [6]. In Section IV.B, channel hardware impairments were considered and it was shown that a certain number of channels above the theoretical minimum are required to ensure high probability of successful recovery of the support despite hardware impairments. In Section IV.C various CS reconstruction algorithms were considered and it was shown that OMP with an additional refinement step was the most robust to low input SNR when recovering the input signal support. In Section IV.D signal energy dynamic range was considered and it was shown that with a slight degradation in the probability of successful support recovery, signal energies can vary as much as three orders of magnitude. Given at least 30 channels, the degradation was shown to be less than 10%. Any larger ratios were shown to significantly degrade performance. In Section IV.E a 2.4GHz receiver was studied under various impairments and channel mismatches. Results were shown be similar to those found in [6] and Section IV.B.

Results generally agree to the expected tradeoff between channel count and system performance, that is, a system with more channels is expected to be more robust to impairments. This is likely due to the increased randomness associated with the addition of each channel, each of which has its own independently drawn periodic mixing sequence. While makes it difficult to predict the signal in any given channel, it varies the impact of impairments in each branch. The CTF block reduces the samples to a set whose size is proportional to the number of signal bands [32]. Therefore, if the channel count is large, then it can be assumed that there exists enough channels which were relatively unaltered by impairment and improves the overall chance of successful support recovery.

Comparing the results form Section IV.B and Section IV.E, one can conclude that while similar trends can be expected when considering various impairments, a system should be designed for the specific signal model and the intended application. Randomness in the signal combined with randomness in the system itself can make it difficult to predict exact system performance under impairments. The results of this provide preliminary insight into the impact of the impairments in Xampling based receivers.

VI. Appendix

A. Code Sample

The code used for the presented simulations is split into two main portions. The first is `start_tool.m`, which controls the overall simulation. A configuration file is also used, which is selected at runtime. The purpose of this file is to store various parameters specific to the simulation. Samples of both are included below.

1. start_tool.m

```
%% based on Demo for Modulated Wideband Converter, Version 1.0 [Eldar]

% cc : Control Case, set to be default output of testbench (configurable)

%% 0) system Intialization
clear all;
close all;
clc;

% add subdirectories to path
path(path, './models');                                 % system block models
path(path, './plotting');                               % functions used to plot
various signals
path(path, './config');                                 % config files
path(path, './CS');                                     % CS Solvers in this
folder
path(path, './CS/l1magic/Optimization');                % l1 magic functions
path(path, './CS/l1_ls_matlab');                        % l1 ls
path(path, './CS/cvx');
path(path, './CS/cvx/structures');
path(path, './CS/cvx/lib');
path(path, './CS/cvx/functions');
path(path, './CS/cvx/commands');
path(path, './CS/cvx/builtins');


% initialize config handle and config_opt
config_opt = 'none';
% config_handle = 'tb_config_IQ_no_noise_delaymm';
config_file_name = uigetfile('./config/*.m','Select testbench configuration
file');
config_name = config_file_name(1:end-2);        % rm '.m'

% record results
result_path_temp = strcat('./results/tool_output_', config_name);
result_path = strcat(result_path_temp, '.txt');
fid = fopen(result_path,'w');
trial_result_path_temp = strcat('./results/tool_trial_output_', config_name);
```

```
trial_result_path = strcat(trial_result_path_temp, '.txt');
fid_trial = fopen(trial_result_path,'w');


% iterators
countVar = 1;
countSNR = 1;
countChan = 1;
countTrial = 1;

% viewing options
showAll              = true;
showQuiet            = false;

showPrint            = true;
showParameterCheck   = false;
showModels           = false;
showPiSeq            = false;
showInputSignal      = false;
showInputNoise       = false;

showMixerOut         = false;
showMixerNI          = true;
showFilter           = false;
showFilterOut        = false;
showFilterNI         = false;
showADCOut           = false;
showADCNI            = false;

showSignalResults    = true;


visOnly = false;    % {T, F}: T single trial, F collect data

if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
    fprintf(1,'0) Initialize System ...\n');
    fprintf(1,'   results will be stored in %s for variable sweep\n',
result_path);
    fprintf(1,'   results will be stored in %s for results of each trial\n',
trial_result_path);
end


%% 1) ~ 3) loaded into tb_config files

if(visOnly)
    numTrials        = 1;    % number of trials to run
    sys_channels     = [50]; % number of system channels
    SNR_vals         = [10]; % SNR values to sweep (OVERALL SNR, in-band SNR
will be higher_
    var_sweep        = [0]; % user defined variable;; r_smooth
end
```

```
% evalueate parameters portion of config script
config_opt = 'parameters';
eval(config_name);
config_opt = 'none';

%% 4) Start Trials

fprintf(1,'START SIMULATION\n');

% SWEEP user def var
%     SWEEP SNR_vals
%         SWEEP sys_channels
%             ITER numTrials

% for countVar = 1:size(var_sweep,1)  % matrix input
for countVar = 1:length(var_sweep)  % matrix input
%% 4.0.1) use this section to sweep chosen variable

    % evaluate set variable portion of config script
    config_opt = 'set_var';
    eval(config_name);
    config_opt = 'none';


if((showPrint || showAll) && ~showQuiet)
   fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
   fprintf(1,'4.0.1) Set User Defined Variable ...\n');
   fprintf('   user variable set to: %d\n', var_sweep(countVar));
end

if(showQuiet)
   fprintf(1,'user var = %d\n', var_sweep(countVar));
end

%%
for countSNR = 1:length(SNR_vals)
%% 4.0.2) re-initialize SNR
    if(showQuiet)
       fprintf(1,'SNR = %3.1fdB\n', SNR_vals(countSNR));
    end
    SNR = SNR_vals(countSNR);       % note this is SNR calculated using
ENTIRE bandwidth

    % Store Reconstruction Results Each Trial
    suppResult = zeros (length(mChan),numTrials);               % Array used
to store result
    suppResultExact = zeros (length(mChan),numTrials);          % Array used
to store result, exact only
    suppResultFalsePos = zeros(length(mChan),numTrials);        % Array used
to store result with false detect
    CSRunTime = zeros(length(mChan), numTrials);

    % Store Reconstruction Results All Trials
```

```
    successProb = zeros(length(mChan),1);                          % Probability
of recovery
    successExactProb = zeros(length(mChan),1);                     % Probability
of recovery
    successFalseDetectProb = zeros(length(mChan),1);               % Probability
of recovery with false detect
    CSAveRunTime = zeros(length(mChan), 1);

    if((showPrint || showAll) && ~showQuiet)
       fprintf(1,'-------------------------------------------------------------
--------------------------------\n');
       fprintf(1,'4.0.2) Set SNR = %3.2f ...\n', SNR);
    end

%%
for countChan = 1:length(mChan)
%% 4.0.3) re-initialize current number of channels
   if(showQuiet)
        fprintf(1,'m = %d\n', mChan(countChan));
   end

    m = mChan(countChan);

    if((showPrint || showAll) && ~showQuiet)
        fprintf(1,'-------------------------------------------------------------
--------------------------------\n');
        fprintf(1,'4.0.3) Set number of Channels = %d ...\n', m);
    end

%%
for countTrial = 1:numTrials        % start individual trial
%% 4.0.4) set random seed
if(~isnan(UserSeed))
    if(useUserSeed == 1)
        UserSeed = ceil(2^16 * rand());
    end
     rng(UserSeed);
end

if((showPrint || showAll) && ~showQuiet )
    fprintf(1,'-------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.0.4) Set random seed ... \n');
    if (~isnan(UserSeed))
        fprintf(1,'   UserSeed = %d \n', UserSeed);
    end
end


%% 4.1) Generate sign alternating sequence
if((showPrint || showAll) && ~showQuiet )
    fprintf(1,'-------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.1) Generate sign alternating sequence ... \n');
    fprintf(1,'    Matching sequences? : %d\n',useMatchingPi);
end
```

```
SignPatterns = zeros(m, M);
SignPatterns_cc = zeros(m, M);

myPiSequence = [];
SignPatterns = GeneratePi(m,M,r, code_type, myPiSequence);
switch useMatchingPi
    case false
        SignPatterns_cc = GeneratePi(m,M,r,'rand', myPiSequence);
    case true
        SignPatterns_cc = SignPatterns;
    otherwise
        fprintf('ERROR: choice for matching pi sequences not recognized,
defaulting to matching...\n');
        SignPatterns_cc = SignPatterns;
end


% plot sign alternating function, one period
if ((showPiSeq || showAll) && ~showQuiet)
    plotPi(SignPatterns, SignPatterns_cc, M, 1,Tp, TimeResolution);
end


%% 4.2) Generate Signal

if((showPrint || showAll) && ~showQuiet)
    fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.2) Generating signal...\n');
end

% Signal Generation, details stored in GenerateSignal SCRIPT (not function
due to large number of variables
x = zeros(size(t_axis_sig));
x_components = zeros(N/2, length(x));          % used to store each signal
which makes up x


eval('GenerateSignal');


% Calculate original support set
Sorig = [];
% explain:  we take the starting edges: fi-B/2  and divide by fp. minus
% 0.5 to shift half fp towards zero. then M0+1 is to move 0 = M0+1.
Starts = ceil((fi-B/2)/fp-0.5+L0+1);
Ends = ceil((fi+B/2)/fp-0.5+L0+1);
for n=1:(N/2)
    Sorig = union (Sorig,  Starts(n):Ends(n));
end

% Now add the negative frequencies
Sorig = union(Sorig, L+1-Sorig);
Sorig = sort(Sorig);
```

```
% scale signal according to specified full scale
x_max = max(max(x), abs(min((x))));
x = (x./(x_max)).*(volt_fs/2);

for n=1:(N/2)
    x_components(n,:) =
(x_components(n,:)/max(x_components(n,:))).*(volt_fs/2);
end

% % calculate and show signal power
% power_x = norm(x)^2/length(x)

x_no_noise = x;                     % save ideal copy of input signal
x_cc = x;

if((showInputSignal || showAll) && ~showQuiet)
    plotInputSignal(x, x_components, fnyq, t_axis, TimeResolution);
end

if((showPrint || showAll) && ~showQuiet)
    fprintf(1,'Signal Details\n');
    fprintf(1,'   SNRin=%3.2f\n', SNR);
    fprintf(1,'   fc=[ ');
    fprintf(1,'%g ',fi(1,:)/1e6);
    fprintf(1,'] MHz, \n');
    fprintf(1,'   Ei=[ ');
    fprintf(1,'%g ',Ei(1,:));
    fprintf(1,']\n');
end


%% 4.3) Noise Generation and Energy Calculation
if((showPrint || showAll) && ~showQuiet)
    fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.3) Generate Noise...\n');
end

%%% (only used for 'digital' noise)
% generate noise
noise_nyq = randn(1,(K+K0)*L);              % Generate white Gaussian nosie
within [-fnyq,fnyq]
noise = interpft(noise_nyq, R*(K+K0)*L);    % Interpolate into a finer grid
(the same length as the signal)
noise_cc = noise;
% Calculate energies
NoiseEnergy = norm(noise)^2;
SignalEnergy = norm(x_no_noise)^2;
CurrentSNR = SignalEnergy/NoiseEnergy;
% set current SNR
SNR_linear = 10^(SNR/10);          % not dB
%%%

% Add noise
x_a_noise = awgn(x_no_noise,SNR,'measured');
```

```
if(strcmp(noise_type,'analog'))
    x = x_a_noise;
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'   Analog Noise selected\n');
    end
end

if(strcmp(noise_type_cc,'analog'))
    x_cc = x_a_noise;
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'   Analog Noise selected (control)\n');
    end
end

if((showInputNoise || showAll) && ~showQuiet)
    plotInputNoise(x_no_noise, x, x_cc, t_axis, TimeResolution);
end


%% 4.4) Mix Signal with Pi mixing sequence
if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.4) Mix Signal with sign alternating sequence...\n');
    fprintf(1,'\n');
    fprintf(1,'Applying mixer non-idealities ...\n');
    if(ischar(m_mixer.noise_dbm))
        fprintf('   noise = %s\n',m_mixer.noise_dbm);
    else
        fprintf('   noise = %3.2f\n',m_mixer.noise_dbm);
    end
    fprintf(1,'   non-linear type = %s\n', m_mixer.nl.type);
    fprintf(1,'   max channel delay = %3.1f %% \n', (m_mixer.delay_max)*100);
end

% variable allocation for mixer output
MixedSigSequences           = zeros(m,length(t_axis));
MixedNoiseSequences         = zeros(m,length(t_axis));
MixedSigSequences_cc        = zeros(m,length(t_axis));
MixedNoiseSequences_cc      = zeros(m,length(t_axis));
MixedSigSequences_nodelay   = zeros(m,length(t_axis));        % copy with no
delay
MixedSigSequences_nodelay_cc = zeros(m,length(t_axis));     % copy with no
delay


% generate random channel timing delays based on threshold set in model_mixer
delay=m_mixer.delay_max*quant((rand(1,m)-0.5)*(Tp/M), TimeResolution);
delay_cc=m_mixer_cc.delay_max*quant((rand(1,m)-0.5)*(Tp/M), TimeResolution);
% delay_cc = zeros(1,m);
% delay = -ones(1,m)*1e-12*11;

for channel=1:m
    % mix signal with channel delay mismatch
```

```
    MixedSigSequences(channel,:)        =
DelayMixSignal(x,t_axis,SignPatterns(channel,:),Tp, delay(channel));
    MixedNoiseSequences(channel,:)      =
DelayMixSignal(noise,t_axis,SignPatterns(channel,:),Tp, delay(channel));
    MixedSigSequences_cc(channel,:)     =
DelayMixSignal(x_cc,t_axis,SignPatterns_cc(channel,:),Tp, delay_cc(channel));
    MixedNoiseSequences_cc(channel,:)   =
DelayMixSignal(noise_cc,t_axis,SignPatterns_cc(channel,:),Tp,
delay_cc(channel));
    % create versions with no delay
    MixedSigSequences_nodelay(channel,:)    =
DelayMixSignal(x,t_axis,SignPatterns(channel,:),Tp, 0);
    MixedSigSequences_nodelay_cc(channel,:) =
DelayMixSignal(x_cc,t_axis,SignPatterns(channel,:),Tp, 0);

    % apply non-linearity
    MixedSigSequences(channel,:)    =
m_mixer.nl.applyNLGain(MixedSigSequences(channel,:));
    MixedSigSequences_cc(channel,:) =
m_mixer_cc.nl.applyNLGain(MixedSigSequences_cc(channel,:));

    % add noise to signal
    MixedSigSequences(channel,:)    =
m_mixer.addNoise(MixedSigSequences(channel,:));
    MixedSigSequences_cc(channel,:) =
m_mixer_cc.addNoise(MixedSigSequences_cc(channel,:));
end

% plot mixer output
if ((showMixerOut || showAll) && ~showQuiet)
    chanNum = 1;
    plotMixerOut(chanNum, M, R, x,
SignPatterns(chanNum,:),MixedSigSequences_nodelay(chanNum,:),x_cc,SignPattern
s_cc(chanNum,:), ...
                    MixedSigSequences_nodelay_cc(chanNum,:), t_axis, Tp,
TimeResolution);
end
if ((showMixerNI || showAll) && ~showQuiet)
    chanNum = 2;
    plotMixerNI(chanNum, M,R, Tp, m_mixer, m_mixer_cc,
MixedSigSequences(chanNum,:),
MixedSigSequences_cc(chanNum,:),delay(1,chanNum), delay_cc(1,chanNum), ...
                    SignPatterns(chanNum,:),
SignPatterns_cc(chanNum,:),t_axis,TimeResolution);
end

%% 4.5) low-pass filtering (gain mismatch introducded here)
if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.5) Filtering signal ...\n');
    fprintf('   filter type = %s\n',m_lpf.type);
    fprintf(1,'\n');
    fprintf(1,'Applying filter non-idealities ...\n');
    if(ischar(m_lpf.noise_dbm))
        fprintf('   noise = %s\n',m_lpf.noise_dbm);
    else
```

```
        fprintf('   noise = %3.2f\n',m_lpf.noise_dbm);
    end
    fprintf(1,'   non-linear type = %s\n', m_lpf.nl.type);
    fprintf(1,'   max gain mismatch = +/-%3.1f %% \n\n',
(m_lpf.gain_mm_max)*100/2);
end

% allocate variables to store filter output
filter_out          = zeros(m,length(MixedSigSequences));
filter_out_noise    = zeros(m,length(MixedNoiseSequences));
filter_out_cc       = zeros(m,length(MixedSigSequences_cc));
filter_out_noise_cc = zeros(m,length(MixedNoiseSequences_cc));

% Generate Low Pass Filter coefficients
m_lpf.generateFilters(K+K0, t_axis, R, L, fnyq);
m_lpf_cc.generateFilters(K+K0, t_axis, R, L, fnyq);

% generate gain mismatch per channel
gain_mm = 1 + (m_lpf.gain_mm_max.*(rand(m,1)-0.5));               % zero mean
gain_mm_cc = 1 + (m_lpf_cc.gain_mm_max.*(rand(m,1)-0.5));         % zero mean

% filter signal
if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'    Filtering Channel ');
end
for channel = 1:m
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'.');
        if ( (mod(channel,5)==0) || (channel==m)) fprintf(1,'%d',channel);
end
        if (channel == m) fprintf(1,'\n'); end
    end
    SignalSequence      =  MixedSigSequences(channel,:);
    NoiseSequence       =  MixedNoiseSequences(channel,:);
    SignalSequence_cc   =  MixedSigSequences_cc(channel,:);
    NoiseSequence_cc    =  MixedNoiseSequences_cc(channel,:);

    % filter signal
    zeroPhaseFiltering = true;
    filter_out(channel, :)          = m_lpf.filterSignal(SignalSequence,
zeroPhaseFiltering);
    filter_out_noise(channel, :)    = m_lpf.filterSignal(NoiseSequence,
zeroPhaseFiltering);
    filter_out_cc(channel, :)       =
m_lpf_cc.filterSignal(SignalSequence_cc, zeroPhaseFiltering);
    filter_out_noise_cc(channel, :) = m_lpf_cc.filterSignal(NoiseSequence_cc,
zeroPhaseFiltering);

    % apply channel gain mismatch
    filter_out(channel,:)       = filter_out(channel,:).*gain_mm(channel);
    filter_out_cc(channel,:)    =
filter_out_cc(channel,:).*gain_mm_cc(channel);

    % apply filter non-linearity
    filter_out(channel,:)       =
m_lpf.nl.applyNLGain(filter_out(channel,:));
```

```
    filter_out_cc(channel,:)    =
m_lpf_cc.nl.applyNLGain(filter_out_cc(channel,:));

    % apply filter noise
    filter_out(channel,:)       = m_lpf.addNoise(filter_out(channel,:));
    filter_out_cc(channel,:)    =
m_lpf_cc.addNoise(filter_out_cc(channel,:));

end

% show results
if((showFilter || showAll) && ~showQuiet)
    plotFilter(m_lpf, m_lpf_cc, t_axis, TimeResolution);
end

if((showFilterOut || showAll) && ~showQuiet)
    chanNum = 2;
    plotFilterOut(chanNum,MixedSigSequences(chanNum,:),filter_out(chanNum,
:), MixedSigSequences_cc(chanNum,:),filter_out_cc(chanNum, :), ...
                    m_lpf, m_lpf_cc, t_axis, TimeResolution);
end

if((showFilterNI || showAll) && ~showQuiet)
    chanNum = 3;
    plotFilterNI(chanNum, m_lpf, m_lpf_cc, gain_mm, filter_out(chanNum,
:),filter_out_cc(chanNum, :),t_axis, TimeResolution);
end



%% 4.6) Sample Signal
if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'-----------------------------------------------------------------
------------------------------\n');
    fprintf(1,'4.6) Sampling signal ...\n');
    fprintf('   resolution = %d bits\n',m_adc.res);
    fprintf('   quantization level = %f\n',m_adc.quant_step);
    fprintf(1,'\n');
    fprintf(1,'Applying ADC non-idealities ...\n');
    if(ischar(m_adc.noise_dbm))
        fprintf('   noise = %s\n',m_adc.noise_dbm);
    else
        fprintf('   noise = %3.2f\n',m_adc.noise_dbm);
    end
    fprintf(1,'   non-linear type = %s\n', m_adc.nl.type);
end

% decimation factor
decfactor = L*R;
% scale time axis
Digital_time_axis = downsample(t_axis,decfactor);
DigitalLength = length(Digital_time_axis);

% allocate variables to store filter output
DigitalSignalSamples        = zeros(m,DigitalLength);
DigitalNoiseSamples         = zeros(m,DigitalLength);
```

```
DigitalSignalSamples_cc     = zeros(m,DigitalLength);
DigitalNoiseSamples_cc      = zeros(m,DigitalLength);

% sample signal
if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'    Sampling Channel ');
end
for channel = 1:m
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'.');
        if ( (mod(channel,5)==0) || (channel==m)) fprintf(1,'%d',channel);
end
        if (channel == m) fprintf(1,'\n'); end
    end

    % sample signal (downsample)
    DigitalSignalSamples(channel, :)    =
m_adc.sampleSignal(filter_out(channel, :), decfactor);
    DigitalSignalSamples_cc(channel, :) =
m_adc_cc.sampleSignal(filter_out_cc(channel, :), decfactor);
    DigitalNoiseSamples(channel, :)     =
m_adc_cc.sampleSignal(filter_out_noise(channel, :), decfactor);
    DigitalNoiseSamples_cc(channel, :)  =
m_adc_cc.sampleSignal(filter_out_noise_cc(channel, :), decfactor);

    % apply non-linearity
    DigitalSignalSamples(channel,:)     =
m_adc.nl.applyNLGain(DigitalSignalSamples(channel,:));
    DigitalSignalSamples_cc(channel,:)  =
m_adc_cc.nl.applyNLGain(DigitalSignalSamples_cc(channel,:));

    % apply quantization
    if(m_adc.res ~= inf)
        DigitalSignalSamples(channel,:)     =
m_adc.applyQuant(DigitalSignalSamples(channel,:));
    end
    if(m_adc_cc.res ~= inf)
        DigitalSignalSamples_cc(channel,:)  =
m_adc_cc.applyQuant(DigitalSignalSamples_cc(channel,:));
    end
    % add noise
    DigitalSignalSamples(channel,:)     =
m_adc.addNoise(DigitalSignalSamples(channel,:));
    DigitalSignalSamples_cc(channel,:)  =
m_adc_cc.addNoise(DigitalSignalSamples_cc(channel,:));
end

if((showADCOut || showAll) && ~showQuiet)
    chanNum = 4;
    plotADCOut(chanNum, filter_out(chanNum, :),DigitalSignalSamples(chanNum,
:), filter_out_cc(chanNum, :),DigitalSignalSamples_cc(chanNum, :),...
                    t_axis, Digital_time_axis, Ts, TimeResolution);
end

if((showADCNI || showAll) && ~showQuiet)
    chanNum = 4;
```

```
    plotADCNI(chanNum, m_adc, m_adc_cc, DigitalSignalSamples(chanNum, :),
DigitalSignalSamples_cc(chanNum, :),...
                    Digital_time_axis, Ts);
end

if ((showPrint || showAll) && ~showQuiet)
    fprintf(1,'\n-----------------------------------------------------------------
--------------------------------\n');
    fprintf(1,'Sampling summary\n');
    fprintf(1,'   %d channels, each gives %d samples\n',m,DigitalLength);
end


%% 4.7) CTF block

if ((showPrint || showAll) && ~showQuiet)
    fprintf('-----------------------------------------------------------------
-----------------------------\n');
    fprintf('4.7) Entering Continuous-to-Finite (CTF) block ...\n');
end

% define matrices for fs=fp
S = SignPatterns;
S_cc = SignPatterns_cc;
theta = exp(-j*2*pi/L);
F = theta.^([0:L-1]'*[-L0:L0]);
np = 1:L0;
nn = (-L0):1:-1;
% This is for digital input only. Note that when R -> infinity,
% D then coincides with that of the paper
dn = [   (1-theta.^nn)./(1-theta.^(nn/R))/(L*R)      1/L    (1-
theta.^np)./(1-theta.^(np/R))/(L*R)];
D = diag(dn);
A = S*F*D;
A = conj(A);
A_cc = S_cc*F*D;
A_cc = conj(A_cc);

% combine signal and noise
if(strcmp(noise_type, 'digital'))
    DigitalSamples = DigitalSignalSamples +
DigitalNoiseSamples*sqrt(CurrentSNR/SNR_linear);
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'Digital Noise  selected\n');
    end
else
    DigitalSamples = DigitalSignalSamples;
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'No Noise or analog noise selected\n');
    end
end

if(strcmp(noise_type_cc, 'digital'))
    DigitalSamples_cc = DigitalSignalSamples_cc +
DigitalNoiseSamples_cc*sqrt(CurrentSNR/SNR_linear);
    if ((showPrint || showAll) && ~showQuiet)
```

```
        fprintf(1,'Digital Noise  selected (control)\n');
    end
else
    DigitalSamples_cc = DigitalSignalSamples_cc;
    if ((showPrint || showAll) && ~showQuiet)
        fprintf(1,'No Noise or analog noise selected (control)\n');
    end
end


% Frame construction
Q = DigitalSamples* DigitalSamples';
Q_cc = DigitalSamples_cc*DigitalSamples_cc';
% decompose Q to find frame V (reduces all samples to 2N)
NumDomEigVals= FindNonZeroValues(eig(Q),5e-8);          %% look into this
NumDomEigVals_cc = FindNonZeroValues(eig(Q_cc),5e-8);
[V,d] = eig_r(Q,min(NumDomEigVals,2*N));
[V_cc,d_cc] = eig_r(Q_cc,min(NumDomEigVals_cc,2*N));
v = V*diag(sqrt(d));
v_cc = V_cc*diag(sqrt(d_cc));


%% 4.8) Support Recovery via Compressed Sensing
if ((showPrint || showAll) && ~showQuiet)
    fprintf('-------------------------------------------------------------------
----------------------------\n');
    fprintf('4.8) Recovering Support using CS ...\n');
end

% choose reconstruction method

switch reconstruction_choice

    case 1
        tic;
        [u, RecSupp] = RunOMP_Unnormalized(v, A, 2*N, 0, 0.01, false);
        CSRunTime(countChan, countTrial) = toc;
    case 2
        tic;
        [u, RecSupp] = RunOMP_Unnormalized(v, A, N, 0, 0.01, true);
        CSRunTime(countChan, countTrial) = toc;
    case 3
        tic;
        [RecSupp] = RunOMP_forMB(v, A, N, Rec_maxiter);
        CSRunTime(countChan, countTrial) = toc;
    case 4
        tic;
        [RecSupp u] = RunL1_qc(DigitalSamples,A,L,N, 0);
        CSRunTime(countChan, countTrial) = toc;
    case 5
        tic;
        [RecSupp u] = RunL1_cvx(A, v, N);
        CSRunTime(countChan, countTrial) = toc;
    case 6
        tic;
        [u, RecSupp] = RunCoSaMP(v, A, N, Rec_maxiter, 0, 0.001, false);
```

```
        CSRunTime(countChan, countTrial) = toc;
    case 7
        tic;
        search_N = 2*N;
        [u, RecSupp, N_Res] = RunCoSaMP_findN(v, A, N, search_N, 0, 0.001,
false);
        CSRunTime(countChan, countTrial) = toc;
    otherwise
        fprintf('ERROR: reconstruction_choice not recognized, defaulting to
option ''1''\n...');
        tic;
        [u, RecSupp] = RunOMP_Unnormalized(v, A, 2*N, 0, 0.01, false);
        CSRunTime(countChan, countTrial) = toc;
end


% choose reconstruction method (control case)
switch reconstruction_choice_cc
    case 1
        [u_cc, RecSupp_cc] = RunOMP_Unnormalized(v_cc, A_cc, 2*N, 0, 0.01,
false);
    case 2
        [u_cc, RecSupp_cc] = RunOMP_Unnormalized(v_cc, A_cc, N, 0, 0.01,
true);

    case 3
        [RecSupp_cc] = RunOMP_forMB(v_cc, A_cc, N, 100);

    case 4
        [RecSupp_cc u_cc] = RunL1_qc(DigitalSamples_cc,A_cc,L,N, 0);

    case 5
        [RecSupp_cc u_cc] = RunL1_cvx(A_cc, v_cc, N);


    case 6
        [u_cc, RecSupp_cc] = RunCoSaMP(v_cc, A_cc, N, Rec_maxiter, 0, 0.001,
false);
    otherwise
        fprintf('ERROR: reconstruction_choice not recognized, defaulting to
option ''1''\n...');
end


% sort recovered support
if(knownSupport)
    RecSuppSorted = Sorig;
else
    RecSuppSorted = sort(unique(RecSupp));
end

if(knownSupport_cc)
    RecSuppSorted_cc = Sorig;
else
    RecSuppSorted_cc = sort(unique(RecSupp_cc));
end
```

```
%newRecSupp = RefineSupport(v, A, N, RecSuppSorted);

% Decide on success
% Success:
% 1) recovered support is/iscontainted withing the original support
% 2) A matrix has full rank (?)
if (is_contained(Sorig,RecSuppSorted) == 2  && (rank(A(:,RecSuppSorted)) ==
length(RecSuppSorted)))
    Success=1;
    if(showQuiet)
        fprintf(1,'o');              % successful and exact recovery
    end
    if((showPrint || showAll) && ~showQuiet)
        if(knownSupport)
            fprintf(1,'   Support Known before Sensing...\n');
% successful and exact recovery
        else
            fprintf(1,'   Successful Support Recovery: Exact Match\n');
% successful and exact recovery
        end
    end
    suppResult(countChan,countTrial)=1;
    suppResultExact(countChan, countTrial)=1;
elseif (is_contained(Sorig,RecSuppSorted) == 1  && (rank(A(:,RecSuppSorted))
== length(RecSuppSorted)))
    Success=1;
    if(showQuiet)
        fprintf(1,'f');              % successful and exact recovery
    end
    if((showPrint || showAll) && ~showQuiet)
        fprintf(1,'   Successful Support Recovery: False Detect\n');
% successful and exact recovery
    end
    suppResult(countChan,countTrial)=1;
    suppResultFalsePos(countChan,countTrial) = 1;
else
    Success=0;
    %fprintf('Failed support recovery\n');
    if(showQuiet)
        fprintf(1,'x');
    end
    if((showPrint || showAll) && ~showQuiet)
        fprintf(1,'   Failed Support Recovery\n');              % successful
and exact recovery
    end
end

 if((showPrint || showAll) && ~showQuiet)
    fprintf(1,'   Support          = [ ');
    fprintf(1,'%g ',Sorig(1,:));
    fprintf(1,' ]\n');
    fprintf(1,'   Rec Support      = [ ');
    fprintf(1,'%g ',RecSuppSorted(1,:));
    fprintf(1,' ]\n');
    fprintf(1,'   Rec Support (cc) = [ ');
```

```
    fprintf(1,'%g ',RecSuppSorted_cc(1,:));
    fprintf(1,' ]\n');
 end

%% 4.9) Reconstruct Signal using Recovered Support
if ((showPrint || showAll) && ~showQuiet)
    fprintf('-------------------------------------------------------------------
----------------------------\n');
    fprintf('4.9) Reconstructing Signal using Recovered Support ...\n');
end

% allocate variables
RecCarriers = zeros(1,length(RecSuppSorted));
RecCarriers_cc = zeros(1,length(RecSuppSorted_cc));
x_rec = zeros(1,length(x));
x_rec_cc = zeros(1,length(x_cc));

A_S = A(:,RecSuppSorted);
A_S_cc = A_cc(:, RecSuppSorted_cc);
hat_zn = pinv(A_S)*DigitalSamples;               % inverting A_S
hat_zn_cc = pinv(A_S_cc)*DigitalSamples_cc;      % inverting A_S

hat_zt = zeros(size(hat_zn,1),length(x));
hat_zt_cc = zeros(size(hat_zn_cc,1),length(x_cc));

for ii = 1:size(hat_zt,1)                       % interpolate (by sinc)
    hat_zt(ii,:) = interpft(hat_zn(ii,:),L*R*length(hat_zn(ii,:)));
end

for ii = 1:size(hat_zt_cc,1)                      % interpolate (by sinc)
    hat_zt_cc(ii,:) = interpft(hat_zn_cc(ii,:),L*R*length(hat_zn_cc(ii,:)));
end

for ii = 1:size(hat_zt,1)                        % modulate each band to their
corresponding carriers
    x_rec = x_rec+hat_zt(ii,:).*exp(j*2*pi*(RecSuppSorted(ii)-L0-
1)*fp.*t_axis);
end
for ii = 1:size(hat_zt_cc,1)                       % modulate each band to
their corresponding carriers
    x_rec_cc = x_rec_cc+hat_zt_cc(ii,:).*exp(j*2*pi*(RecSuppSorted_cc(ii)-L0-
1)*fp.*t_axis);
end

x_rec = real(x_rec);
x_rec_cc = real(x_rec_cc);

% add noise to original signal
x_d_noise = x_no_noise + noise*sqrt(CurrentSNR/SNR_linear);



%% Trial Results: Signal Analysis & Plots
%% 4.10) Trial Results: Signal Analysis & Plots
if ((showPrint || showAll) && ~showQuiet)
```

```
    fprintf('-------------------------------------------------------------------
-----------------------------\n');
    fprintf('4.10) Processing Trial Results ...\n');
end

%%%% signal variables %%%%

% x               : input signal with selected options
% x_no_noise      : input signal with no noise
% x_d_noise       : input signal with 'digital' noise (same as original demo)
% x_a_noise       : input signal with analog noise (awgn())
% x_rec           : reconstructed signal
% x_rec_cc        : reconstructed signal, control case


% run portion of config script to process data
config_opt = 'process_trial_result';
eval(config_name);
config_opt = 'none';


% write to the file at the end of a trial (SNR)
config_opt = 'write_trial_result';
eval(config_name);
config_opt = 'none';


end %% number of Trials

if(showQuiet)
    fprintf('\n');
end


end %% number of Channel




%% 5) Process Results from All Channels and Trials
if ((showPrint || showAll) && ~showQuiet)
    fprintf('-------------------------------------------------------------------
-----------------------------\n');
    fprintf('5) Processing Results from All Channels and Trials ...\n');
    fprintf (1, 'For an SNR of ')
    fprintf (1, '%d',SNR)
    fprintf (1, ' and ')
    fprintf (1, '%d',numTrials)
    fprintf (1, ' trials\n')

end

for n = 1:length(mChan)
```

```
    successProb(n,1) = 100*nnz(suppResult(n,:))/numTrials;
% Calculate Probability of Success, overall
    successExactProb(n,1) = 100*nnz(suppResultExact(n,:))/numTrials;
% Calculate Probability of Success, exact only
    successFalseDetectProb(n,1) = 100*nnz(suppResultFalsePos(n,:))/numTrials;
% Calculate Probability of Success, with false detect
    CSAveRunTime(n,1) = sum(CSRunTime(n,:))/numTrials;

    if ((showPrint || showAll) && ~showQuiet)
        fprintf (1, 'For m = %d\n',mChan(n));
        fprintf (1, 'Probability of successful Support, Overall is %d%%\n',
successProb(n,1));
        fprintf (1, 'Probability of successful Support, exact only is
%d%%\n', successExactProb(n,1));
        fprintf (1, 'Probability of successful Support with False Positive is
%d%%\n', successFalseDetectProb(n,1));
        fprintf (1, 'Average Run Time for CS Algorithm is %f ms\n',
CSAveRunTime(n,1)*1e3);
    end


    % write to file (each iteration of var sweep
    config_opt = 'write_var';
    eval(config_name);
    config_opt = 'none';

end


end %% SNR count

%% 6) Process Results from Each Iteration of User Defined Variable Sweep
if ((showPrint || showAll) && ~showQuiet)
    fprintf('-----------------------------------------------------------------
-----------------------------\n');
    fprintf('6) Processing Results from User Defined Variable iteration
...\n');
end
% fprintf(fid, '\r\n');


end % variable count

%% 7) Process Results from User Defined Variable Sweep
if ((showPrint || showAll) && ~showQuiet)
    fprintf('-----------------------------------------------------------------
-----------------------------\n');
    fprintf('7) Processing All Results from User Defined Variable iterations
...\n');
end


%% 8) Conclude Simulation
if ((showPrint || showAll) && ~showQuiet)
```

```
    fprintf('-------------------------------------------------------------------
-----------------------------\n');
    fprintf('8) Wrapping up simulation ...\n');
end

fclose(fid);
fclose(fid_trial);
fprintf(1, 'SIMULATION COMPLETE\n');
```

2. Sample configuration file (mixer non-linearity simulation)

```
switch config_opt

    case 'parameters'

        if(~(visOnly))
            showQuiet       = true;
            numTrials       = 100;            % number of trials
            sys_channels    = [30];           % number of system channels
            SNR_vals        = [10];


            % combine noise and NL into one sim
            var_sweep       = [10 9 8 7 6 5 4 3 2 1]; % user defined variable
        end

        % additional settings
        noise_type = 'analog';   % {'analog', 'digital', 'none'}
        noise_type_cc = 'analog'; % {'analog', 'digital', 'none'}
        code_type = 'rand';                % { 'rand' }
        useMatchingPi = true;  % decides whether pi sequences for control
        knownSupport = false;  % if true, RecSuppSorted=Sorig
        knownSupport_cc = false; % if true, RecSuppSorted=Sorig
        % Reconstruction
        reconstruction_choice = 1;
        reconstruction_choice_cc = 1;
        % 1 : RunOMP_Unnormalized(v, A, 2*N, 0, 0.01, false);
        % 2 : RunOMP_Unnormalized(v, A, N, 0, 0.01, true);
        % 3 : RunOMP_forMB(v, A, N, 100);
        % 4 : l1qc_logbarrier                        %not working
        % 5 : l1_cvx                                  %not working
        % 6 : RunCoSaMP(v, A, N, MaxIter, 0.01, 0.01)


        % random seed
        useUserSeed = 0;        % true: use seed below, false: Matlab
        % 0 : Matlab will initialize random seed
        % 1 : Set UserSeed to a random int, used to reproduce random case
        % 2 : Set UserSeed to a user defined value to reproduce case
        switch useUserSeed
        case 0
            UserSeed = NaN;
        case 1
            % generate random integer
            UserSeed = ceil(2^16 * rand());
        case 2
             % use explicit seed
            UserSeed = 41548;
            UserSeed = 24120;

        otherwise
            UserSeed = NaN;
            fprintf('ERROR: invalid input for useUserSeed, letting Matlab
seed...\n');
```

```
        end

        if ((showPrint || showAll) && ~showQuiet)
            fprintf(1,'-------------------------------------------------------
-------------------------------------\n');
            fprintf(1,'1) Testing parameters\n');
            fprintf(1,'   num trials=%d, random seed=%d\n', numTrials,
UserSeed );
            fprintf(1,'   noise type=''%s'', mixing seq=''%s'',
reconstruction choice=%d \n',noise_type, code_type,reconstruction_choice);
            fprintf(1,'   m=[ ');
            fprintf(1,'%g ',sys_channels(1,:));
            fprintf(1,']\n');
            fprintf(1,'   SNRin=[ ');
            fprintf(1,'%g ',SNR_vals(1,:));
            fprintf(1,']\n');
            fprintf(1,'   User Defined Variable=[ ');
            fprintf(1,'%g ',var_sweep(1,:));
            fprintf(1,']\n');
        end

        %% 2) Initialize User Input
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % choose input signal type to be created by GenerateSignal function
        input_sig_type = 'sinc_sin';            % { 'sinc_sin', 'qpsk' }
        % included signal types:
        %  sinc_sin : same as included with test bench
        %  qam16    : signal from 'Robust Xampling Rx' (Abhishek)

        % Signal Model and Sampling Parameters
        N = 6;  % Number of bands
        B = 50e6;                               % Maximal width of each band
        fnyq = 10e9;                            % Nyquist rate (2 x fmax)
        Tnyq = 1/fnyq;                          % period of Nyquist Frequency
        volt_fs = 1;                            % full scale voltage (+/- 0.5V)

        % System Details
        mChan = sys_channels;                   % number of channels to test
        r = 100;                                % Number of iid drawn pi(t);

        %%% some confusion in definition
        L = 195;                                % length of unknown vector, z(f)
        M = L;                                  % (number of alt in one period)
        L0 = floor(M/2);
        L = 2*L0+1;                             % length of unknown vector, z(f)


        fp = fnyq/L;                            % frequency of pi sequence
        Tp =1/fp;                               % period of pi sequence
        fs = fnyq/L;                            % Sampling rate at each channel,
        Ts = 1/fs;                              % sampling period

        % More signal details
        Ei = [1 2 3];                           % Energy of the i'th band
        Bi = ones(1,N/2)*B;                     % Generate signal bandwidths
        R = 1;                                  % R sets time resolution
```

```
        K = 91;                                 % (K+K0)= set cutoff ideal filter
        K0 = 10;                                % R*K0*L is res for padding zeros

        % time length of simulation
        TimeResolution = Tnyq/R;                % temporal resolution time axis
        TimeWin = [0  L*R*K-1 L*R*(K+K0)-1]*TimeResolution; % Time interval
in which signal is observed

        Taui = [0.4 0.7 0.2]*max(TimeWin);    % Time offest of the i'th band

        t_axis = TimeWin(1)   : TimeResolution : TimeWin(end);      % Time axis
        t_axis_sig   = TimeWin(1)   : TimeResolution : TimeWin(2);


        if ((showPrint || showAll) && ~showQuiet)
            fprintf(1,'-------------------------------------------------------
---------------------------------------\n');
            fprintf(1,'2) Initialize User Input ...\n');
            fprintf(1,'Signal model\n');
            fprintf(1,'   N= %d, B=%3.2f MHz, fnyq = %3.2f GHz\n', N, B/1e6,
fnyq/1e9);
            fprintf(1,'\n');
            fprintf(1,'Sampling parameters\n');
            fprintf(1,'   fp = %3.2f MHz, fs = %3.2f MHz, m=[
',fp/1e6,fs/1e6);
            fprintf(1,'%g ',mChan(1,:));
            fprintf(1,'], Tp=%3.2f ns, Ts=%3.2f ns\n',Tp/1e-9, Ts/1e-9);
            fprintf(1,'   M=%d, L0 = %d, L=%d \n', M, L0, L );
            fprintf(1,'\n');
            fprintf(1,'Simulation settings\n');
            fprintf(1,'   Time  window = [%3.2f , %3.2f)
uSec\n',TimeWin(1)/1e-6, TimeWin(2)/1e-6 );
            fprintf(1,'   Time resolution = %3.2f nSec, grid length = %d\n',
TimeResolution/1e-9, length(t_axis));
            fprintf(1,'   Num Sampling Periods = %10.2f, Num Mixing Periods =
%10.2f\n',floor((TimeWin(2))/Ts),floor(TimeWin(2)/Tp));
        end


        if((showParameterCheck || showAll) && ~showQuiet)
           fprintf(1,'\n');
           fprintf(1,'Checking System Parameters ...\n');
           checkParameters(B, fp, fs, fnyq, mChan(1), N, M, L0, L)
        end


        %% 3) Generate system models

        if((showPrint || showAll) && ~showQuiet)
           fprintf(1,'-------------------------------------------------------
---------------------------------------\n');
           fprintf(1,'3) Generate System Models ...\n');
        end

        % low-pass filter model
```

```
        % {'ideal', 'elliptic', 'butterworth'}
        filter_type = 'ideal';
        % filter cutoff normalized to Fs corresponding the digital data
        filter_e_cutoff = fs/(1/TimeResolution);
        % filter cutoff normalized to Fs corresponding the digital data
        filter_b_cutoff = fs/(1/TimeResolution);
        % percent gain mismatch (chosen at random, [0,x.x], worst case)
        filter_gain_mm_max = 0.0;
        filter_nl_type = 'none';
        filter_noise_dbm = NaN;
        filter_full_scale = volt_fs;
        m_lpf = model_low_pass_filter(filter_type, filter_e_cutoff,
filter_b_cutoff, filter_full_scale, filter_gain_mm_max, filter_noise_dbm,
filter_nl_type);

        % mixer
        % percent delay in mixing sequence (IMPORTANT: only works for R > 1)
        mixer_delay_max = 0.0;
        mixer_nonlinear_type = 'none';
        mixer_full_scale = volt_fs;
        mixer_noise_dbm = NaN;
        m_mixer = model_mixer(mixer_full_scale, mixer_delay_max,
mixer_noise_dbm, mixer_nonlinear_type);

        % adc
        adc_quant_bits = inf;                       % res (inf = inf res)
        adc_full_scale = volt_fs;
        adc_nonlinear_type = 'none';
        adc_noise_dbm = NaN;
        m_adc = model_adc(adc_quant_bits, adc_full_scale, adc_noise_dbm,
adc_nonlinear_type);

        % generate Control Case (cc) models
        m_lpf_cc = model_low_pass_filter('ideal', fs/(1/TimeResolution),
fs/(1/TimeResolution), volt_fs, 0, NaN, 'none');
        m_mixer_cc = model_mixer(volt_fs, 0, NaN,'none');
        m_adc_cc = model_adc(inf, volt_fs, NaN, 'none');

        if((showModels || showAll) && ~showQuiet)
           fprintf('Test Models:\n\n');
           m_adc.printInfo();
           m_lpf.printInfo();
           m_mixer.printInfo();

           fprintf('\nControl Case (cc) Models:\n\n');
           m_adc_cc.printInfo();
           m_lpf_cc.printInfo();
           m_mixer_cc.printInfo();
        end



        %% 3.1) Write to file just prior to simulation start and save
workspace

        save './results/trial_workspace'
```

```
        fprintf(fid, '%s noise; sweep mixer noise and delay\r\n',
noise_type);
        fprintf(fid, ' %g', var_sweep);
        fprintf(fid, '\r\n Var(x); NUM TRIALS; noise type; #chan ; SNRin;
NLtype;  %% Success; %% Success Exact; %% Success False Detect; AVE RUNTIME
[s]\r\n');

    case 'set_var'
        m_mixer.setNoiseDBm(NaN);
        m_mixer.setNLType('rapp');
        m_mixer.setNLRSmooth(var_sweep(countVar));

    case 'write_var'

            % write to file
        fprintf(fid, '%d; %d; %s; %d; %f; %s; %f; %f; %f; %f\r\n',  countVar,
numTrials, noise_type, mChan(n) , SNR, m_mixer.nl.type, successProb(n,1),
successExactProb(n,1), successFalseDetectProb(n,1),CSAveRunTime(n,1));

    otherwise
        fprintf(1, 'ERROR: CONFIGURATION OPTION NOT RECOGNIZED!!!!\n');

end
```

References

[1] B. Murmann, "ADC Performance Survey 1997-2011," 2011. [Online]. Available: http://www.stanford.edu/~murmann/adcsurvey.html. [Accessed May 2011].

[2] M. Mishali and Y. C. Eldar, "From Theory to Practice: Sub-Nyquist Sampling of Sparse Wideband Analog Signals," *IEEE Journal of Selected Topics in Signal Processing,* vol. 4, no. 2, pp. 375-391, 2010.

[3] S. Kirolos, J. Laska, M. Wakin, M. Duarte, D. Baron, T. Ragheb, Y. Massoud and R. Baraniuk, "Analog-to-Information Conversion via Random Demodulation," in *Dallas/CAS Workshop on Design, Applications, Integration and Software (IEEE)*, Dallas, 2006.

[4] M. Mishali, Y. C. Eldar and A. Elron, "Xampling-Part I: Practice," *Submitted to IEEE for publication, Technion-Israel Institute of Technology,* pp. 1-17, 2009.

[5] M. Mishali, Y. C. Eldar, O. Dounaevsky and E. Shoshan, "Xampling: Analog to Digital at Sub-Nyquist Rates (CCIT Report No. 751, EE Pub No. 1708)," Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel, 2009.

[6] C. Choudhuri, A. Ghosh, U. Mitra and S. Pamarti, "Robustness of Xampling-based RF Receivers against Analog Mismatches," in *International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan, 2012.

[7] W. C. Black and D. A. Hodges, "Time interleaved converter arrays," *IEEE Journal Solid-State Circuits,* Vols. SC-15, no. 6, pp. 1022-1029, 1980.

[8] E. J. Candès, J. Romberg and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory,* vol. 52, no. 2, pp. 489-509, 2006.

[9] D. Donoho, "Compressed Sensing," *IEEE Transactions on Information Theory,* vol. 52, no. 4, pp. 1289-1306, 2006.

[10] E. Candès and M. B. Wakin, "An Introduction to Compressive Sampling," *IEEE Signal Processing Magazine,* pp. 21-30, March 2008.

[11] E. J. Candès and J. Romberg, "Sparsity and Incoherence in Compressive Sampling," *Inverse Probabilit,* vol. 51, no. 12, pp. 969-985, 2007.

[12] E. Candès and T. Tao, "Decoding by Linear Programming," *IEEE Transactions on Information Theory,* vol. 51, no. 12, pp. 4203-4215, 2005.

[13] E. Candès, "The Restricted Isometry Property and Is Implications for Compressed Sensing," *Academie des sciences (submitted),* 2008.

[14] M. Mishali and Y. C. Eldar, "Expected RIP: Conditioning of the Modulated Wideband Converter," Department of Electrical Engineering, Technion — Israel Institute of Technology, Haifa, Israel, 2009.

[15] R. G. Baraniuk, "Compressive Sensing," *IEEE Signal Processing Magazine,* pp. 118-124, July 2007.

[16] J. A. Tropp and A. C. Gilbert, "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit," *IEEE Transactions on Information Theory,* vol. 53, no. 12, pp. 4655-4666, 2007.

[17] D. Donoho and M. Elad, "Optimally Sparse Representation in General (nonorthogonal) dictionaries via 'l1 minimization'," *Proceedings of National Academy of Sciences of the USA,* vol. 100, pp. 2197-2202, 2003.

[18] S. Chen, D. Donoho and M. Saunders, "Atomic Decomposition by Basis Pursuit," *Society for Industrial and Applied Mathematics,* vol. 43, no. 1, pp. 129-159, 2001.

[19] J. Tropp, "Just Relax: Convex Programming Methods for Identifying Sparse Signals in Noise," *IEEE Transactions on Information Theory,* vol. 52, no. 3, pp. 1030-1048, 2006.

[20] E. Candès and Y. Plan, "Near-ideal model selection by l1 minimization," *available online [http://www-stat.stanford.edu/~candes/papers/LassoPredict.pdf],* 2007.

[21] R. Baraniuk, M. Davenport, R. Devore and M. B. Wakin, "A simple prof of the restricted isometry principle for random matrices (aka the Johnson-Lindenstrauss lemma meets compressed sensing," *Constructive Approximation,* p. available online [http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/JL_RIP.pdf], 2006.

[22] R. Calderbank, S. Howard and S. Jafapour, "Deterministic Compressive Sensing with Groups of Random Variables," *available online [http://dsp.rice.edu/sites/dsp.rice.edu/files/cs/strip-more.pdf] ,* 2009.

[23] D. Needell and J. A. Tropp, "CoSaMP: Iterative Signal Recovery From Incomplete and Inaccurate Samples," *available online [http://arxiv.org/abs/0803.2392v2],* 2008.

[24] D. Donoho, Y. Tsaig, I. Drori and J.-l. Starck, "Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit," *available online [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.5221],* 2006.

[25] M. B. W. Emmanuel Candès, "An Introduction to Compressive Sampling," *IEEE Signal Processing Magazine,* pp. 21-30, March 2008.

[26] T. T. Emannuel Candès, "Decoding by Linear Programming," *IEEE Transactions on Information Theory,* vol. 51, no. 12, pp. 4203-4215, 2005.

[27] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation,," *IEEE Transactions on Information Theory,* vol. 50, no. 10, pp. 2231-2242, 2004.

[28] D. Needell and R. Vershynin, "Uniform uncertainty principles and signal recovery via Regularized Orthogonal Matching Pursuit (StOMP)," *available online [http://www-personal.umich.edu/~romanv/papers/ROMP.pdf],* 2007.

[29] S. Kunis and H. Rauhut, "Random Sampling of Sparse Trigonometric Polynomials II - Orthogonal Matching Pursuit versus Basis Pursuit," *available online [http://rauhut.ins.uni-*

bonn.de/KunisRauhutOMP.pdf], 2006.

[30] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 1.12," available online [http://cvxr.com/cvx/], 2011.

[31] A. Weinstein, "Beginners Code for Compressive Sensing," available online [http://control.mines.edu/mediawiki/upload/f/f4/Beginners_code.pdf], 2009.

[32] 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.

[33] M. Mishali, Y. C. Eldar, O. Dounaevsky and E. Shoshan, "Xampling: analog to digital at sub-Nyquist rates," *IET Circuits, Devices & Systems,* vol. 5, no. 1, pp. 8-20, 2010.

[34] Landau, H. J.; Bell Telephone Laboratories, Incorporated, "Necessary Density Conditions for Sampling and Interpolation of Certain Entire Functions," *Acta Mathematica,* vol. 117, no. 1, pp. 37-52, 1967.

[35] M. Mishali and Y. Chen, "Yonina Eldar Homepage (Demo_MWC.zip)," 2010. [Online]. Available: http://webee.technion.ac.il/people/YoninaEldar/software_det2.html. [Accessed 2011].

[36] Y. Chen, M. Mishali, Y. Eldar and A. O. H. III, "Modualted Wideband Converter with Non-Ideal Lowpass Filters," *available online [http://web.eecs.umich.edu/~hero/Preprints/ChenICASSP10.pdf].*

[37] Matlab, "Amplifier - Complex baseband model of amplifier with noise," [Online]. Available: http://www.mathworks.com/help/toolbox/rfblks/amplifier.html. [Accessed April 2012].

[38] E. Candès and J. Romberg, "L1-MAGIC," available online [http://users.ece.gatech.edu/~justin/l1magic/], 2005.