BPSK WAVEFORM DEMODULATION USING AN ALL ANALOG GAUSSIAN WAVELET TRANSFORM

by

Daniel Eddowes

A thesis submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science.

Baltimore, Maryland

May, 2020

Abstract

A binary phase shift keying (BPSK) waveform is a ubiquitous digital communications modulation technique. Presented is a novel method of demodulating a BPSK waveform back into its constitute bits by way of an analog Gaussian wavelet transform. A comprehensive design method for an all-analog wavelet transform is presented. This includes converting the wavelet transform into a state-space form by way of a singular value decomposition of the daughter wavelets, and the design of an analog circuit implementing the wavelet transform. The Gaussian wavelet is used for its ability to capture fast transients in an input signal, which is the key factor which allows for the wavelet transform to demodulate a BPSK waveform. The validity of an all-analog transform is presented through simulations and laboratory measurements. Because of the nature of the wavelet transform this method of demodulation has great noise resilience which is demonstrated both with simulated and laboratory measured bit error rates.

Thesis Committee

Primary Readers

Jeff Houser (Primary Advisor)

Johns Hopkins University Whiting School of Engineering

Engineering for Professionals

Amir Najmi

Johns Hopkins University Whiting School of Engineering

Engineering for Professionals

Brian Jennison

Johns Hopkins University Whiting School of Engineering

Engineering for Professionals

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Core Focus

This thesis explores a novel method to demodulate binary phase shift keying (BPSK) waveforms using an analog wavelet transform. The main focus of the thesis is the design and implementation of the all analog Gaussian wavelet transform. The wavelet transform is implemented with custom 5th order active-RC filters, wherein each filter's impulse response approximates a different Gaussian daughter wavelet. The wavelet transform is able to identify the phase transitions in the BPSK waveform, which are used to demodulate the modulated data.

Wavelets are often used in the digital domain for data compression, transient signal analysis, and noise reduction [1,2,3]. Analog wavelet transforms are often used in medical devices, as they offer a low-power, computationally light method of computing the wavelet transform [4,5]. The results presented in this thesis represent not only a new application for the analog wavelet transform, but a novel method of BPSK waveform demodulation.

This chapter will present a high-level overview of the target application for the analog wavelet transform as well as a full system diagram. Chapter 2 presents some theory on the wavelet transform while chapter 3 dives into mathematical approximation techniques for the Gaussian wavelet. These approximations lay the ground work for the analog implementation of the transform, which is presented in chapter 4. Finally, full system performance and laboratory measurements of the wavelet transform and demodulation technique are presented in chapter 5. The thesis is concluded and future work is discussed in chapter 6.

1.2 System Overview

For reasons that will be described in detail during the wavelet theory discussion in chapter 2, the analog wavelet transform is comprised of an analog filter bank, where each filter in the bank represents a different scale of the Gaussian wavelet. The BPSK waveform, sourced from an arbitrary waveform generator, supplies the input to the wavelet filter bank. The output of the filters are first summed, then provided gain and a direct current (DC) offset voltage. The output is then passed into a microcontroller unit (MCU) where the final demodulation steps occur. A computer running MATLAB closes the loop between input and output, both commanding the arbitrary waveform generator and receiving the demodulated bits from an MCU, thus enabling bit error rate (BER) measurements. See figure 1.1 below for a detailed block diagram of the entire system.

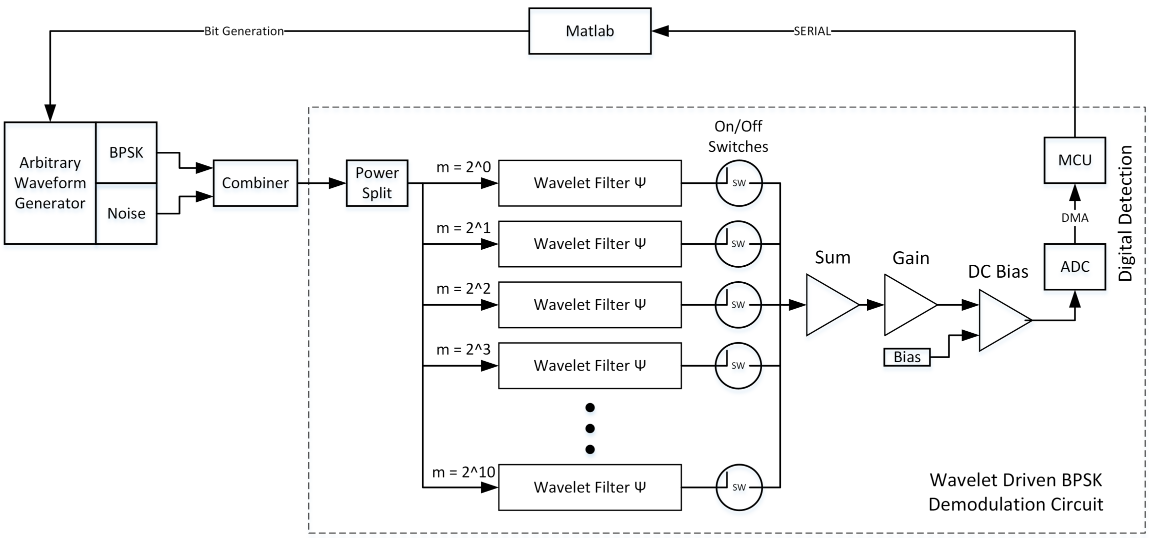


Figure 1.1: System Block Diagram

1.3 BPSK Waveform

An unencoded BPSK waveform is a phase modulated sinusoid, with phase transitions synchronized with bit transitions in a data stream. BPSK was selected as a data modulation type because of its mathematical simplicity and ease of test verification. BPSK modulation is commonplace in wireless communication, often found in deep space and near-earth satellite communications as well as the 802.11 WiFi standards [6].

An example of a BPSK waveform and associated data stream is shown in figure 1.2.

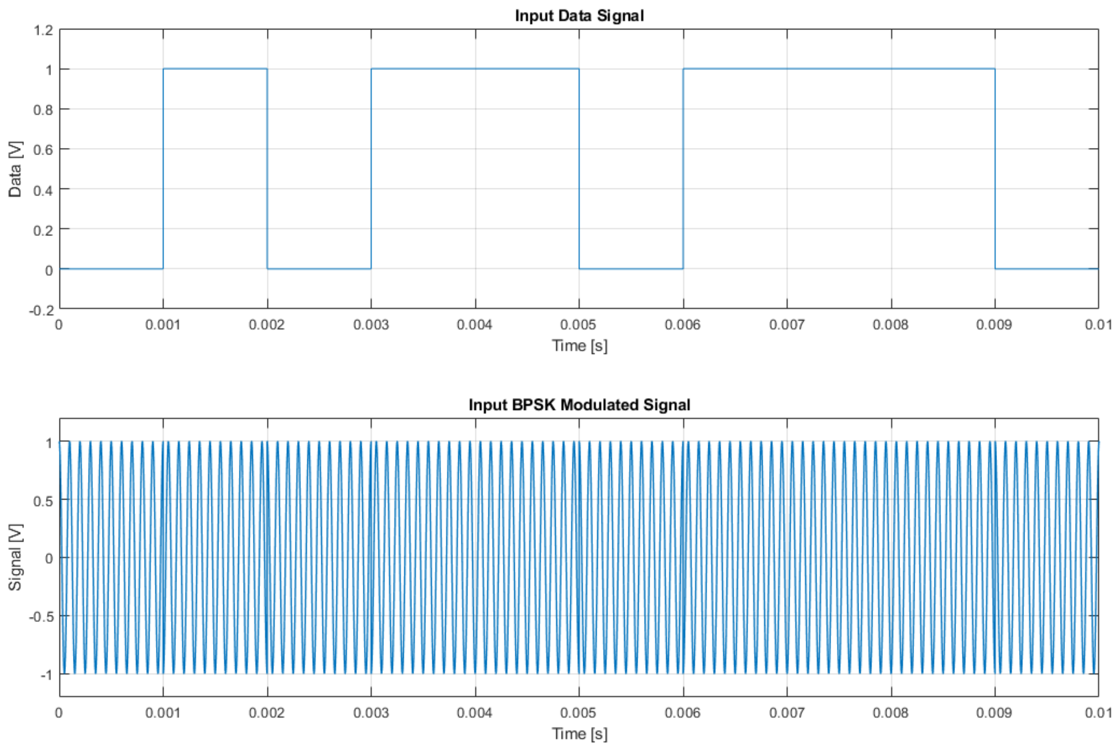


Figure 1.2: Ideal BPSK waveform. Data shown on top and its corresponding BPSK modulation on bottom

The carrier frequency in figure 1.2 is 10 kHz, while the data rate is 1 kbps. Figure 1.3 looks closer at a few of the bit transitions, where the phase transitions in the BPSK waveform become visible.

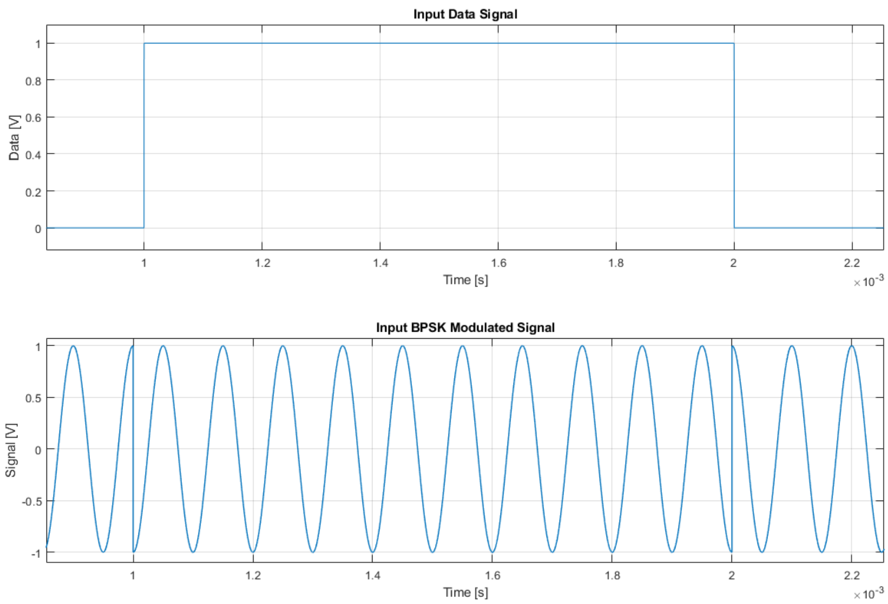


Figure 1.3: BPSK phase transitions coincide with data transitions

Data is first coded as bipolar non-return-to-zero level (NRZ-L) encoding. The bits are then aligned with the minimum and maximum amplitude points on the BPSK carrier, so bit transitions result in both 180 degree phase shifts and peak-to-peak amplitude changes. This is slightly different from typical BPSK waveforms in which bit transitions are aligned with the zero crossings of the carrier frequency, resulting in only phase transitions. In theory, the wavelet transform circuitry constructed in this thesis can function with such a BPSK waveform, however, bit transitions at peak amplitude allow for an easier demodulation scheme in practice. In short, this is because the wavelet transform is able to pick out fast transients in time (amplitude) and frequency (phase), providing better demodulation ability than the case of zero-crossing phase transitions wherein only a fast frequency transient is present.

Regardless of when bit transitions occur, there must be consistency to ensure the wavelet transform output is predictable. Thus, the carrier frequency and the bit rate must be harmonically related. Equation 1.1 describes the BPSK waveform:

$$s(t) = Acos(2\pi f_c t + \pi(1-n)) \tag{1.1}$$

with carrier frequency f_c, time t, bit n, and amplitude A. In equation 1.1, zero crossing transitions can be accomplished by replacing $\cos(\bullet)$ with $\sin(\bullet)$, or by replacing π with $\pi/2$. The target bit rates in this thesis will be less that 100 bps, and the target BPSK carrier frequency will be less than 10 kHz. In practice, 20 bps bit rates and 1 kHz carrier frequencies are measured.

A few papers have used wavelets to demodulate BPSK and more complex waveforms in the past [7,8]. Their results have laid the groundwork to show that wavelet demodulation is both possible and practical. However, this is the first known instance that the transform and demodulation will take place in the analog domain.

Chapter 2

Wavelets

2.1 Wavelet Properties

The wavelet transform is a mathematical tool that captures both time and frequency domain information of an input signal. It is often used in data compression, noise reduction, or as in the case of this thesis, as a tool to capture fast frequency transients in an input BPSK waveform in the presence of noise.

The wavelet transform is often compared to the well-known Fourier transform, which allows an input time-domain signal to be represented solely in the frequency domain:

$$\hat{F}(f) = \int_{-\infty}^{+\infty} f(t)e^{-2\pi ift}dt \tag{2.1}$$

The wavelet transform operates by correlating the input signal with a dilated and time shifted mother wavelet that is localized in time and not a sinusoid as is the case with the Fourier transform, allowing for the preservation of time-domain information in the transform output [9]. Given a time span Δt and angular frequency span $\Delta\omega$, by the uncertainty principle:

$$\Delta t \Delta\omega \geq 2\pi \tag{2.2}$$

With greater time-domain resolution, frequency resolution suffers, and vice versa. The quality of the time and frequency resolution is dependent on the projection basis, or mother wavelet, selected for the transformation.

The wavelet transform, $W_\psi(a,b)$, is described by the following equation:

$$W_\psi(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} \psi^* \left(\frac{t-b}{a} \right) f(t) dt \tag{2.3}$$

Here a is the dilation parameter which represents the wavelet scale, b is translation parameter, and $f(t)$ is any $\mathbb{L}_2$ input signal. A $\mathbb{L}_2$ function is square integrable, thus the following must hold true:

$$\int_{-\infty}^{\infty} |f(t)|^2 dt < \infty \tag{2.4}$$

A BPSK waveform is symmetric about the x-axis and has finite duration, it is therefore an $\mathbb{L}_2$ function. The $\frac{1}{\sqrt{a}}$ factor in equation 2.3 normalizes the energy across each scale. There is a similar equation to equation 2.3 for the wavelet transform in the discrete time domain, but it is ignored here as all operations in this thesis occur in continuous time.

Strictly when talking about wavelets, the wavelet scale is analogous to frequency. The scale corresponds to a contraction or dilation of the mother wavelet. The larger the wavelet scale, the shorter the mother wavelet's impulse response is in the time domain, allowing the wavelet to represent higher frequency content in an input signal.

There are many mother wavelets available to choose from, all of which have a few key properties:

$$\begin{aligned}&\int_{-\infty}^{\infty} \psi(t)dt = 0\\&\int_{-\infty}^{\infty} \frac{|\hat{\Psi}(\omega)|^2}{|\omega|}d\omega = C_{\hat{\Psi}} < \infty\end{aligned} \tag{2.5}$$

The first property above states that the mother wavelet must integrate to zero. The second property is the admissibility condition, which is a necessary condition to ensure the wavelet transform inverse exists. In the admissibility condition, $\hat{\Psi}(\omega)$ is the Fourier transform of the mother wavelet $\psi(t)$.

The mother wavelet function chosen for this thesis is the first order Gaussian wavelet. This mother wavelet performs well at capturing fast transients in the input signal. The first order Gaussian wavelet can be expressed as the first derivative, and hence first order, of the Gaussian function.

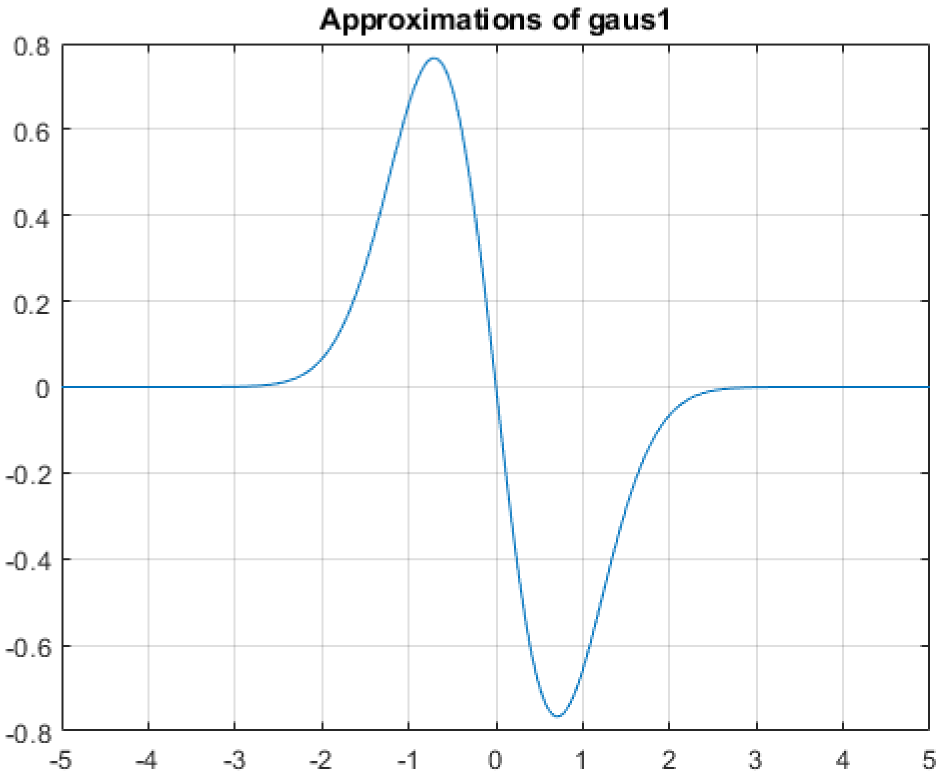


Figure 2.1: Ideal first order Gaussian wavelet, generated with MATLAB's *wavefun* function

$$f(t') = Ce^{-t'^2} \tag{2.6}$$

The mother wavelet, shown graphically in figure 2.1, is expressed with the following closed-form equation:

$$\frac{d}{dt'}f(t') = \psi = -2Ct'e^{-t'^2} \tag{2.7}$$

where $C = 1$. The Gaussian daughter wavelets are found by substituting $t' =$

$at - \tau$, where a is the dilation parameter and t is the translation parameter:

$$\psi(t, a) = -2(at - \tau)e^{(at-\tau)^2} \tag{2.8}$$

Normally a wavelet is centered around zero, but τ is a time offset used to shift all contents of the wavelet to $t > 0$. This time offset is needed to build causal analog filters. Discussion on selecting the value for τ is found in chapter 3.

2.2 Wavelet Theory

This section will dive into some deeper theory behind the wavelet transform. While not critical for the engineering work done in the remainder of the thesis, it provides a fundamental backdrop to how the wavelet transform operates.

2.2.1 Multi-Resolution Analysis

The first concept of interest is multi-resolution analysis (MRA). MRA reduces the problem of computing wavelet coefficients on a dyadic grid to a series of orthogonal projections which can be implemented using finite impulse response (FIR) filters.

A sequence of closed subspaces $V_n, n \in \mathbb{Z}$ in $\mathbb{L}_2$ have a hierarchy:

$$\ldots V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \ldots \tag{2.9}$$

where the only intersection of the nested subspaces is the zero function, and the union of the subspaces is dense in $\mathbb{L}_2$. The subspace hierarchy is constructed so the following two properties exists. First, the subspaces are self-similar such that:

$$f(2^j t) \in V_j \iff f(t) \in V_0 \tag{2.10}$$

This is an extension of the dyadic grid. Second, a scaling function, $\phi(t)$, exists that is an orthogonal basis for the subspace V_0. For a function $f(t) \in \mathbb{L}_2$:

$$f(t) = \sum_k c_j \phi(x - k) \tag{2.11}$$

V_0 contains the set of all functions $f(t) \in \mathbb{L}_2$ such that $f(t)$ can be written as a linear combination of the scaling function as is done in equation 2.11. Since $V_0 \subset V_{-1}$, $\phi(t)$ can be written as a linear combination of the scaling function in V_{-1}.

$$\phi(t) = \sqrt{2} \sum_{k \in \mathbb{Z}} h_k \phi(2t - k) \tag{2.12}$$

where h_k are the coefficients for an FIR filter. When a sequence of subspaces satisfies the above properties, an orthonomal, though possibly non-unique, basis exists with the following properties:

$$\psi_{jk}(t) = 2^{j/2}\psi(2^j t - k) \quad j, k \in \mathbb{Z} \tag{2.13}$$

Here $\psi_{jk}(t)$ spans the subspace $V_j^{\perp}$, whch is the orthogonal complement of V_j in V_{j-1}. Because of the nature of the subspace hierarchy, $V_{j-1} = V_j \oplus V_j^{\perp}$. In other words, the next highest subspace, V_{j-1}, is the direct sum of the next lowest subspaces in the hierarchy spanned by $\phi(t)$ and $\psi_{jk}(t)$. The ψ_{00} function is defined as the mother wavelet. Details of the derivation of $\psi_{jk}(t)$ from $\phi(t)$ are found in the literature [10,11]. The mother wavelet can be expressed as:

$$\psi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} g_k \phi(2t - j) \tag{2.14}$$

where $g_k = (-1)^k h_{1-k}$, g and h are quadrature mirror filters. The above mathematics can be distilled into the following explanation. A square integrable function $f(t)$ can be obtained by projecting it onto a subspace V_{j-1} in a multi-resolution analysis space. This results in two terms, a projection due to the scaling function $\phi(t)$ on the next coarser scale V_j, and the error missed when going from V_{j-1} to V_j. This error, in subspace $V_j^{\perp}$, is due to the wavelet function $\psi(t)$.

Every subspace V_j is a direct sum of all detail in lower scale subspaces:

$$V_{j-1} = \oplus_{m=j}^{\infty} V_m^{\perp} \tag{2.15}$$

Taking the limit as $j \to -\infty$ results in the wavelet approximation

$$f(t) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} d_{jk} \psi_{jk}(t) \tag{2.16}$$

where d_{jk} are the wavelet transform coefficients [10]. This overview of MRA provides a background to the linear algebra structure of wavelets and the wavelet transform.

2.2.2 Vanishing Moments

Attention is now turned to zero moments of the mother wavelet. For a given function $f(t)$, the kth zero moment is defined by:

$$\langle t^k \rangle_f = \int_{-\infty}^{\infty} t^k f(t) dt \tag{2.17}$$

The zeroth moments of the mother wavelet must equal zero due to the admissibility condition, there is not a guarantee for the scaling function ϕ. For a wavelet with $K \geq 1$ vanishing moments the zero moments of the wavelet can be rewritten as:

$$\langle t^k \rangle_\psi = 0, 1 \le k \le K \tag{2.18}$$

The number of vanishing moments of a mother wavelet $\psi(t)$ is also equal to the number of zero derivatives of the Fourier transform of the mother wavelet, $\hat{\Psi}(\omega)$, at $\omega = 0$. Zero moments of a mother wavelet result in the following theorem.

A mother wavelet $\psi(t)$ with $K \ge 1$ vanishing moments when applied to a polynomial function of degree $\le K$ will produce wavelet coefficients d_{mn} that are identically zero for all $m, n \in \mathbb{Z}$.

In theory, the polynomial mapping to a sinusoidal of any reasonable time length would be greater than the order of any practical wavelet. This vanishing moment concept acts to demonstrates the wavelet's ability to ignore input polynomials of degree less than K. The first order Gaussian wavelet has one vanishing moment.

2.3 Filter Theory

As equation 2.3 shows, the wavelet transform can be thought of as a cross-correlation between the daughter wavelet and the input function.

$$f \star g = \int_{-\infty}^{\infty} f^*(\tau) g(t+\tau) d\tau \tag{2.19}$$

where $f^*(t)$ is the complex conjugate of $f(t)$. Cross-correlation $f(t) \star g(t)$ is equivalent to the convolution $f^*(-t) * g(t)$. Because the first order Gaussian wavelet is a real valued function, $\psi^*(t) = \psi(t)$, and $f(t) \star g(t) = f(-t) * g(t)$. If the Gaussian wavelet was symmetric about the y-axis, then the cross-correlation and convolution would be equivalent. For the Gaussian wavelet however, with symmetry about the x-axis, $f(t) \star g(t) = -[f(t) * g(t)]$. The general equation for a convolution is written below for two continuous time $\mathbb{L}_2$ functions f and g:

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.20}$$

In filter theory, a filter operates by convolving an input signal with the filter impulse response, or equivalently by multiplying the filter frequency response and Fourier transform of the input signal. For an analog filter:

$$\begin{aligned} v_o(t) &= \int_{-\infty}^{\infty} h(\tau)v_i(t-\tau)d\tau \\ \hat{V}_o(\omega) &= \hat{H}(\omega)\hat{V}_i(\omega) \end{aligned} \tag{2.21}$$

Here v_o is the filter output, v_i is the filter input, $h(t)$ is the filter impulse response, and $\hat{H}(\omega)$ is the filter frequency response. To implement an analog Gaussian wavelet transform, the analog filter needs to have an impulse response that matches the daughter wavelet time domain response. Or put another way, $h(t) = \psi(t)$ and $\hat{H}(\omega) = \hat{\Psi}(\omega)$. Technically, this will procure the negative wavelet transform output, but this is not a problem for BPSK demodulation. Nonetheless, an extra analog inversion at the output of each wavelet is used to create the positive wavelet transform. Creating filters to represent the Gaussian wavelet at multiple scales is the focus of the remainder of this thesis. Ultimately, each scale is implemented by its own filter, creating a filter bank.

A visual example of a wavelet transform occurring with a first order Gaussian mother wavelet and a BPSK waveform is below in figure 2.2.

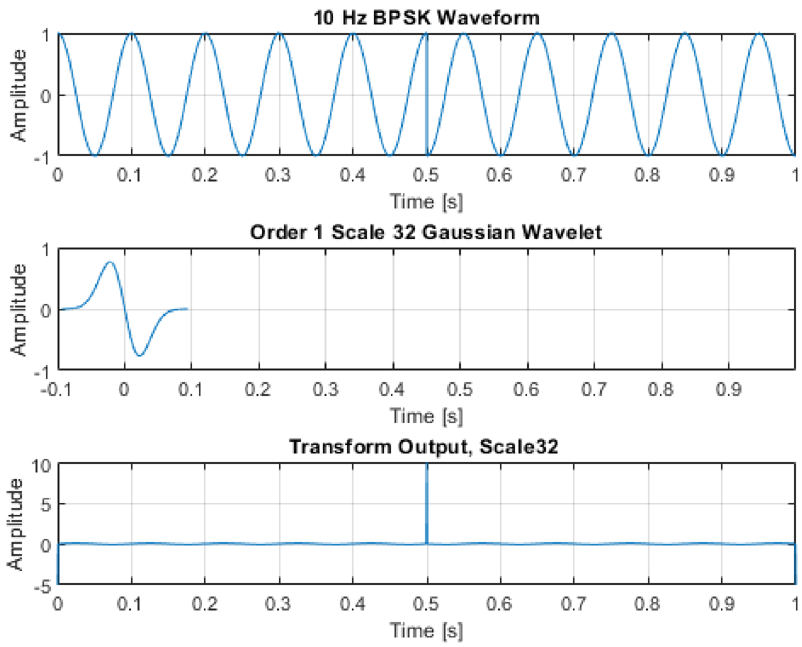


Figure 2.2: BPSK wavelet transform example using a Gaussian wavelet

The wavelet output has a peak at times corresponding to the bit transition in the BPSK waveform, but still shows oscillatory behavior outside the BPSK phase transition. The 32nd scale of the Gaussian wavelet is used in this example as the wavelet period is roughly equal to a full cycle of the BPSK carrier frequency. This simulation is performed using the ideal wavelet transform function in MATLAB, but the same output results when correlating the daughter wavelet with the input BPSK signal.

Chapter 3

Numerical Approximations

The ideal first order Gaussian wavelet is pictured in figure 2.1 with closed form equation 2.8. In order to be implemented in analog hardware, the wavelet must be represented in a form that allows for translation into circuitry. This chapter will explore such forms.

3.1 Wavelet Approximation

The ideal wavelet must be time shifted, as all physical hardware must respect causality. The factor τ in equation 2.8 performs this time shift. Ideally, the physical hardware is also linear and time invariant and choosing this time shift is non-trivial. If the time shift is too long, a high-order system is necessary to insure the system impulse response stays near zero for a period of time. If the time shift is too short, then there will be lost information, as only the wavelet post $t = 0$ can be modeled. A 5th order system will be used to model the wavelets, as it properly models the system with reasonable length time shifts while still being simple enough to design repeatable analog filters. Using a time offset of $t = 1.7$ seems to preserve most of the wavelet information while retaining accuracy in 5th order system. In this case equation 2.8 becomes:

$$\Psi = -2(at - 1.7)e^{-(at-1.7)^2} \tag{3.1}$$

where $a \in \mathbb{Z}$ is the wavelet scale. This is the final closed form equation for the Gaussian daughter wavelets and is the starting point for the wavelet approximations. Two approximation methods, one using the Padé Approximate and the other using the singular value decomposition (SVD) of the Gaussian wavelet are explored and compared. The approximation is then transformed into a state-space representation which allows for easy circuit translation.

3.2 State-Space Representation

A state-space representation of a physical system is a mathematical model that relates a system's input to its output via a first order differential equation and state variables. The state variables evolve over time according to varying input. A state-space representation is a natural way of representing a continuous linear time invariant (LTI) system. An analog filter is generally an LTI system. A state space is represented via equation 3.2

$$\begin{aligned} \dot{x}(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) \end{aligned} \tag{3.2}$$

In equation 3.2, $u(t) \in \mathbb{R}^p$ is the input vector where p is the number of inputs, $y(t) \in \mathbb{R}^q$ is the output vector where q is the number of outputs, and $x(t) \in \mathbb{R}^n$ is the state vector where n is the number of states in the system. **A** is the $n \times n$ state matrix, **B** is the $n \times p$ input matrix, and **C** is the $q \times n$ output matrix. **D** is the feed-through matrix, modeling an instantaneous connection between the input and output, and is not used in this application [12,13]. The state space can also easily be represented in the Laplace domain:

$$\begin{aligned} sx(s) &= \mathbf{A}x(s) + \mathbf{B}u(s) \\ y(s) &= \mathbf{C}x(s) + \mathbf{D}u(s) \end{aligned} \tag{3.3}$$

The analog filter designed to represent the Gaussian wavelet will have one input and output, so $p = q = 1$. It will also be a 5th order filter, which translates into five distinct states, thus $n = 5$, and $x(t)$ is a 5×1 matrix. The state-space representation is easily translated into a circuit as will be shown in section 4.2.

3.2.1 Numerical Approximation Considerations

The goal is to take the continuous Gaussian wavelet and convert it into a state-space representation. To do so, the wavelet must be approximated. The approximation must have a few key properties, namely it should be observable, controllable, and bounded-input bounded-output (BIBO) stable.

An observable system is one in which the internal state vector x can be determined from the system output y. A state-space system is observable if its observability matrix is full rank, i.e. $Rank(\mathbf{W}_o) = dim(x)$. In other words, the rank of the observability matrix must be equal to the number of dimensions in the state vector. The observability matrix is written as follows:

$$\mathbf{W}_o = [\mathbf{C}\ \mathbf{CA}\ \mathbf{CA}^2\ \ldots\ \mathbf{CA}^{n-1}]^T \tag{3.4}$$

where n is the number of dimensions in the state vector. A controllable system means that any bounded-output y can be achieved given a bounded-input u in a finite amount of time. The condition for controllability is similar to the condition for observability in that the controllability matrix needs to be non singular, i.e. full rank. The controllability matrix looks as follows:

$$\mathbf{W}_c = [\mathbf{B}\ \mathbf{BA}\ \mathbf{BA}^2\ \ldots\ \mathbf{BA}^{n-1}] \tag{3.5}$$

Stability is verified by looking at the pole-zero map of the state-space system. For the case of a Padé approximation, where the state space is continuous, there can be no poles in the right-half plane when plotted in the Laplace domain. For the singular value decomposition (SVD) approximation, all poles must be inside the unit circle when plotted in the z-domain.

3.3 Padé Approximate

In this thesis, the Padé approximation is first attempted to produce the wavelet approximation. It takes a time domain vector of the Gaussian wavelet and produces a rational transfer function in the Laplace domain [14]. The approximation procedure functions as follows, starting with the Taylor series of Gaussian wavelet of order $M+N$:

$$T_{M+N}(x) = \sum_{n=0}^{M+N} c_n x^n \tag{3.6}$$

This is set equal to the Padé approximation $P_M^N(x)$

$$P_M^N(x) = \frac{\sum_{n=0}^{N} a_n x^n}{\sum_{n=0}^{M} b_n x^n} \tag{3.7}$$

such that:

$$c_0 + c_1x + c_2x^2 + ... + c_{M+N}x^{M+N} = \frac{a_0 + a_1x + a_2x^2 + ... + a_Nx^N}{b_0 + b_1x + b_2x^2 + ... + b_Mx^M} \tag{3.8}$$

$M > N$ ensurs that P_M^N is a proper rational transfer function. The denominator on the left hand side of equation 3.8 is multiplied on both sides of equation 3.8 and resulting terms with the same exponential order are set equal:

$$\begin{aligned}
a_0 &= c_0 \\
a_1 &= c_1 + c_0b_1 \\
a_2 &= c_2 + c_1b_1 + c_2b_2 \\
&... \\
a_N &= c_N + c_{N-1}b_1 + ... + c_1b_{N-1} + c_0b_N \\
0 &= c_{N+1} + c_Nb_1 + ... + c_1b_N + c_0b_{N+1} \\
&... \\
0 &= c_M + c_{M-1}b_1 + ... + c_1b_{M-1} + c_0b_M \\
0 &= c_{M+1} + c_Mb_1 + ... + c_2b_{M-1} + c_1b_M \\
&... \\
0 &= c_{M+N} + c_{M+N-1}b_1 + ... + c_{N+1}b_{M-1} + c_Nb_M
\end{aligned} \tag{3.9}$$

In equation 3.9 there are $M + N$ equations with $M + N$ unknowns, so they can be solved for all the coefficients in the transfer function creating the Padé approximation of the Taylor series $T_{M+N}(x)$.

A fourth order ($M = 4$, $N = 2$) transfer function is used for this Padé approximation. Higher order systems have convergence issues and lower order systems are bad approximations for the mother wavelet. It is a simple procedure to transform the transfer function into a state space. Starting from a rational transfer function:

$$H(s) = \frac{a_0 s^2 + a_1 s + a_2}{s^4 + b_1 s^3 + b_2 s^2 + b_3 s + b_4} \tag{3.10}$$

The controllable conical state-space representation looks as follows [12]:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -b_4 & -b_3 & -b_2 & -b_1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & a_2 & a_1 & a_0 \end{bmatrix} \tag{3.11}$$

With this state-space representation, the impulse response of the Padé approximation is compared to the ideal Gaussian Wavelet below in figure 3.1.

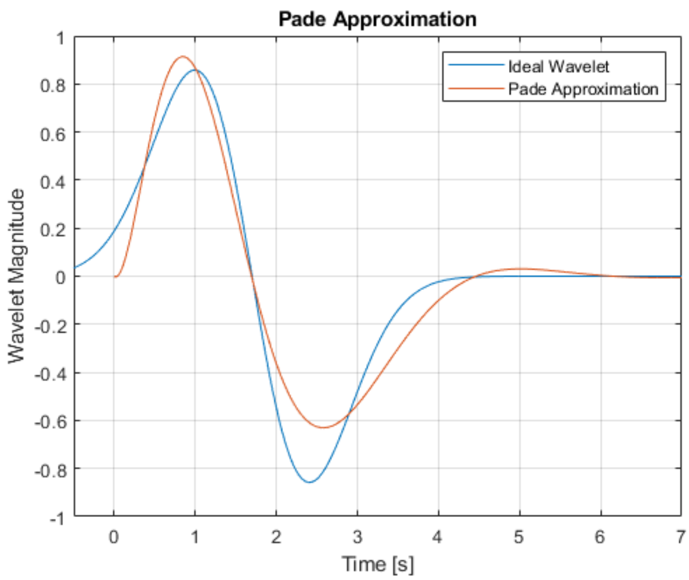


Figure 3.1: Padé approximation impulse response in orange, compared to the ideal first order Gaussian wavelet in blue

This approximation is stable, as shown in figure 3.2, with no poles in the right-half plane:

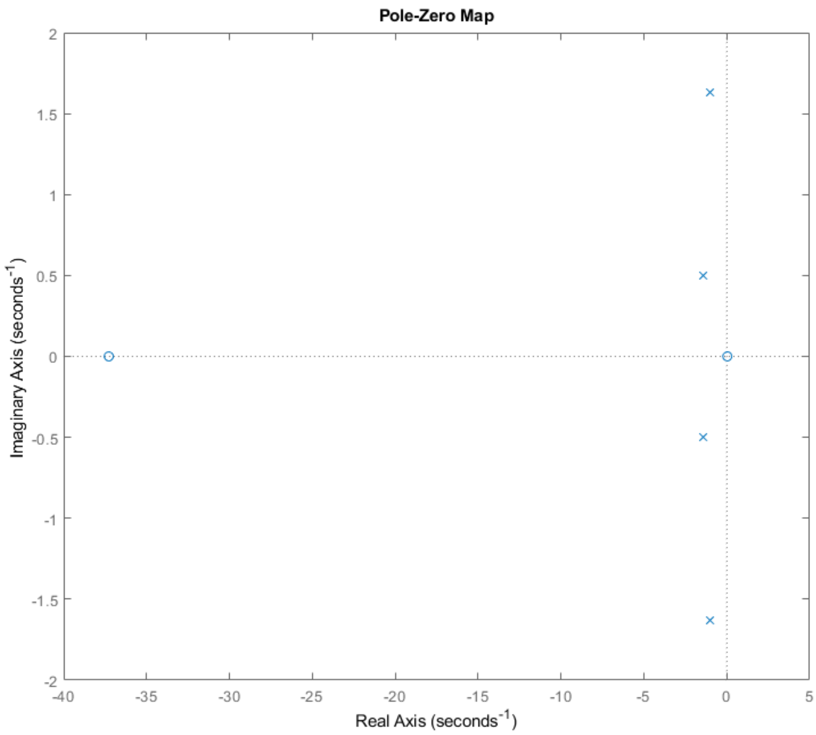


Figure 3.2: Pole-zero map of Padé approximate for the first order Gaussian wavelet

The approximation does not maintain the first wavelet property described in section 2.1, wherein the integral of the wavelet must be zero. In a statistical sense, it does not perform as well as the SVD approximation method described in the next section.

3.4 Singular Value Decomposition

Singular value decomposition is a matrix factorization that generalizes eigen-decomposition for any rectangular matrix. It is often used in finding the best k-dimensional subspace representation for an input n-dimensional matrix, where $k < n$ [15]. The continuous time Gaussian daughter wavelet is sampled into the discrete time domain, which acts as the input vector of interest.

All linear systems have an input-output relation according to a transfer matrix **H**:

$$y = \mathbf{H}u \tag{3.12}$$

where the input vector u is the sampled wavelet vector. Because **H** is LTI, the impulse response of the system will be as follows:

$$h = [\ldots\ 0\ 0\ h_0\ h_1\ h_2\ \ldots]^T \tag{3.13}$$

The system is necessarily zero for all times before $t = 0$. Because equation 3.13 must hold for all times t, the transfer matrix must therefore have a lower triangular structure. The transfer matrix is a Toeplitz matrix with the following form:

$$\mathbf{H} = \begin{bmatrix} \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ \dots & h_0 & 0 & 0 & 0 & 0 & 0 \\ \dots & h_1 & h_0 & 0 & 0 & 0 & 0 \\ \dots & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ \dots & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ \dots & h_4 & h_3 & h_2 & h_1 & h_0 & 0 \\ \dots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.14}$$

This matrix has an infinite dimension, but only past inputs and future outputs need to be considered:

$$\begin{gathered} \bar{y} = \bar{\mathbf{H}}\bar{x} \\ \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & \dots \\ h_2 & h_3 & h_4 & h_5 & \dots \\ h_3 & h_4 & h_5 & h_6 & \dots \\ h_4 & h_5 & h_6 & h_7 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} u_{-1} \\ u_{-2} \\ u_{-3} \\ u_{-4} \\ \vdots \end{bmatrix} \end{gathered} \tag{3.15}$$

here $\bar{\mathbf{H}}$ takes the form of a Hankel matrix. If the feed-through term D in equation 3.2 was non-zero, a term for u_0 would be present in equation 3.15. The Hankel matrix can be represented by the state-space matrices A, B, and C as follows [15]:

$$\bar{\mathbf{H}} = \begin{bmatrix} \mathbf{CB} & \mathbf{CAB} & \mathbf{CA^2B} & \dots \\ \mathbf{CAB} & \mathbf{CA^2B} & \mathbf{CA^3B} & \dots \\ \mathbf{CA^2B} & \mathbf{CA^3B} & \mathbf{CA^4B} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.16}$$

The values of $\bar{\mathbf{H}}$ are known as it contains shifted copies of the sampled wavelet impulse response. In order to extract the **A**, **B**, and **C** matrices, SVD is utilized and the results are used to determine the observability and controllability matrices.

$$\bar{\mathbf{H}} = \mathbf{U}\Sigma\mathbf{V}^T \tag{3.17}$$

The eigenvectors of $\bar{\mathbf{H}}\bar{\mathbf{H}}^T$ produce the columns of $\mathbf{U}$, and the eigenvectors $\bar{\mathbf{H}}^T\bar{\mathbf{H}}$ produce the columns of $\mathbf{V}$. Σ contains the singular values of $\bar{\mathbf{H}}$ positioned along its main diagonal in descending order. The observability and controllability matrices are extracted via:

$$\begin{aligned} \mathbf{W}_o &= \mathbf{U}\boldsymbol{\Sigma}^{1/2} \\ \mathbf{W}_c &= \boldsymbol{\Sigma}^{1/2}\mathbf{V}^T \end{aligned} \tag{3.18}$$

The $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ matrices can then be extracted from $\mathbf{W}_o$ and $\mathbf{W}_c$ using equation 3.18 and equations 3.4 and 3.5. The analog filter that will impalement the state-space system is 5th order, so $\bar{\mathbf{H}}$ is a 5×5 matrix, $\mathbf{A}$ is a 5×5 matrix, $\mathbf{B}$ is a 5×1 matrix, and $\mathbf{C}$ is a 1×5 matrix. Below in figure 3.3, is the impulse response of the state-space system constructed from the SVD approximation along with the ideal wavelet.

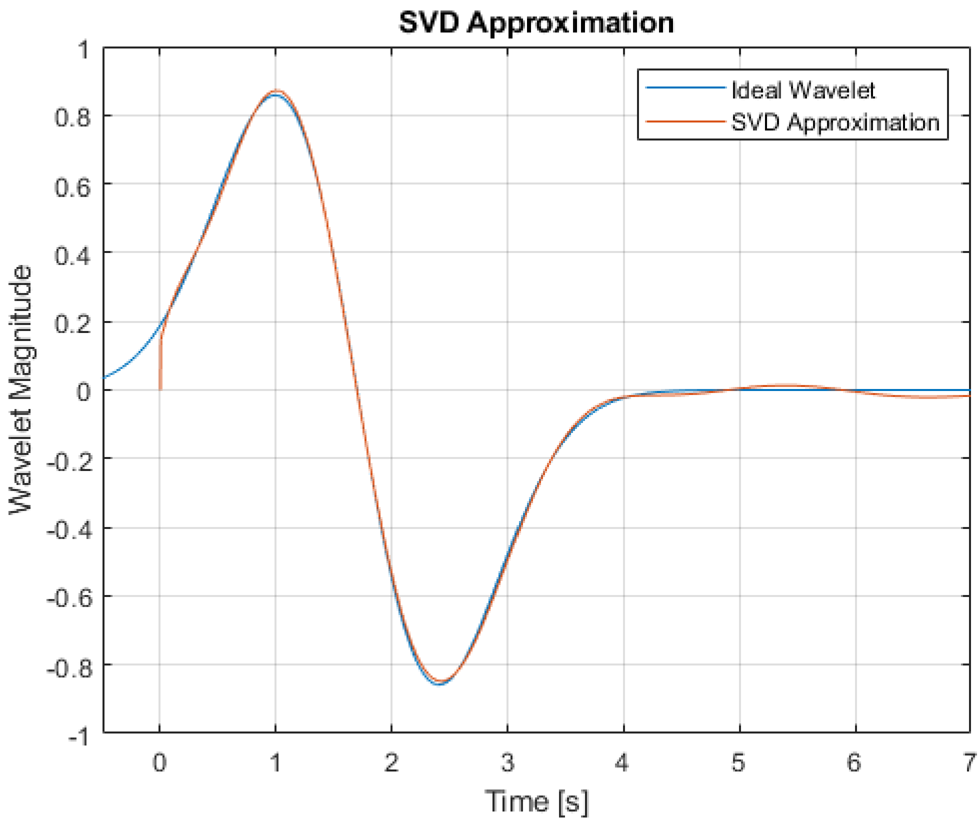


Figure 3.3: SVD approximated impulse response in orange, compared to the ideal first order Gaussian wavelet in blue

In a statistical sense, the SVD state-space representation of the Gaussian wavelet performs far better than the Padé approximate.

Below in figure 3.4 is the z-domain pole-zero map of the SVD approximation, although hard to see with the resolution of the image, all poles are inside the unit circle and the system is stable.

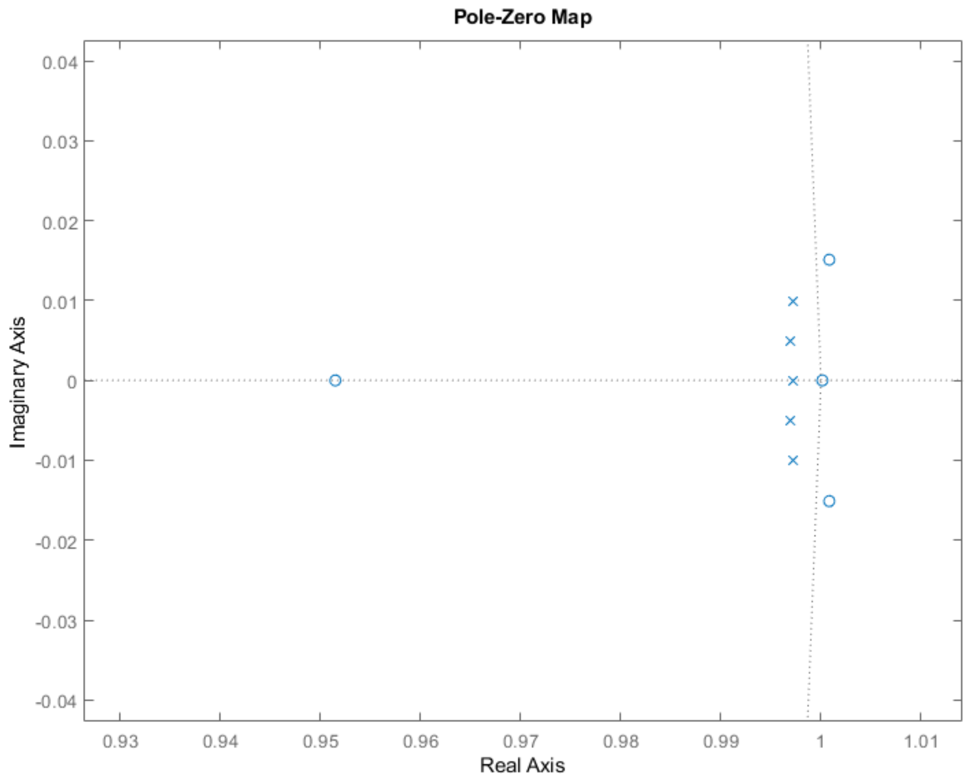


Figure 3.4: Pole-zero map of the SVD wavelet approximation

The SVD approximation is easily converted back into continuous time by the zero-order hold method:

$$f_{ZOH}(t) = \sum_{n=0}^{m} f[n] rect\left(\frac{t - \frac{T}{2} - nT}{T}\right) \tag{3.19}$$

Where $rect(\cdot)$ is the rectangular function, m is the length of the sampled wavelet, and T is the sampling period.

The only downside to the SVD approximation method is that the **A**, **B**, and **C** matrices are fully dense. This means every internal node of the analog wavelet filter would have a connection and gain to every other node, making for a complex, non-repeatable circuit. The state-space matrices can be put into a banded sparse form, called Schwarz form, by using the Lyapunov equation. This is explored in the next section.

3.5 Sparse State-Space Matrix Parameterization

In order to reduce the density of the state-space matrices, a parameterization is employed to reduce the state-space matrices into Schwarz form:

$$
\begin{aligned}
A &= \begin{bmatrix} a_{11} & -\alpha_1 & 0 & 0 & 0 \\ \alpha_1 & 0 & -\alpha_2 & 0 & 0 \\ 0 & \alpha_2 & 0 & -\alpha_3 & 0 \\ 0 & 0 & \alpha_3 & 0 & -\alpha_4 \\ 0 & 0 & 0 & \alpha_4 & 0 \end{bmatrix}, \; B = \begin{bmatrix} b_1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\
C &= \begin{bmatrix} c_1 & c_2 & c_3 & c_4 & c_5 \end{bmatrix}
\end{aligned}
\tag{3.20}
$$

Theorems proving the existence of this Schwarz matrix transform and algorithm for converting applicable state-space matrices are found in the literature and not investigated fully here [17,18,19]. A rough outline of the algorithm to transform the state space matrices into Schwarz form is as follows. The Lyapunov equation is used to solve for the controllability matrix of the original dense state-space representation:

$$
\begin{aligned}
&AW_c + W_cA^T = BB^T \\
&W_c = \int_0^\infty e^{tA}BB^Te^{tA^T}\,dt
\end{aligned}
\tag{3.21}
$$

An equivalent state-space representation is found via a conical mapping:

$$
\Gamma : S_n^{p,q} \longrightarrow S_n'^{\,p,q} \tag{3.22}
$$

where $S_n^{p,m}$ is the set of all minimum state-space systems (A, B, C, D), with n dimensional state space, p dimensional input space, and q dimensional output space. In this case $n = 5$ and $p = q = 1$. State spaces are equivalent if there exists a nonsingular matrix T, such that:

$$
\begin{aligned}
&A_1 = TA_2T^{-1} \\
&B_1 = TB_2 \\
&C_1 = C_2T^{-1}
\end{aligned}
\tag{3.23}
$$

The transfer matrix T is found via the singular value decomposition of W_c, $T = U_{W_c}\left(\Sigma_{W_c}\right)^{1/2}V_{W_c}^T$. This creates a new, normalized state space. The controllability matrix of the new normalized state space, $\tilde{W}_c$, is decomposed using QR decomposition to create an orthogonal matrix Q and upper triangular matrix

R. The normalized Schwarz form is then found via a transform with the matrices Q and R [17]. The state-space matrices now have the form as in equation 3.20.

For circuit design, the non-zero elements in the matrices are relabeled as follows:

$$A = \begin{bmatrix} -k_1 & -k_2 & 0 & 0 & 0 \\ k_2 & 0 & -k_3 & 0 & 0 \\ 0 & k_3 & 0 & -k_4 & 0 \\ 0 & 0 & k_4 & 0 & -k_5 \\ 0 & 0 & 0 & k_5 & 0 \end{bmatrix} \quad B = \begin{bmatrix} k_{11} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
$$C = \begin{bmatrix} k_6 & k_7 & k_8 & k_9 & k_{10} \end{bmatrix} \tag{3.24}$$

As an example, the numerical values for the scale 4 wavelet are below in equation 3.25:

$$A = \begin{bmatrix} -17.47 & -12.95 & 0 & 0 & 0 \\ 12.95 & 0 & -8.176 & 0 & 0 \\ 0 & 8.176 & 0 & -6.643 & 0 \\ 0 & 0 & 6.643 & 0 & -4.68 \\ 0 & 0 & 0 & 4.68 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 5.911 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 0.02412 & 0.107 & 0.01138 & 0.5456 & -0.04336 \end{bmatrix} \tag{3.25}$$

The conversion between the above numbers and the final circuit realization are described in section 4.2.

3.6 MATLAB Simulation

A MATLAB simulation is completed to verify the demodulation and wavelet approximation methods. Starting from an input data stream, the data is modulated, injected with noise, and then passed through the Gaussian wavelet at different scales. Only 10 bits are shown here to make the plots more readable, the BPSK carrier frequency is 10 kHz, the data rate is 1 kbps. See figure 3.5 below:

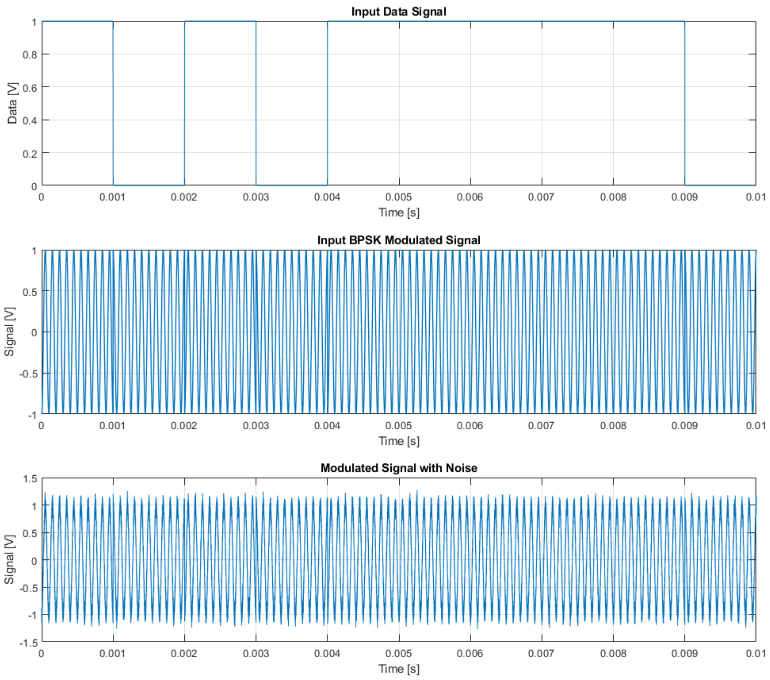


Figure 3.5: MATLAB BPSK input example for wavelet demodulation simulations. Input data on top, modulated data in middle, modulated data with noise on bottom

The modulated data with noise is passed through both the ideal and SVD approximations of the first order Gaussian wavelet. In figure 3.5, the noise level is set such that the signal-to-noise ratio (SNR) is 10 dB. As stated in section 2.3, the wavelet is correlated with the input signal. Figure 3.6, below, contains the ideal output, found with MATLAB's *cwt* function, for varying dyadic scales 1 to 128.

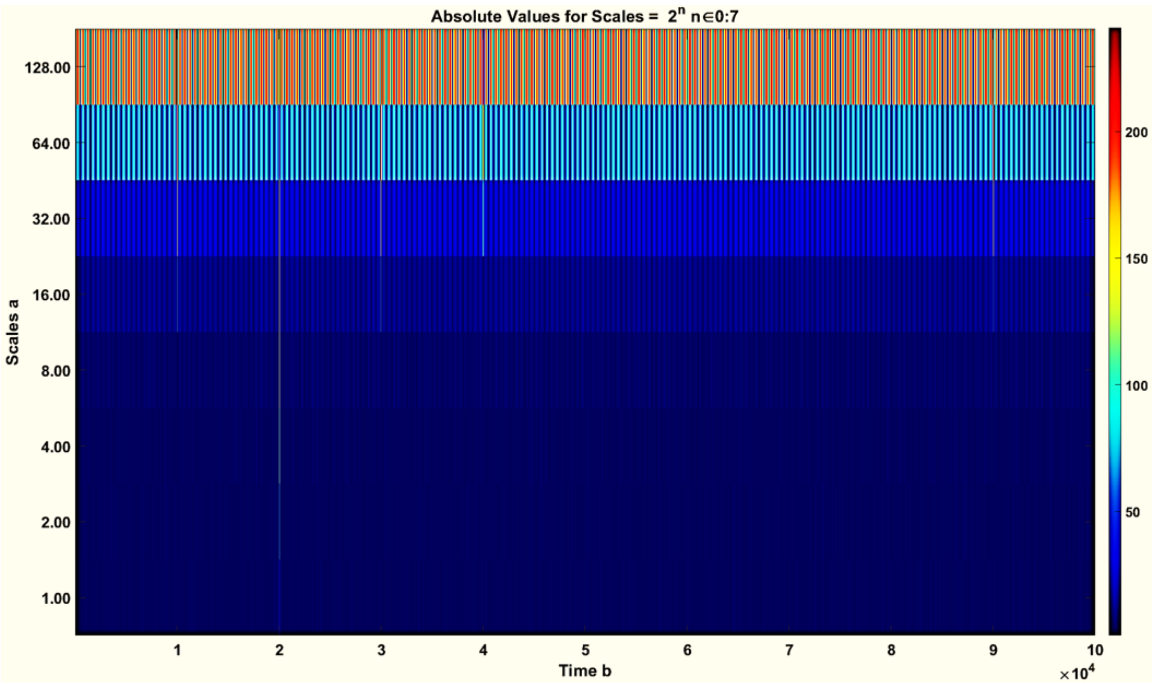


Figure 3.6: Ideal continuous wavelet transform output.

A number of scales experience amplitude changes at times that correlate to bit transitions. This same effect is compared to the SVD wavelet approximation output below in figure 3.7:

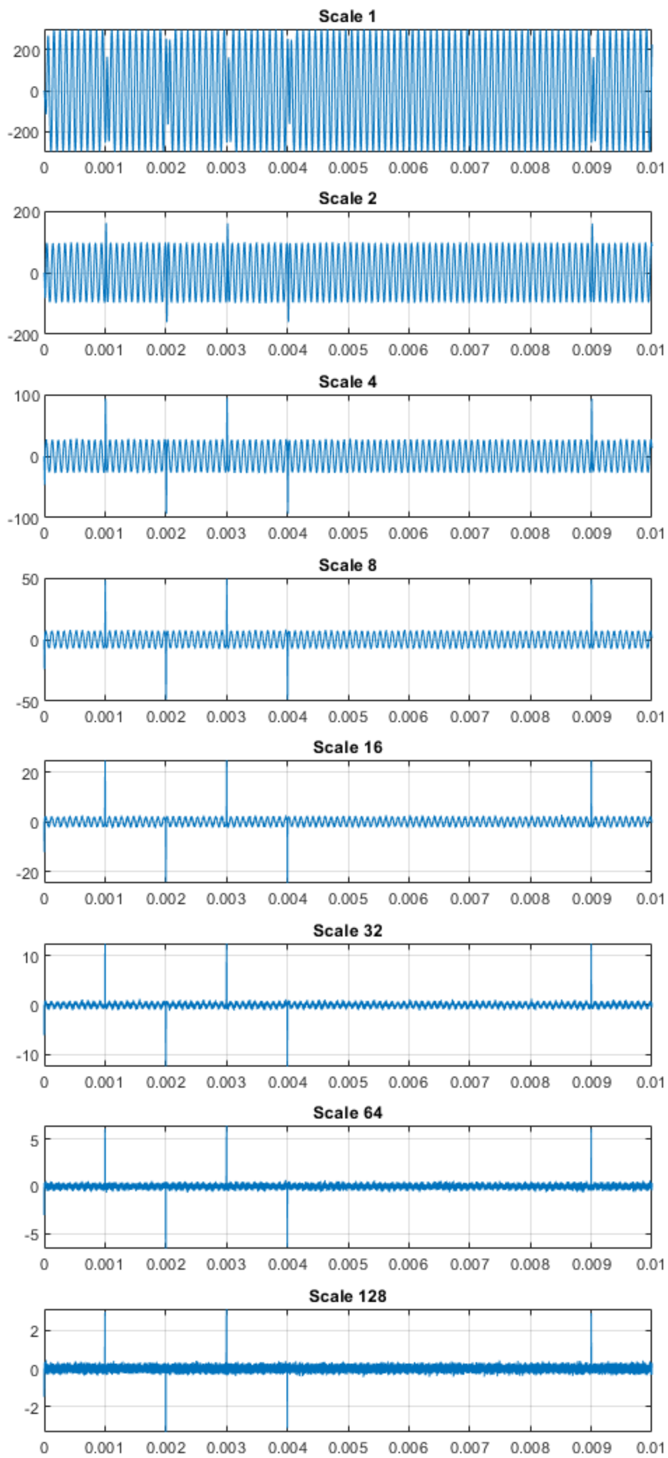


Figure 3.7: SVD approximation CWT output, y-axis is wavelet output magnitude, x-axis is time

In both cases, the outputs from multiple scales can be used to determine when a bit transition occur. In the case of the SVD approximation, scales 4 through 128 can be used directly or summed to gain a more confident answer. This provides a clear indication of the data bit transitions. The data bits themselves are also determined, as the wavelet output is either positive if the bit transition is high to low or negative if the bit transition is low to high.

The large 10 dB SNR in the previous example allows for all of the features of the wavelet transform output to be easily seen. This SNR is increased or decreased in simulation to produce a bit error rate (BER) curve, plotted in figure 3.8 below.

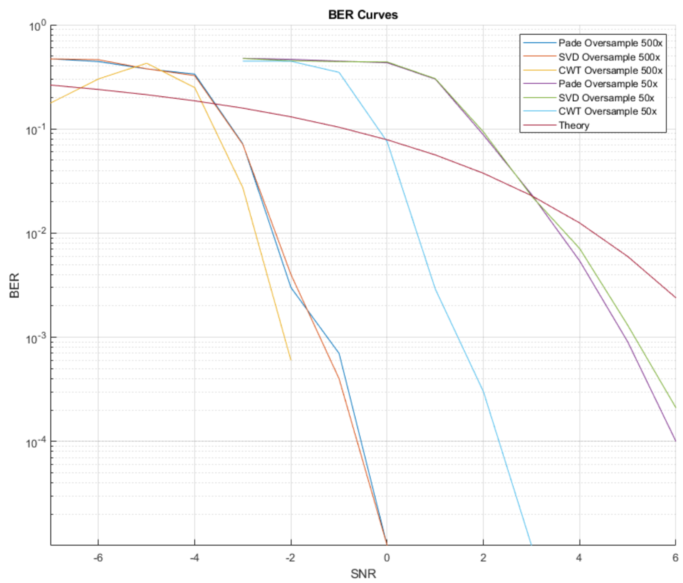


Figure 3.8: BER Curve, ideal SVD and Padé approximations all plotted with different SNRs and oversampling ratios

The Padé and SVD approximations perform similarly while the ideal *cwt* function in MATLAB out performs them both. The theory line represents that of unencoded BPSK through a matched filter. The use of convolutional or Turbo codes would greatly improve the theoretical performance, wherein the BER results would improve orders of magnitude over small SNR increases. The wavelet transform however does not follow the matched filter theory.

In MATLAB all simulations are obviously performed in discrete time, but by oversampling the inputs, outputs, and wavelets an approximation for the continuous time system can be achieved. For this reason, multiple oversampling ratios are considered when performing BER curves, and as such the greater the system is over-sampled, the better it performs. It is noted that less than a million bits were passed through the simulation at each SNR to calculate the error rates in figure 3.8, many more bits are needed to develop concrete statistics, especially at low error rates. These BER results, however, roughly match previous work showing 10^{-4} to 10^{-5} bit error rates at 0 dB SNR [7,8].

Code for these BER simulations and all the wavelet approximations are found in appendix B, wherein the citations for a number of the sub functions used for the wavelet approximations are found.

Chapter 4

Circuit Design

The wavelet transform circuitry is designed for layout on a printed circuit board (PCB) and is made up of discrete components. An integrated approach to the necessary circuitry could have saved on area and allowed for greater complexity, but due to time and monetary constraints, the wavelet circuits are implemented on a PCB.

4.1 Filter Topology Discussion

The circuit implementing each daughter of the first order Gaussian wavelet is, at its core, a 5th order active RC filter. An active RC filter topology is targeted for a number of reasons, the largest of which is manufacturability. There are a number of filter topologies used in the literature to produce an analog wavelet, including transliner, switched capacitor, Gm-C, and active RC filters [3,4,20,21].

Active RC filters can be tuned with resistors alone, while keeping capacitance values constant. Off-the-shelf surface mount resistors come in numerous values with good tolerances, which cannot be said for surface mount capacitors. Additionally, active RC filters are not tuned based on amplifier parameters, opening up more options in the design, since off-the-shelf operational amplifiers and transistors must be used. This is in contrast to capacitive transconductive (Gm-C) or transliner filters, which require highly custom transistor configurations to achieve design performance. These topologies lend themselves well to an integrated circuit but do not allow for much flexibility on a PCB.

Other typologies were considered but deemed inadequate for quick and simple development of the wavelet circuits. Passive filters, for example, do not lend themselves well to the wavelet implementation because of their limited quality factors and they easily fall victim to component mismatch. Active filters employ amplifiers which enable filter gain and better impedance control, and do not require the use of inductors.

With the active RC filter topology the clear front runner topology, the AD822 operational amplifier is selected as the amplifier backbone of the filter. The AD822 has a few key features that enable the overall wavelet filter operation.

These features are summarized below in table 4.1.

Parameter	Value
Output Current	$\pm 15\ mA$
Input DC Offset	$800\ \mu V$
Max Input Ref Voltage Noise	$25\ nV/\sqrt{Hz}$
Slew Rate	$3\ V/\mu s$
Settling Time to 1%	$1.4\ \mu s$
Quiescent Current	$1.3\ mA$

Table 4.1: AD822 Operational Amplifier Characteristics [22]

N-channel junction gate field-effect transistors (JFET) are used to provide a low offset, low noise, high impedance input stage. The low input offset and maximum input referred noise voltage will ensure accurate summation and integration throughout the filter stages. The output current will dictate lower limits on load resistances throughout the circuit. The output saturation resistance is 40 Ω for sourcing and 20 Ω for sinking, these source and load limitations will not be reached by design. The fast slew rate and settling time will allow for the fast voltage transitions necessary to capture the bit transitions in the BPSK waveform. The AD822 amplifiers will be powered from a +5V and a -5V supply.

4.2 5th Order Gaussian Filter Design

As discussed in section 2.3, to implement the analog wavelet transform a filter needs to be designed that has an impulse response identical to the function represented by the wavelet. The Gaussian mother wavelet was represented in a state space, which is easily converted into a circuit. Starting from the matrix equation 3.24, the matrix is expanded into its a system of equations.

$$
\begin{aligned}
x_1 &= -\frac{1}{s}(k_1x_1 + k_2x_2 - k_11u) \\
x_2 &= -\frac{1}{s}(-k_2x_1 + k_3x_3) \\
x_3 &= -\frac{1}{s}(-k_3x_2 + k_4x_4) \\
x_4 &= -\frac{1}{s}(-k_4x_3 + k_5x_5) \\
x_5 &= -\frac{1}{s}(-k_5x_4) \\
y &= k_6x_1 + k_7x_2 + k_8x_3 + k_9x_4 - k_10x_5
\end{aligned}
\tag{4.1}
$$

These equations can be represented by a feedback block diagram, shown below in figure 4.1, with all the states, inputs, outputs, integrations, and summations noted.

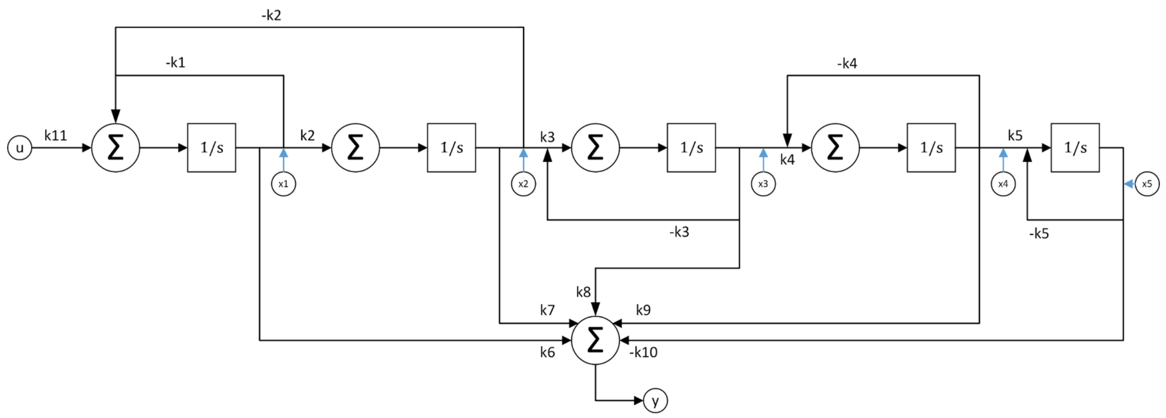


Figure 4.1: Circuit Feedback Diagram

From equation 4.1, it is clear that in order for the system to be implemented with analog circuitry, two functions must be implemented in hardware. An integrator is needed to perform the $1/s$ operation, and a summing circuit is needed for addition and subtraction operations. Operational amplifiers with capacitors and resistors in feedback respectively perform these operations. First, in figure 4.2, the integrator is an op-amp circuit with the following topology:

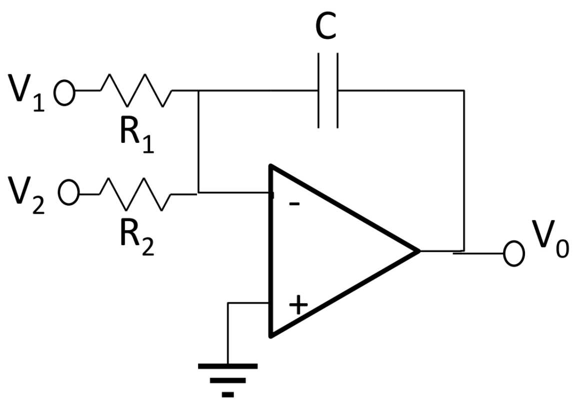


Figure 4.2: Operational Amplifier Integrator Topology [28]

$$V_o = -\frac{1}{s}\left(\frac{V_1}{R_1 C} + \frac{V_2}{R_2 C}\right) \tag{4.2}$$

Equation 4.2 is the transfer function for the operations of the circuit in figure 4.2. Next, in figure 4.3, the topology for the adder circuit:

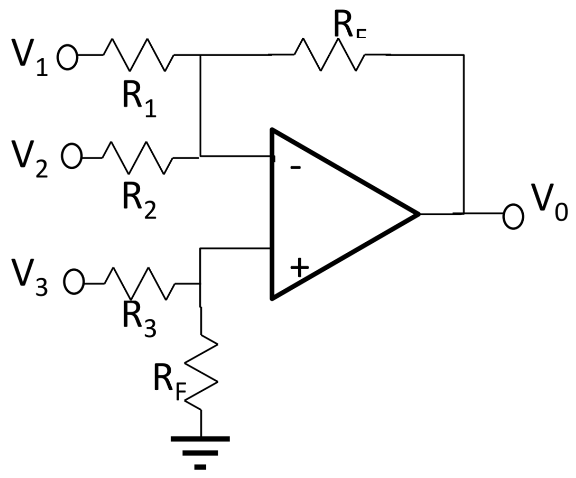


Figure 4.3: Operational Amplifier Summing Topology [28]

$$V_o = -R_F\left(\frac{V_1}{R_1} + \frac{V_2}{R_2} - \frac{V_3}{R_3}\right) \tag{4.3}$$

Again, equation 4.3 is the transfer function of the circuit in figure 4.3. For simplicity, instead of applying voltages to the positive terminals of the op-amps, all negative voltages are achieved by passing through an inverter. An op-amp inverter has the same topology as that in figure 4.3, but with identically valued feedback and input resistors.

By matching the diagram in figure 4.1 and the component circuits in figure 4.2 and figure 4.3, the full wavelet circuit topology is constructed and presented below in figure 4.4.

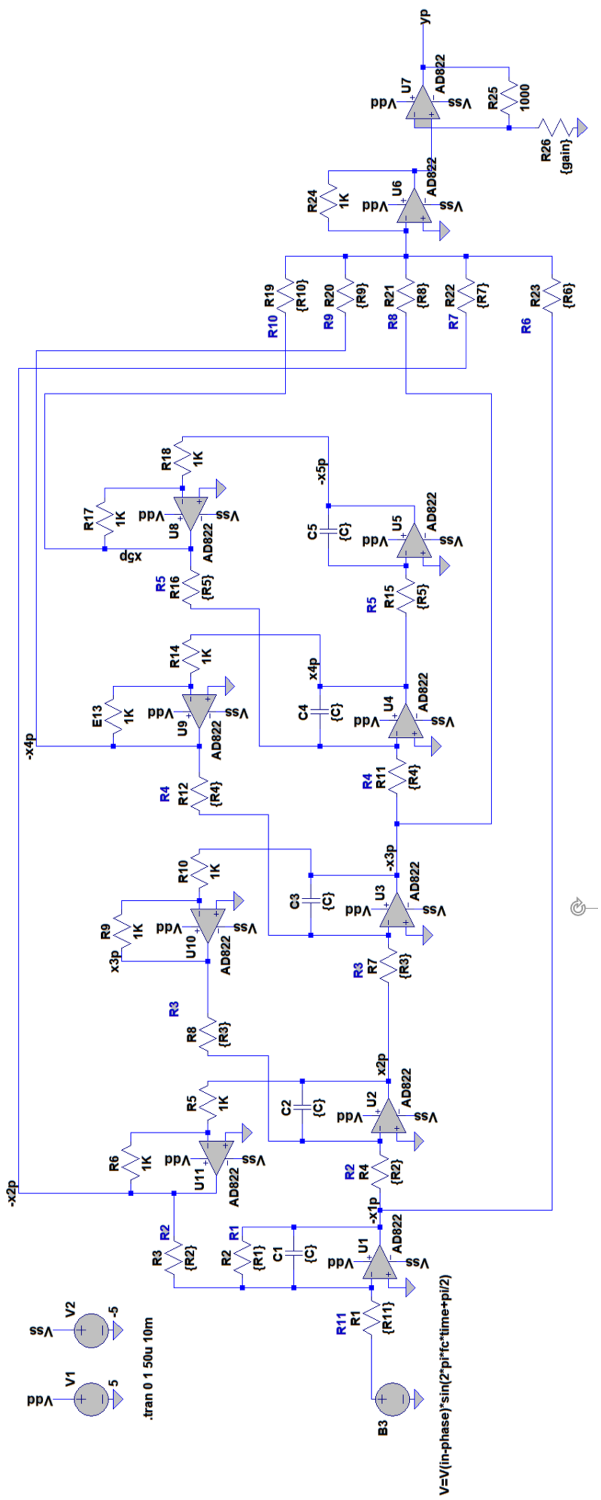


Figure 4.4: Wavelet LTSpice circuit topology

The circuit itself is simulated and tuned in LTSpice. The input to the circuit in figure 4.4 is a BPSK waveform generated in LTSpice for simulation purposes. The final output goes through an additional gain stage, which is used to normalize gain across every wavelet scale. The output of the SVD approximation at every scale has the same amplitude, but due to op-amp nonidealities, the gain across scale is not identical. In figure 4.4, the state-space variable locations are marked with their appropriate sign. When manufactured, all the state-space variables x_1, x_2, x_3, x_4, and x_5 are broken out with both inverted and non-inverted copies for summation at the output of the filter. This allows for easy troubleshooting, if necessary.

Every resistor and capacitor is parameterized in the LTSpice circuit for ease of simulation. The matrices in equation 3.25 are filled out with values for the wavelet scale 4, every scale will have different state-space values. The values are related to the k variables in equation 3.24 and set equal to resistor and capacitor quantities based on equations 4.2 and 4.3.

As discussed in section 4.1, off-the-shelf surface mount capacitors do not have small tolerances, thus every capacitor used across every scale is an identical 1 μF 5% capacitor. This allows for high-tolerance resistors to make the changes from scale to scale. Essentially, moving up in scale by a dyadic ratio will increase the frequency response pass-band by a factor of two. Therefore, most of the resistors are simply scaled down by a factor of two between scales.

The resistors that are used to sum the states at the output of the filter change at slightly different ratios and are set based on LTSpice simulations. The full list of resistor values used in every scale are found in appendix C.

4.3 Circuit Simulations

Simulations are completed for every wavelet scale, but for brevity, only the results for the fourth scale will be presented here.

4.3.1 Impulse Response

The impulse response of the fourth scale is shown below in figure 4.5.

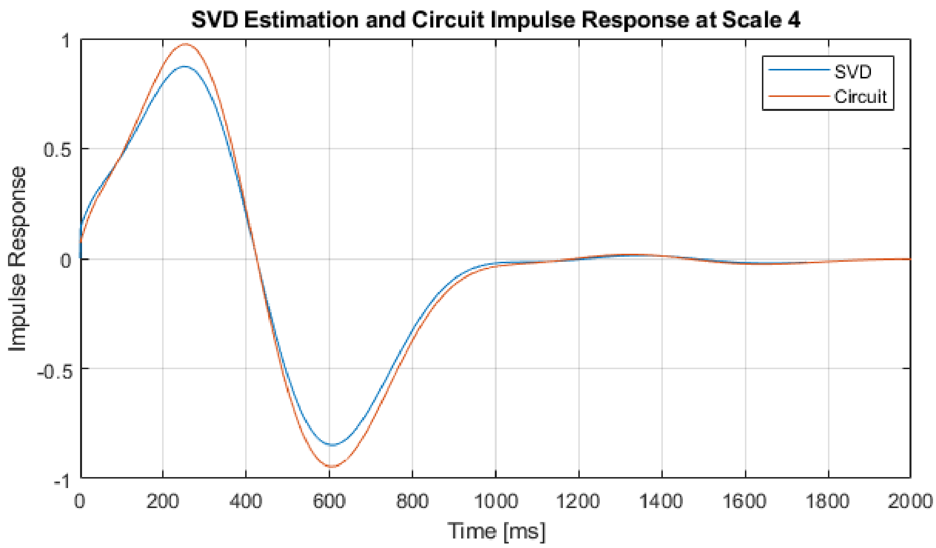


Figure 4.5: LTSpice circuit simulation of the fourth wavelet scale impulse response. Circuit simulation in orange, SVD wavelet approximation in blue

From figure 4.5, it is clear that the impulse response of the circuit simulation and the SVD are nearly identical. The circuit impulse response is slightly larger in amplitude, which is due to the extra gain stage at the output of the filter.

The other notable difference is the impulse response behavior at $t = 0$. As explained in section 3.1, the Gaussian wavelet is centered around $t = 0$, but in order to be modeled as a causal LTI system, the impulse response needs to be shifted forward in time. The response is therefore shifted forward by $t = 1.7$. However, that means that the impulse response does not perfectly start at the origin $(V, t) = (0, 0)$. The circuit simulation response is altered to ensure the impulse response started closer to the origin. This is at the sacrifice of perfect symmetry in the wavelet response, but it allows for a continuous impulse response without the discrete jump at $t = 0$.

4.3.2 Frequency Response

The impulse response and frequency response of the circuit are Fourier Transform pairs:

$$\mathcal{F}(\psi(t)) = \mathcal{F}^{-1}(\hat{\Psi}(\omega)) \tag{4.4}$$

where $\hat{\Psi}(\omega)$ is the wavelet in the frequency domain. The impulse response of the fourth wavelet scale is found in LTSpice and presented below in figure 4.6.

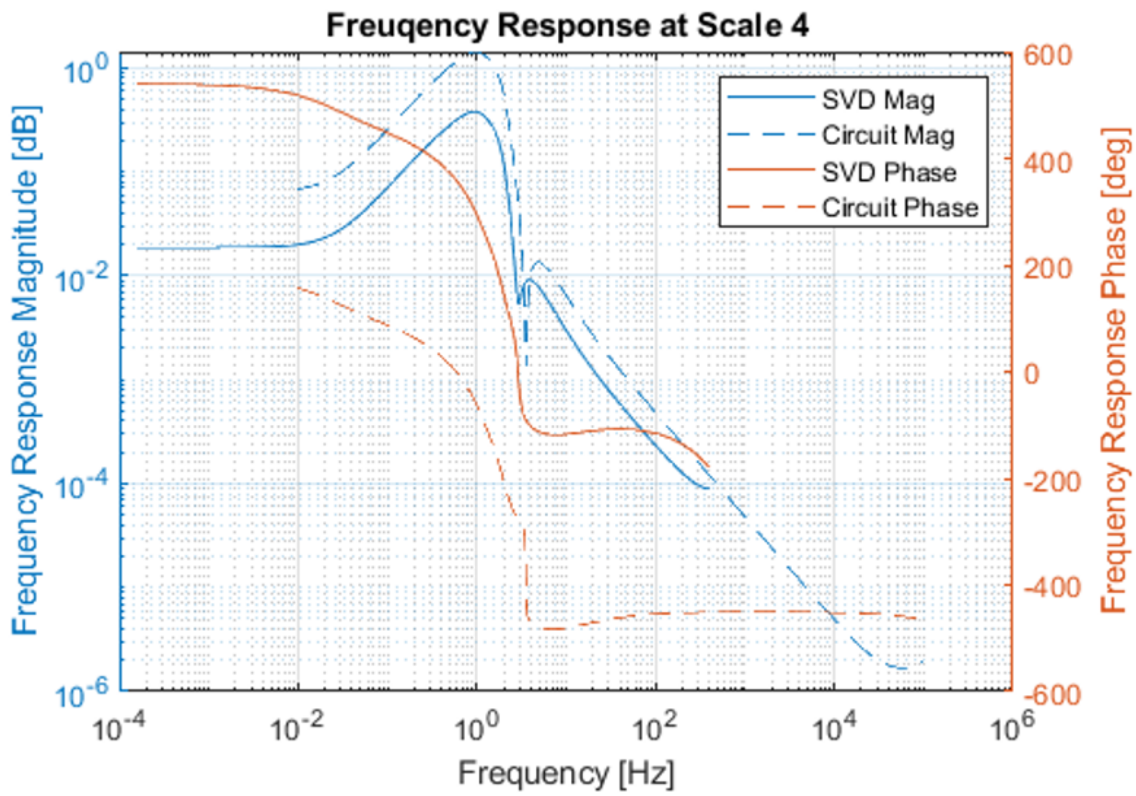


Figure 4.6: LTSpice simulation of wavelet frequency response. Magnitude in blue, phase in orange. Circuit simulation is the dotted line, SVD approximation is the solid line

The wavelet circuit frequency response is generally a close match to the SVD approximation. There is a 360 degree difference between the SVD approximation and the circuit simulation. The difference in gain can be traced to the difference in wavelet impulse response amplitude, shown in figure 4.5. Additionally, there is an extra inversion at the output of the wavelet filter in

LTSpice caused by the final gain stage. In theory, the demodulation method described in chapter 1 should work regardless of the inverted or non-inverted nature of the Gaussian wavelet.

4.3.3 Monte Carlo Simulation

As mentioned in section 4.1, the circuit is tuned form scale to scale by changing the resistor values. All the resistors used have a 1% tolerance, and with such tight bounds on their value, part-to-part variation in resistors will have a minimal effect on the circuit performance. However, the 1 μF capacitors used in feedback of every integrators, only have a 5% tolerance. A Monte Carlo simulation was performed with 5% variation across every capacitor used in the circuit. Figure 4.7 below shows the results:

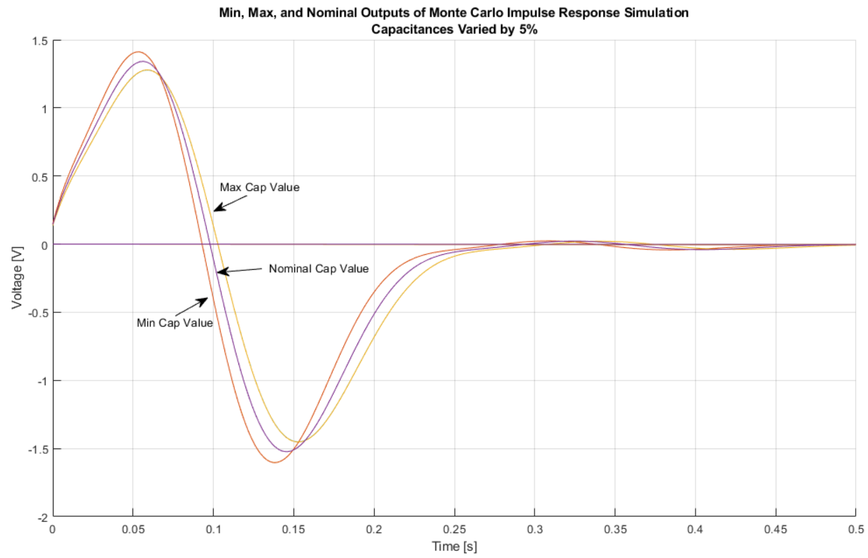


Figure 4.7: Monte Carlo simulation with varying feedback capacitor values. Minimum (0.95 μF), nominal (1 μF), and maximum (1.05 μF) values shown

The Monte Carlo simulation is completed on the scale 64 daughter wavelet circuit. As seen in the above figure, the varying capacitor values act to compress or dilate the overall wavelet impulse response. Compression is caused by smaller capacitance values, while dilation is caused by larger capacitance values. These changes result in the zero crossing of the wavelet response to shift by 4.8 ms to 4.95 ms away from nominal. This corresponds to less than a 2% change in the zero crossing time when compared to the nominal impulse response. This variation is small enough to ensure that there will no interference between the dyadic scales. Interference would occur if one scale compresses or expands so much that it begins to look like an adjacent dyadic scale.

4.3.4 BPSK Demodulation in LTSpice Simulation

All of the scales are created in LTSpice and are provided a simulated BPSK input to gain a closer approximation of their performance in the real world. The BPSK waveform is generated in LTSpice using voltage function generators. Below in figure 4.8 is the time domain output of the wavelet filter implementing the 512th scale with BPSK input.

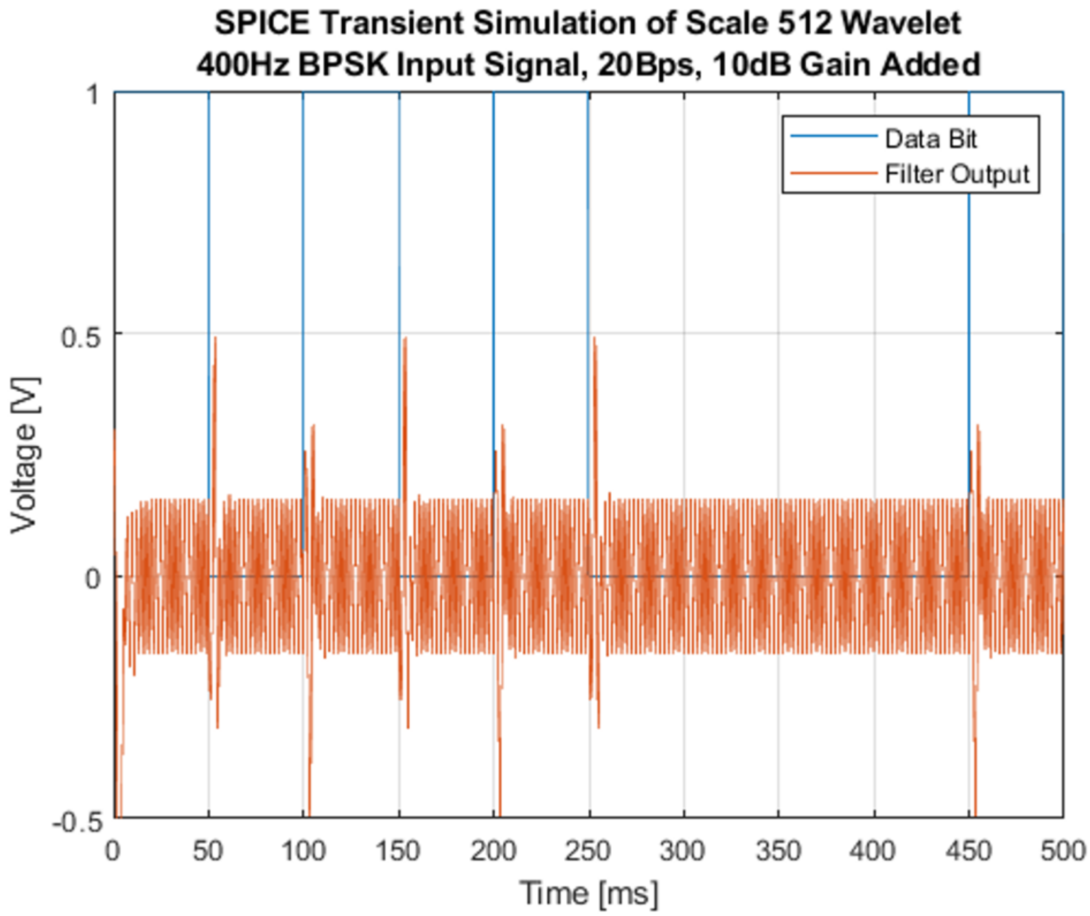


Figure 4.8: BPSK input to 512th wavelet scale in LTSpice, plot shows filter output and data input.

As the figure above shows, the wavelet filter output has a high or low spike in its response according to a one-to-zero or a zero-to-one data bit transition respectively. Detecting this characteristic is what allows for the demodulation of the BPSK waveform.

In figure 4.8, the input carrier frequency of the BPSK waveform is only 400 Hz. A 400 Hz wave period is 2.5 ms, and the length of the 512th scale impulse response is closely related at about 7.5 ms, allowing the wavelet to pick out the bit transitions. The wave period of the BPSK carrier frequency needs to be closely related to the impulse response length of one of the wavelet scales or else performance will be degraded. Multiple dyadic scales, however, are effective at picking up the BPSK waveform for a given carrier frequency.

The result in figure 4.8 indicates that when implemented on a PCB, some demodulation process will be able to be performed. This real-world testing and performance is described in chapter 5. But first, the remainder of this chapter is used to discuss the secondary circuit operations that enable the entire system to operate.

4.4 Digitization

The outputs of each wavelet scale are summed with an amplifier. Refer to section 4.2 for the summing amplifier design. A switch is used to turn on and off the wavelets that are summed at run time. Both inverted and non-inverted versions of the wavelet transform output are available for summation. This helps with debugging and optimizing the circuit output. A second amplifier with a potentiometer in feedback is used to provide a variable gain on the summed output.

The signal is then scaled from a bipolar output, with voltages stretching from -5 V to +5 V, to a unipolar output, with voltages spanning 0 V to +5 V. The analog-to-digital converter (ADC) used, AD7476A, only accepts a unipolar input. When converting to unipolar, the voltage is divided by two, ensuring no damage condition for the ADC can be reached.

The 12 bit AD7476A ADC allows for a 1.2 mV resolution, more than sufficient for the target application. The ADC then pipes this data out serially over an serial peripheral interface (SPI) data line at a rate as high as 1 MBps [23]. Pins on the PCB are also provided to directly output the unipolar or bipolar summed signal. This allows the MCU's internal ADC or an oscilloscope to view the wavelet transform output, which can be used for troubleshooting or performance verification. At 1 MBps, a two byte data word will be transferred at a rate of 500 kHz, and with a maximum input carrier frequency of 10 kHz, this will enable an oversampling of the Nyquist rate by a factor of twenty five. This is more than sufficient to resolve the fast transients in the wavelet output and detect the BPSK phase transitions.

In simulations, oversampling the Nyquist rate by as little as a factor of four yielded workable results. The greater the oversampling rate, the better the noise performance of the system. Because the signal needs to be sampled faster then the Nyquist rate, extremely high data rates become impractical.

The ADC SPI line is driven via a SAMD21 Cortex-M0+ MCU on an Arduino MKRZero development board. The clock rate of the SPI line can be varied from the MCU to a maximum rate of 20 MHz. The MCU itself is clocked with an internal crystal at 48 MHz [24]. The MCU will use a direct memory access (DMA) channel to access the contents of the ADC buffer without sacrificing clock cycles. The contents of the SPI buffer will then be available to the main MCU processor, where a simple threshold comparison can be completed to determine the current bit present at the output of the wavelet circuit. The threshold is found experimentally and discussed further in chapter 5. The demodulated bit is then transferred over a low-rate serial connection to a PC running MATLAB. A block diagram of this digital architecture is below in figure 4.9.

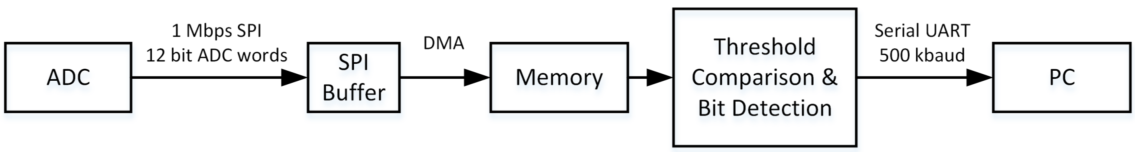


Figure 4.9: SAMD21 digital architecture

The data rate is at least an order of magnitude less than the carrier frequency, thus allowing for a low-rate serial connection to a PC at, a more than sufficient, 500 kbuad, lower rates of 115.2 kbuad are also tested. Embedded C code for the MCU to perform these digital operations is found in appendix B, where all citations for source code are present.

4.5 Board Design

All of the circuits described above are laid out on a PCB, and the schematics for the board are found in appendix A. The board has dimensions of 4.2” by 6.1”, comprising an area of 25.62 sq in. The sole dielectric is FR-4, a cheap composite material made of fiberglass, which is more than sufficient for the low frequency application of this design.

Besides all of the wavelet scales described in section 4.2 and the output circuitry used for digitization described in section 4.4, the board also contains the necessary power circuitry for the AD822 operational amplifiers and the on-board ADC. A single 8 V supply can be used to power the entire board. The 8 V is used directly by a precision low-dropout (LDO) regulator to provide the supply voltage for the ADC. The ADC7476A does not use an external analog voltage reference, but instead requires a precision power supply, here provided by the REF195 reference from Analog Devices. The 8 V is regulated down to 5 V using a MAX5035B chip, which is then split into +5 V and -5 V rails by a TP65133 split-rail boost regulator from Texas Instruments. The TP65133 regulator can provide 250 mA on each rail, which is sufficient to power the total of 156 AD822 amplifiers used across every wavelet scale. Even when derated by 50% there should be no power concerns. The board also has an option of providing ±5 V directly, which can be used for circuit troubleshooting. A rendering and annotated picture of the layout is found below in figure 4.10.

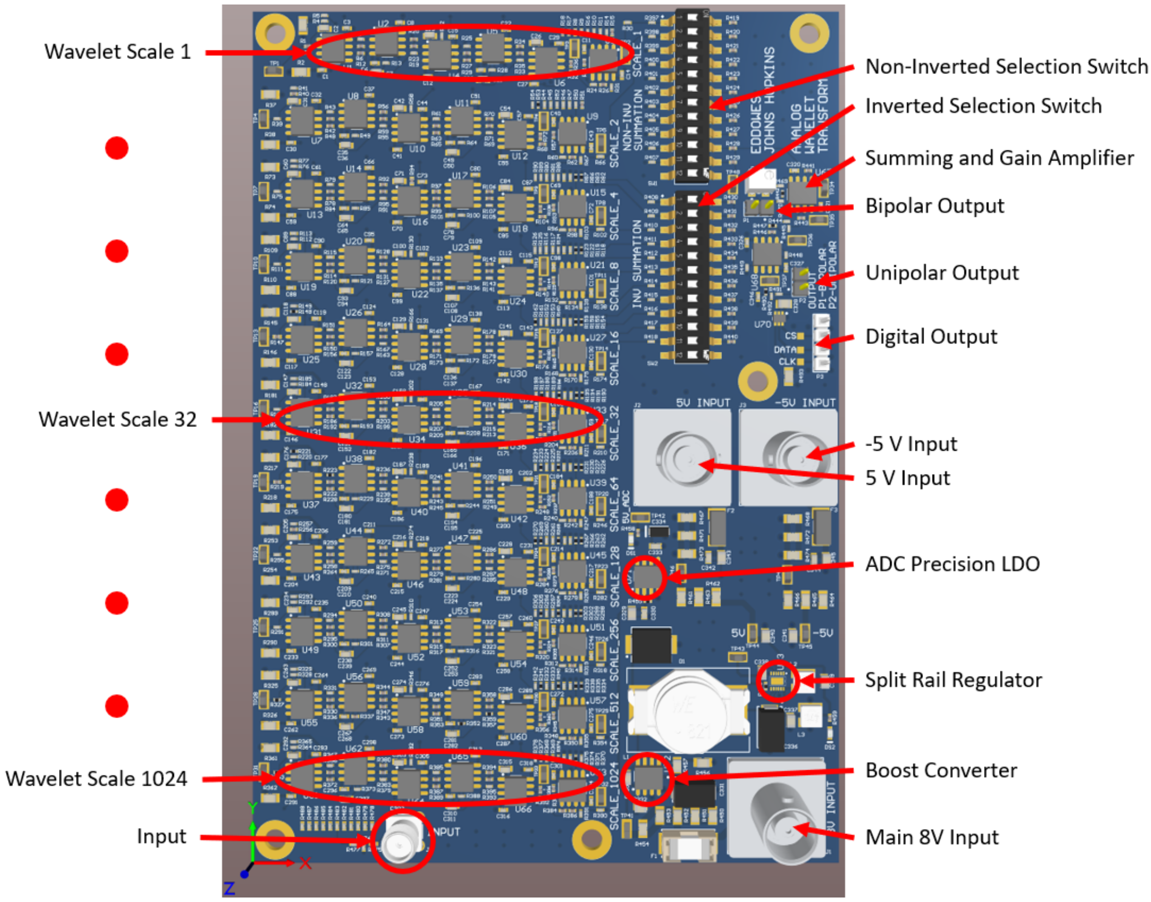


Figure 4.10: Rendering of the final PCB Layout. PCB schematics, layout, and 3D renderings all created in Altium.

In figure 4.10, the input to the circuit is in the bottom left of the image and the output is in the top right. The larger BNC connectors are used for power input to the system. There are test points throughout the board at critical locations to ensure easy troubleshooting of the entire circuit. Below is a picture of the PCB in figure 4.11.

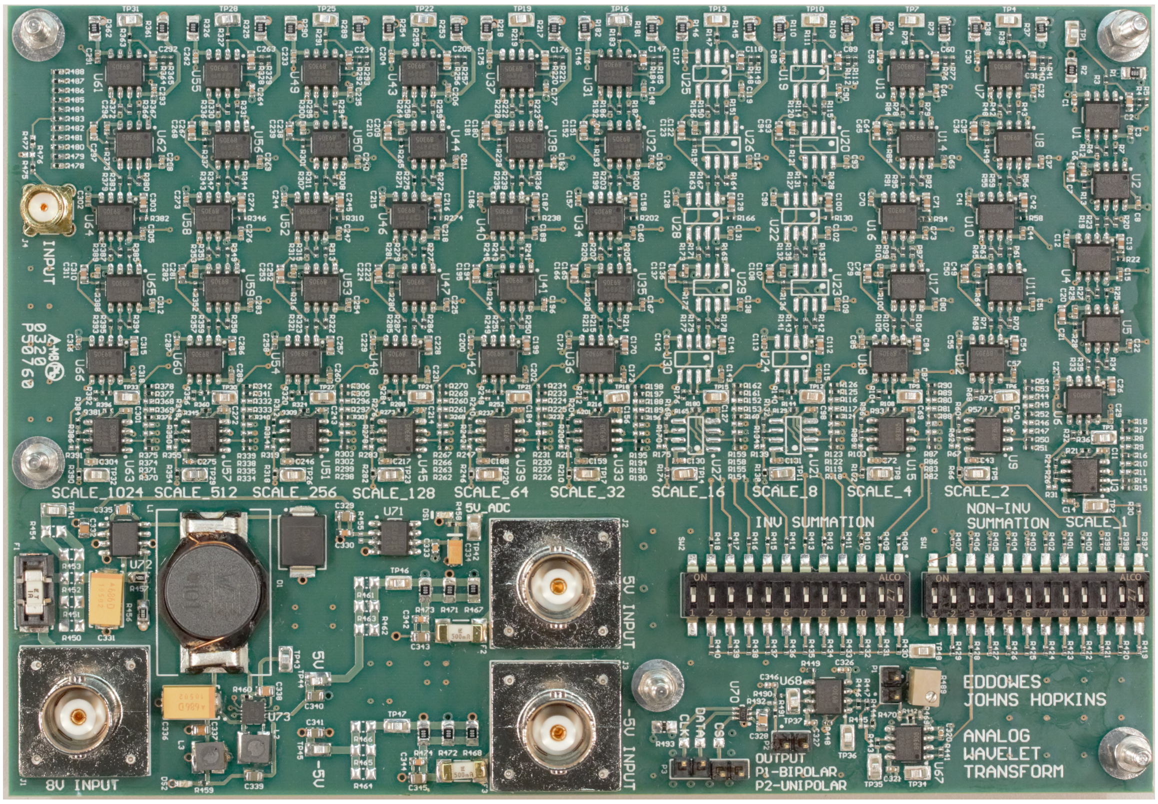


Figure 4.11: Photograph of the final wavelet PCB

The board was assembled via hand component placement and a single reflow step using SnAgCu based solder paste. Not every part is populated in the final board assembly, primarily because of part availability and cost. A complete bill of materials is found in appendix D. The next chapter will explore the measured results and circuit performance.

Chapter 5

System Performance and Results

Once assembled, some care was taken to power on the board with the dedicated +5 V and -5 V inputs, sourced from a known power supply. This minimizes the risk of immediately using the single 8V supply and testing all power circuitry immediately. The entire board uses 163.8 mA current, or 819 mW power, in steady state with all circuits operating. The first step to verifying circuit performance is to measure the impulse and frequency response of the wavelet scales. Once the filters for each scale are verified to properly implement the Gaussian wavelet, the circuit can be used for BPSK demodulation.

5.1 Measured Wavelet Scales

Without too much difficulty, all of the wavelet scales performed as simulations predicted. Each wavelet scale's impulse response matched simulation in every aspect except amplitude. Most scales needed gain adjustments so that their impulse response amplitudes were normalized. These gain differences could be caused by op-amp nonidealities that were not modeled in LTSpice or more likely due to passive component tolerances. To measure the impulse response, an arbitrary waveform generator is used to generate a 20 ns wide pulse with 2 V peak amplitude. This pulse is not a pure Dirac delta, and in testing, multiple copies of this pulse were sent in quick secession. This pulse train input provided a nicely behaved impulse response that matched simulation in everything except amplitude.

For brevity, only the results for the 256th scale wavelet will be presented here. The measured impulse response of the wavelet is presented below in figure 5.1 and compared with the LTSpice circuit simulation as well as the SVD numerical approximation and the ideal wavelet:

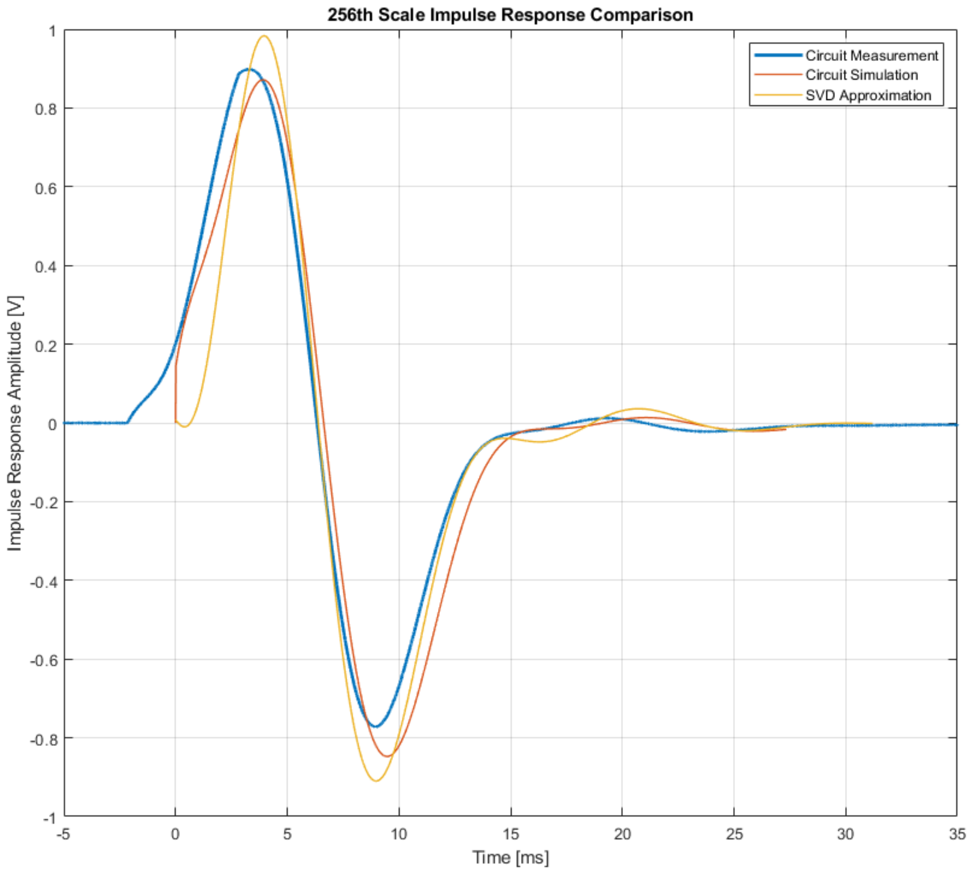


Figure 5.1: Measured 256th Gaussian wavelet scale impulse response in blue, LTSpice simulation in orange, and SVD approximation in yellow

As is evident from figure 5.1 above, the real world filter does a good job at implementing the Gaussian wavelet. The measured impulse response appears to have its peaks and zero crossing slightly earlier than the simulations, which is likely due to the integrator feedback capacitors values being slightly less than their specified 1 μF value. Additionally, in figure 5.2, the frequency response of the 256th scale is plotted along side the LTSpice impulse response.

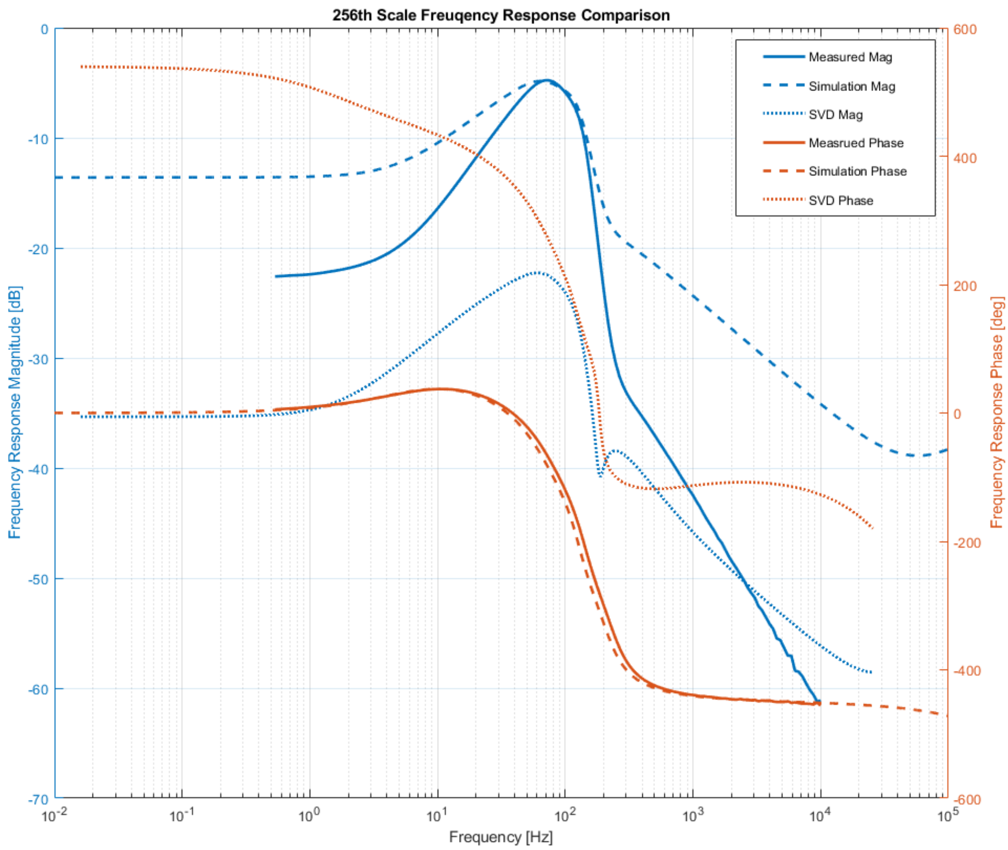


Figure 5.2: Measured 256th wavelet scale frequency response is the solid line, LTSpice simulation is the dashed line, and SVD approximation is the dotted line

The measured frequency response matches the simulated response very nicely, especially in phase. The measured magnitudes peaks at the same frequency as in simulation but falls off more rapidly at higher and lower frequencies. At higher frequencies, the measured responses rolls off at 20 dB/dec, whereas the simulated response rolls off at 10 dB/dec, which could be do to a parasitic pole, arising from board layout or more likely measurement equipment. Differences between the LTSpice simulated frequency response and the ideal response are discussed in section 4.3.

These lab measurements provide a good indication that the circuit is operating as simulated. This means that in theory, the output to these filters should be the Gaussian wavelet transform of the input.

Before BPSK demodulation can occur, the output stage of the circuit is tested. This output, described in section 4.4, performs wavelet summation and overall gain adjustments before scaling the output for an ADC on the PCB. There was a DC biasing problem in the final stage before the ADC, due to the topology of the circuit on the PCB. The easiest fix was using an external bread-boarded circuit using through-hole components and an AD822 op-amp. The output stage before the ADC transformed as follows in figure 5.3:

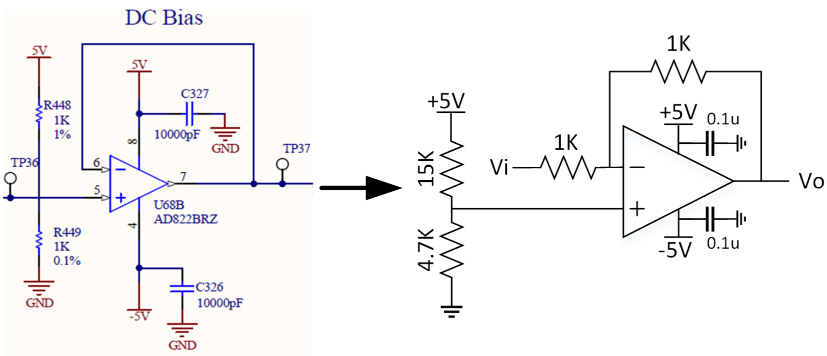


Figure 5.3: Changes to the DC Bias Circuit. As built on the left, final working bread-board circuit on the right

Due to these DC biasing changes the on-board ADC7476A ADC is not used. The output of this circuit is fed directly into the ADC in the SAMD21 Cortex-M0+ processor, bypassing the ADC7476A. The new DC bias circuit itself biases the output with 1.2 volts DC. This isn't exactly half of the 3.3 V input range of the SAMD21 on-board ADC, but the ADC becomes nonlinear with input voltages in the top 25% of its 0 to 3.3 V input range. This may have been due to the specific MCU used, but the issue was not investigated further. This on-board ADC is capable of 350 kSps at 12 bits of resolution. This is the same resolution and only slightly less throughput than the maximum 500 kHz sample rate of the planned AD7476A ADC. This new reduced sample rate can still sufficiently capture the wavelet transform output. From here the software architecture described in section 4.4 remains the same except now a DMA channel is used to access the SAMD21 ADC buffer and not the SPI buffer.

The next step is verifying the BPSK demodulation.

5.2 BPSK Demodulation

The BPSK signal is generated from an arbitrary waveform generator, which is programmed to send a 20 bps BPSK signal with a 1 kHz carrier wave. The generator is commanded to send a repeating one-zero pattern.

Most testing is done using the upper four wavelet scales, 128th, 256th, 512th, and 1024th. All of the data presented below comes from using the summation of both the 256th and 512th scale. Below in figure 5.4 is an oscilloscope screenshot of the BPSK input and the summed wavelet output.

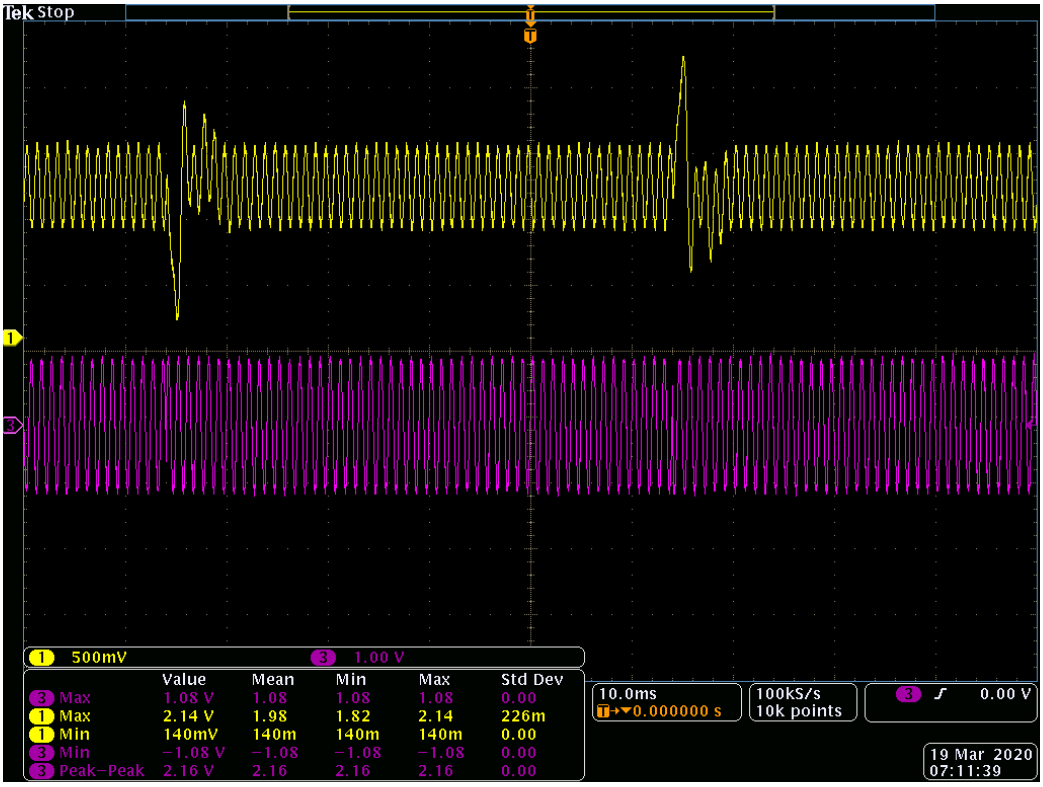


Figure 5.4: BPSK demodulation using the 256th and 512th wavelet scales. 20 bps data rate and a 1kHz carrier frequency. Purple plot shows BPSK input, yellow plot shows wavelet transform output

The positive and negative peaks in the wavelet output correspond to the bit transition in the BPSK waveform. They are distinguished digitally via a simple threshold comparison in the microcontroller to demodulate what bit is sent. The 12 bit ADC will measure a value larger or smaller than nominal when a bit transition occurs. This real world result matches the simulated result in figure 4.8 closely. A more detailed picture showing the wavelet transform output is below in figure 5.5:

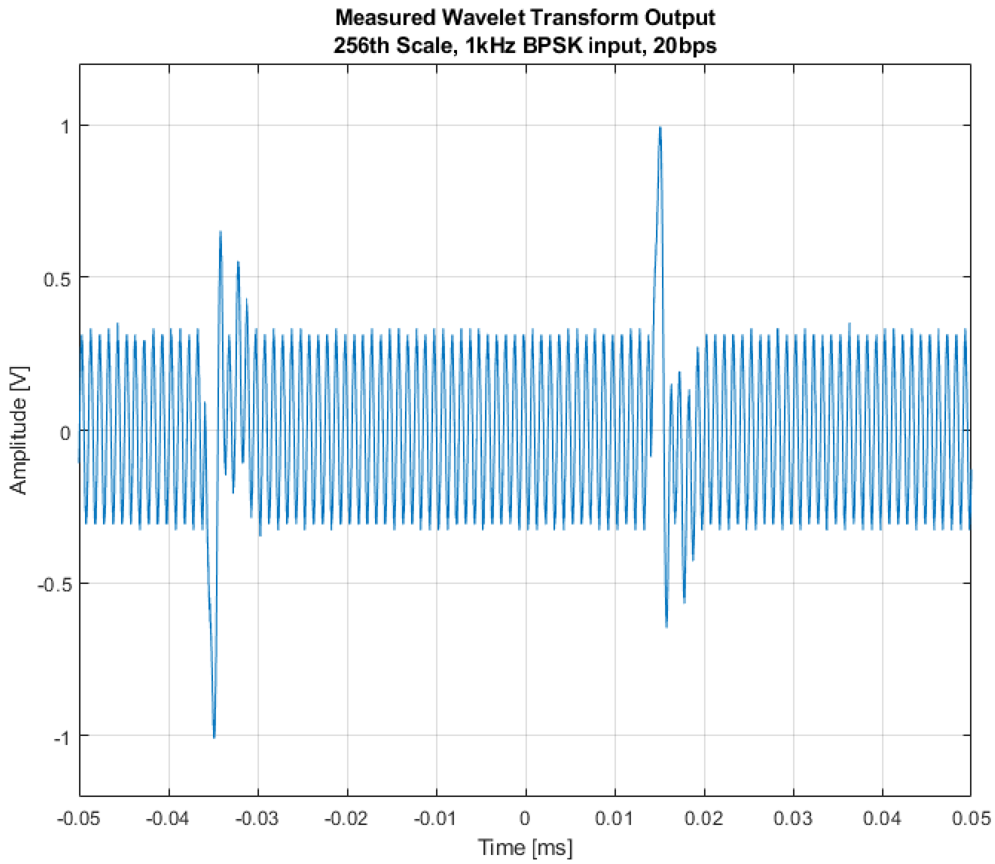


Figure 5.5: Measured BPSK output of 256th wavelet scale. 20bps data rate and a 1 kHz carrier frequency

This positive result, that matches simulation, shows that the real world filters are in fact implementing a Gaussian wavelet and the circuit is performing an analog wavelet transform. The next step in system verification is performing a BER test, which requires quantifying system and signal noise.

The arbitrary waveform generator also has the ability to generate and inject noise into the BPSK signal. An oscilloscope screenshot, below in figure 5.6, shows the wavelet transform output with roughly 3 V of peak-to-peak noise injected into the signal. The arbitrary waveform generator is set to use a 10 MHz bandwidth for noise generation.

Figure 5.6: BPSK demodulation using the 256th and 512th wavelet scale with noise injection. 20 bps data rate and a 1 kHz carrier frequency, 3V peak to peak noise added to BPSK waveform. Purple plot shows BPSK input, yellow plot shows wavelet transform output

As the figure above shows, the wavelet transform is still able to determine where bit transitions occur, albeit with a smaller peak-to-peak amplitude than the case without noise. The injection of noise allows for rudimentary bit error rate calculations. To determine the amount of noise actually injected into the signal of interest, a ratio between the injected signal power and noise power is representative of the system SNR:

$$SNR = \frac{P_S}{P_N} = \frac{\frac{V_{rms}^2}{R_{in}}}{\frac{N_{rms}^2}{R_{in}}} = \frac{V_{rms}^2}{N_{rms}^2} \tag{5.1}$$

The root mean squared voltage at the input of the circuit is kept constant at 176 mV, and the noise voltage is increased or decreased as the test is run. The signal power is measured as -2.4 dBm in a 50 Ω interface. The noise power fluctuates between -20 and +5 dBm. There is some room for measurement error here as the input impedance of the wavelet circuits is never explicitly measured, thus power may be lost and not taken properly into account in the SNR calculation.

The SAMD21 MCU communicates with MATLAB on a PC and sends the demodulated bits. Because the waveform generator is sending a known one-zero BPSK pattern, MATLAB can posteriori determine what bits it should be receiving and calculate the bit error rate. This bit error rate is plotted in figure 5.7 alongside the previous simulated BER curve.

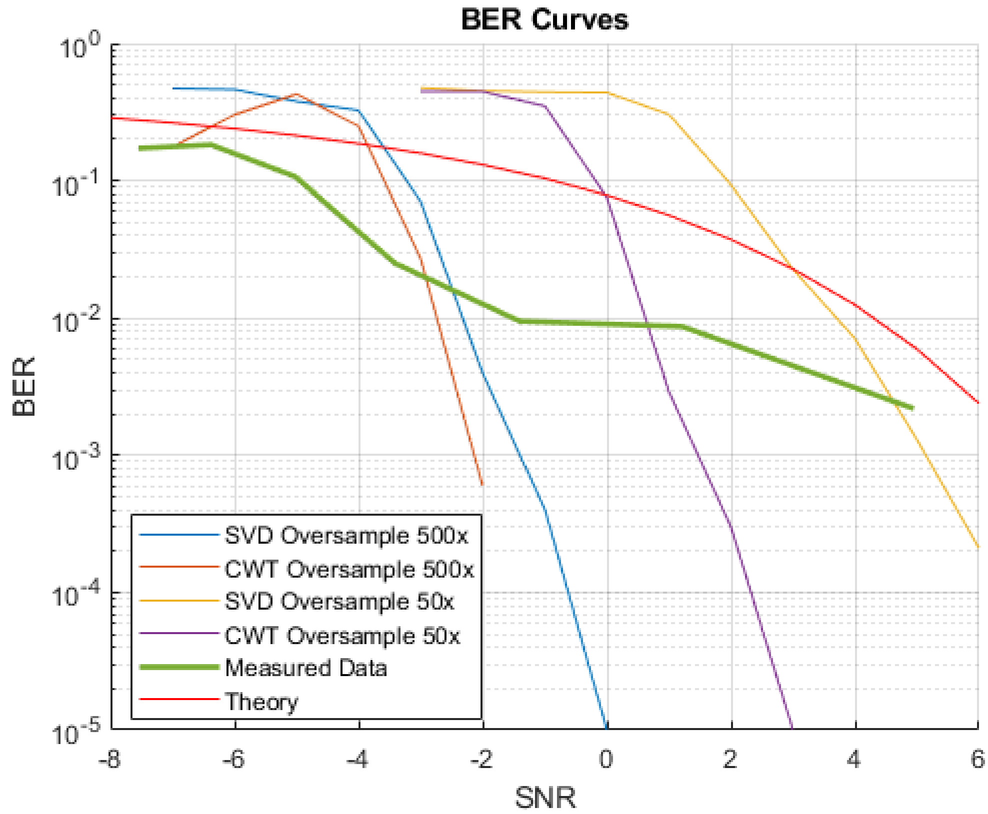


Figure 5.7: Measured bit error rate

For low SNR, the measured bit error rate performed slightly better than the simulated results, but did not experience the large roll-off in errors experienced at high SNRs shown in simulations. No comparison to an optimal matched filter is made in hardware, although a comparison to unencoded BPSK theory is presented in figure 5.7 above. The measured BER roughly follows the unencoded theory. There are a few possible explanations for the degraded measured performance. For one, the threshold comparing taking place in the SAMD21 is likely not optimized. This could lead to false positives in the threshold detection. Or the optimal threshold values are noise dependent, and since the thresholds are not changed once set, performance degrades over varying SNR.

To close the body of this thesis, a picture of the bench-top test setup is presented below in figure 5.8:

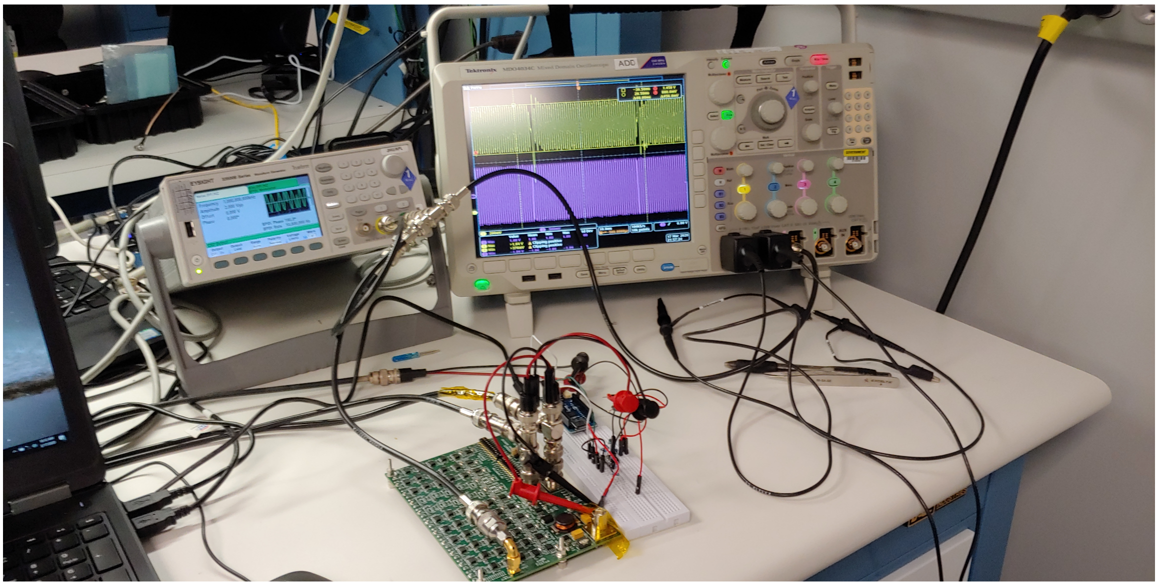


Figure 5.8: Lab bench setup showing wavelet transform circuit, external bread board circuit, Tektronix oscilloscope, and Keysight arbitrary waveform generator

Chapter 6

Conclusion

The objective of this thesis is to build an all-analog wavelet transformation to perform BPSK demodulation. The design of the analog wavelet transform is started from investigating a number of numerical methods for approximating an ideal Gaussian mother wavelet. A method of singular value decomposition is used to convert ideal daughter wavelet into a single input single output state-space representation. This is then converted into a 5th order active-RC filter topology. Ten different wavelet scales and filters are designed, their real-world and simulated parameters are verified to match, and thus a successful all-analog wavelet transform is created.

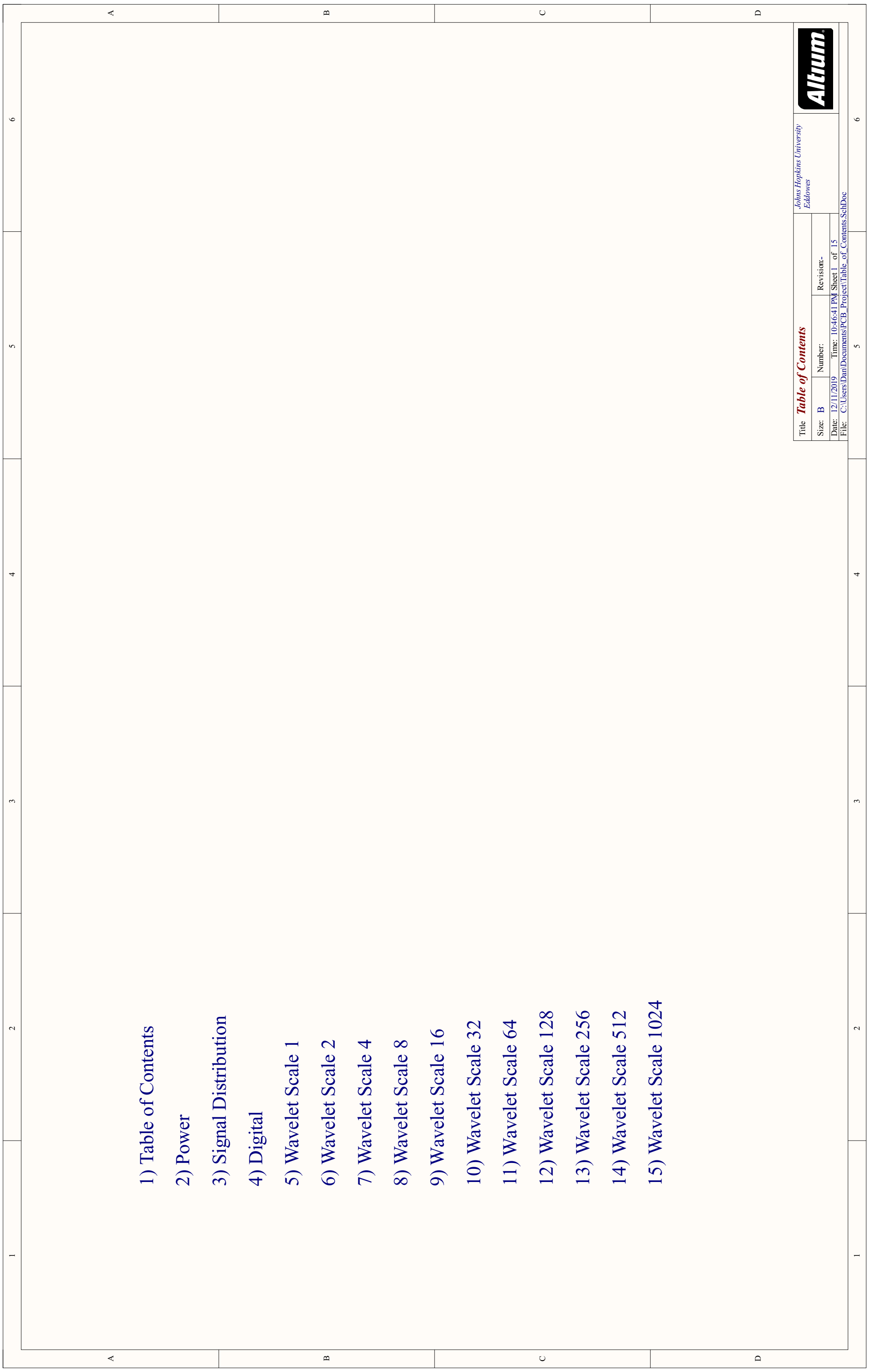
The full verification of BPSK demodulation was not completed in all scenarios, however the system did faithfully show BPSK demodulation is possible. The wavelet transform can determine bit transitions in the BPSK waveform with high fidelity even in the presence of significant noise. There are some shortcomings, likely due to non-optimal threshold comparisons, that results in a deviation of the measured bit error rate from simulation. Some tuning may be able to resolve these issues.

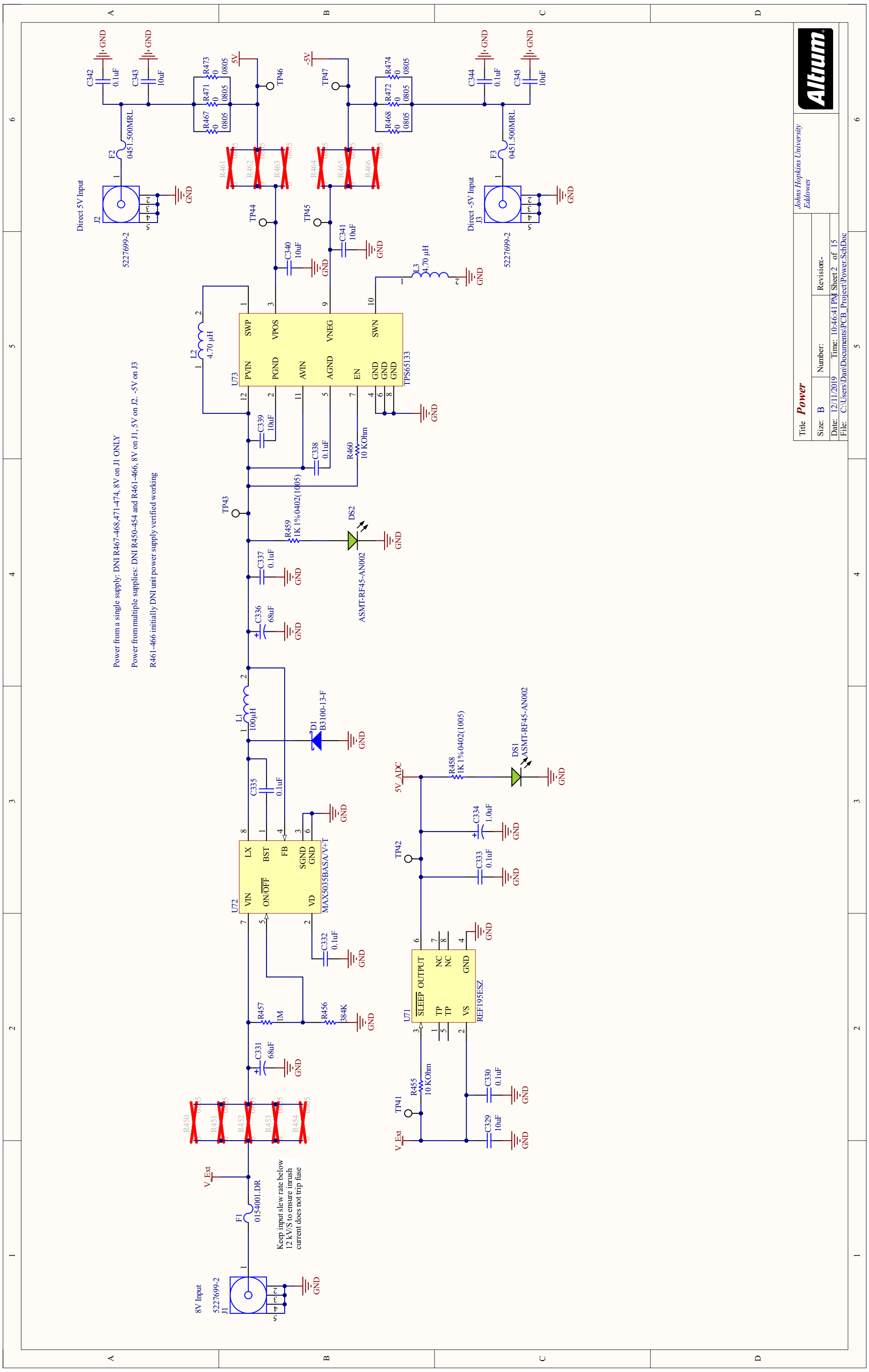
Future effort exploring the wavelet transform as a communication system demodulation method is possible. Higher order modulation schemes may be possible to demodulate with further simulation analysis. Higher frequency demodulation is also possible by selecting operational amplifiers with larger bandwidths and a faster sampling ADC. Increasing the op-amp/transistor bandwidth also allows for filters to respond faster, which will enable higher bit rate operations. Integrating the entire circuit into a high bandwidth transistor technology would provide the same high frequency, and high bit rate capabilities, while opening the door for other filter typologies to be used.

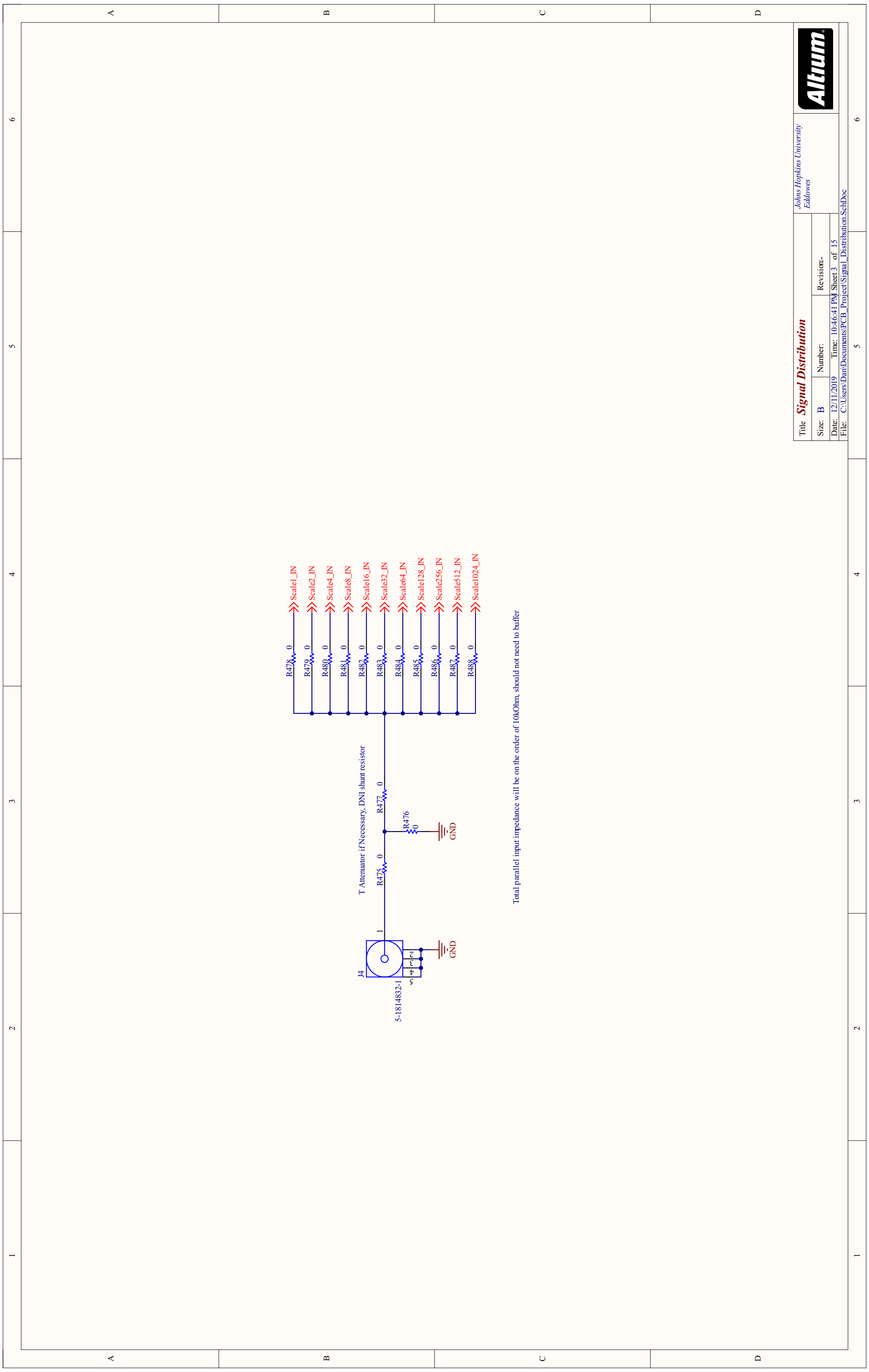
Appendix A

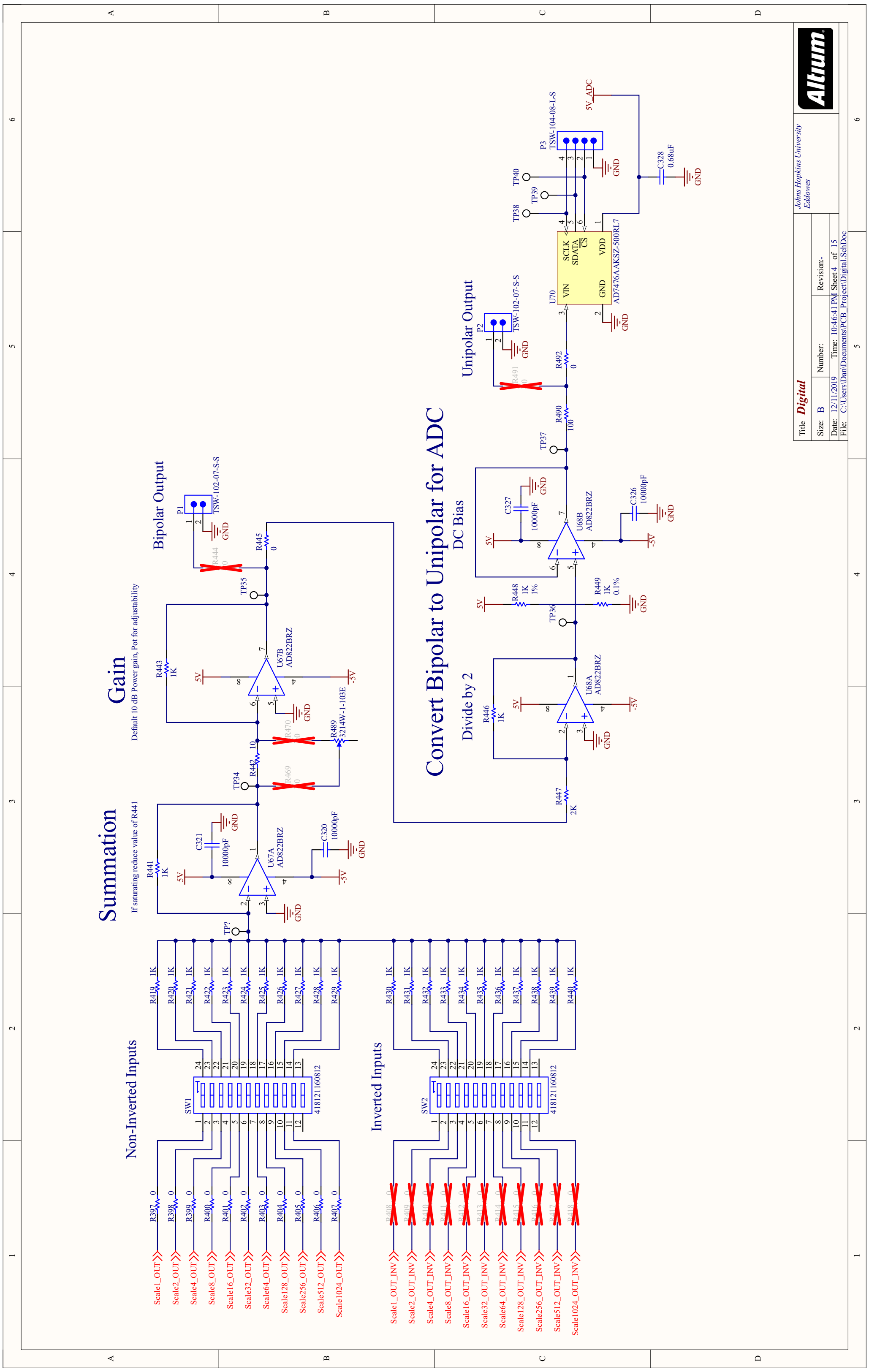
Schematics

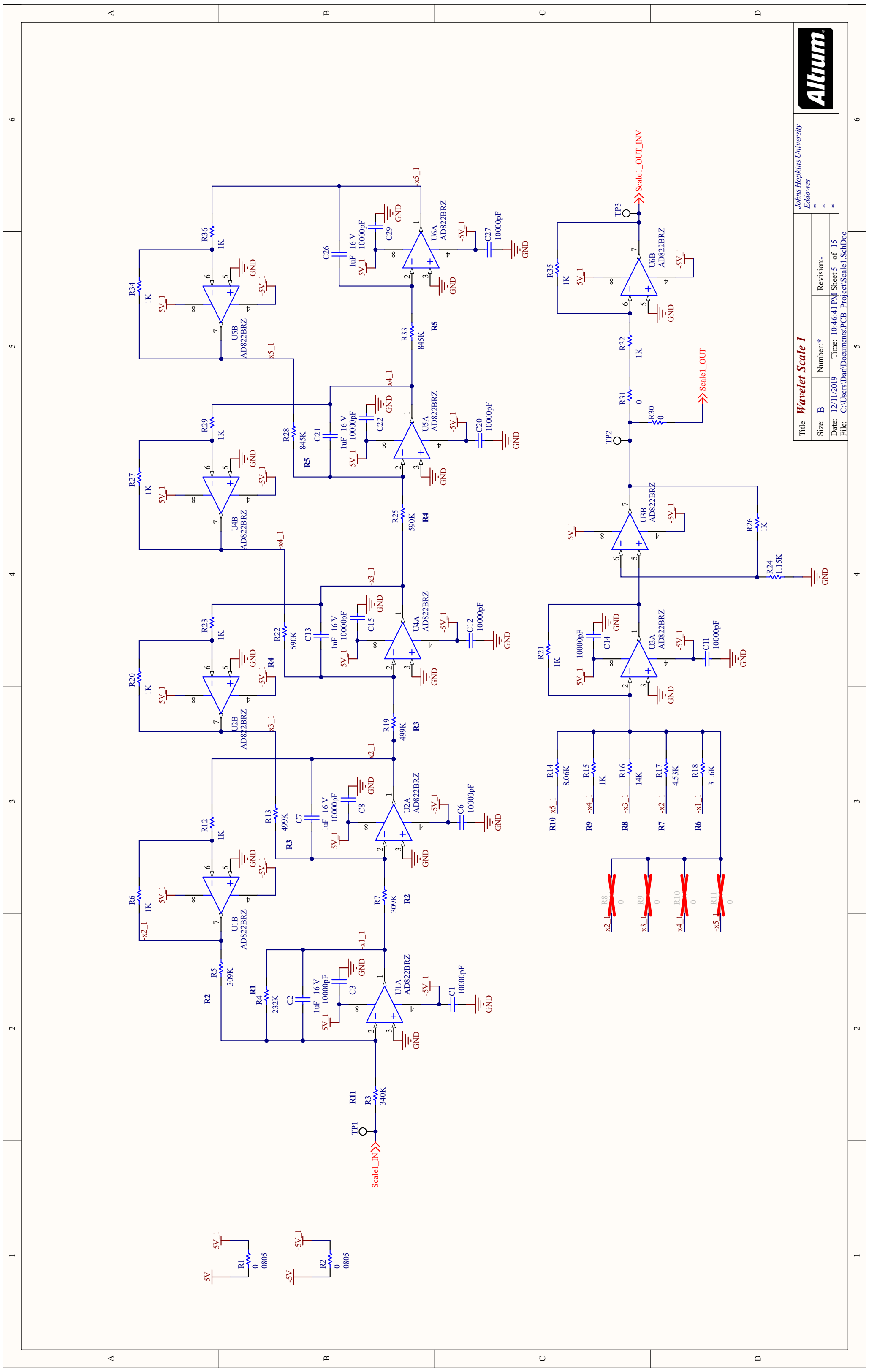
This appendix contains the entire schematic for the analog wavelets and support circuitry. The first page of the schematic contains a table of contents of the schematic pages.

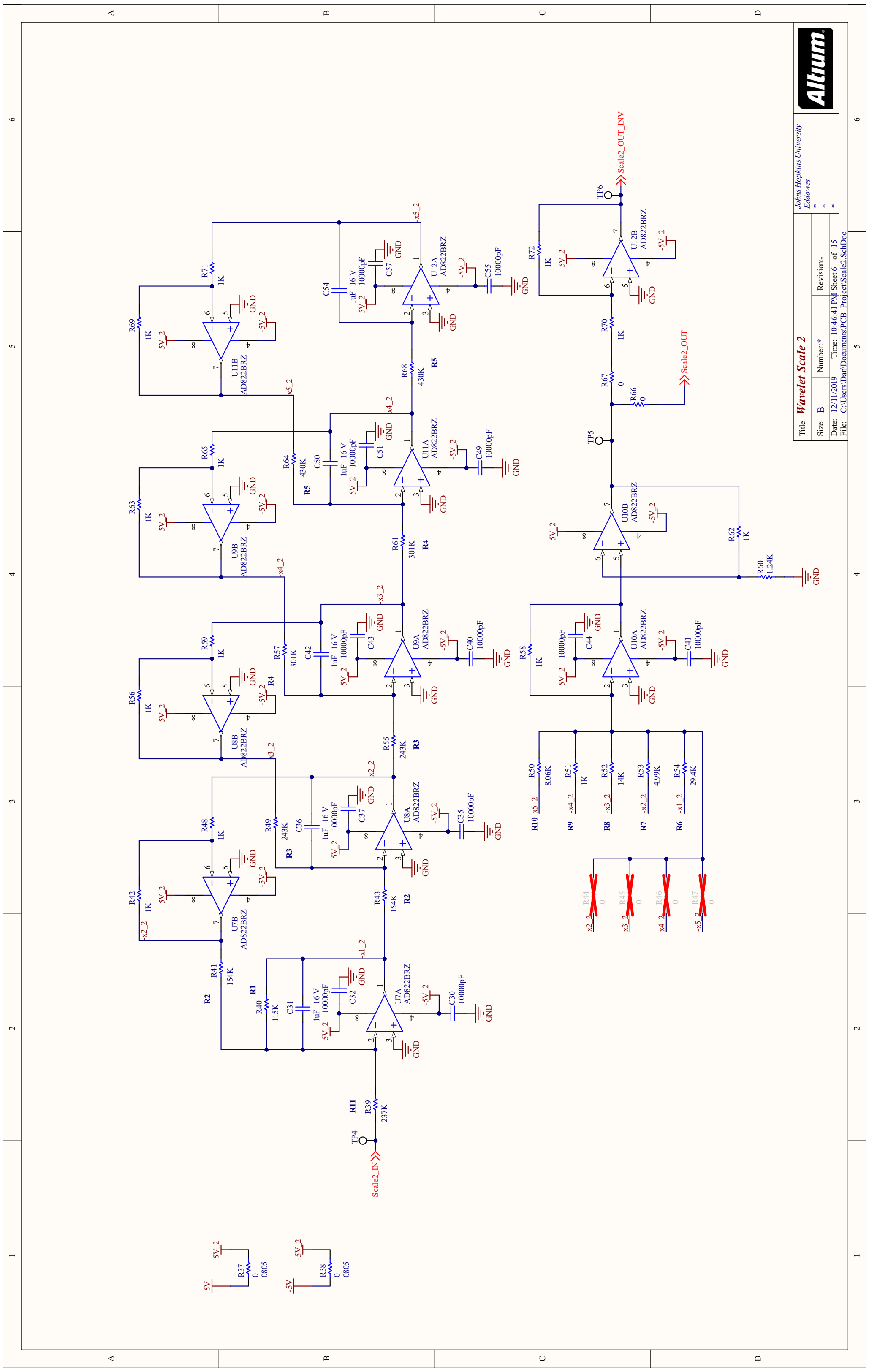

1) Table of Contents
2) Power
3) Signal Distribution
4) Digital
5) Wavelet Scale 1
6) Wavelet Scale 2
7) Wavelet Scale 4
8) Wavelet Scale 8
9) Wavelet Scale 16
10) Wavelet Scale 32
11) Wavelet Scale 64
12) Wavelet Scale 128
13) Wavelet Scale 256
14) Wavelet Scale 512
15) Wavelet Scale 1024
Title Table of Contents
Size: B
Number:
Revision:-
Date: 12/11/2019
Time: 10:46:41 PM
Sheet 1 of 15
File: C:\Users\Dan\Documents\PCB_Project\Table_of_Contents.SchDoc
Johns Hopkins University
Eddowes
A B C D
1 2 3 4 5 6

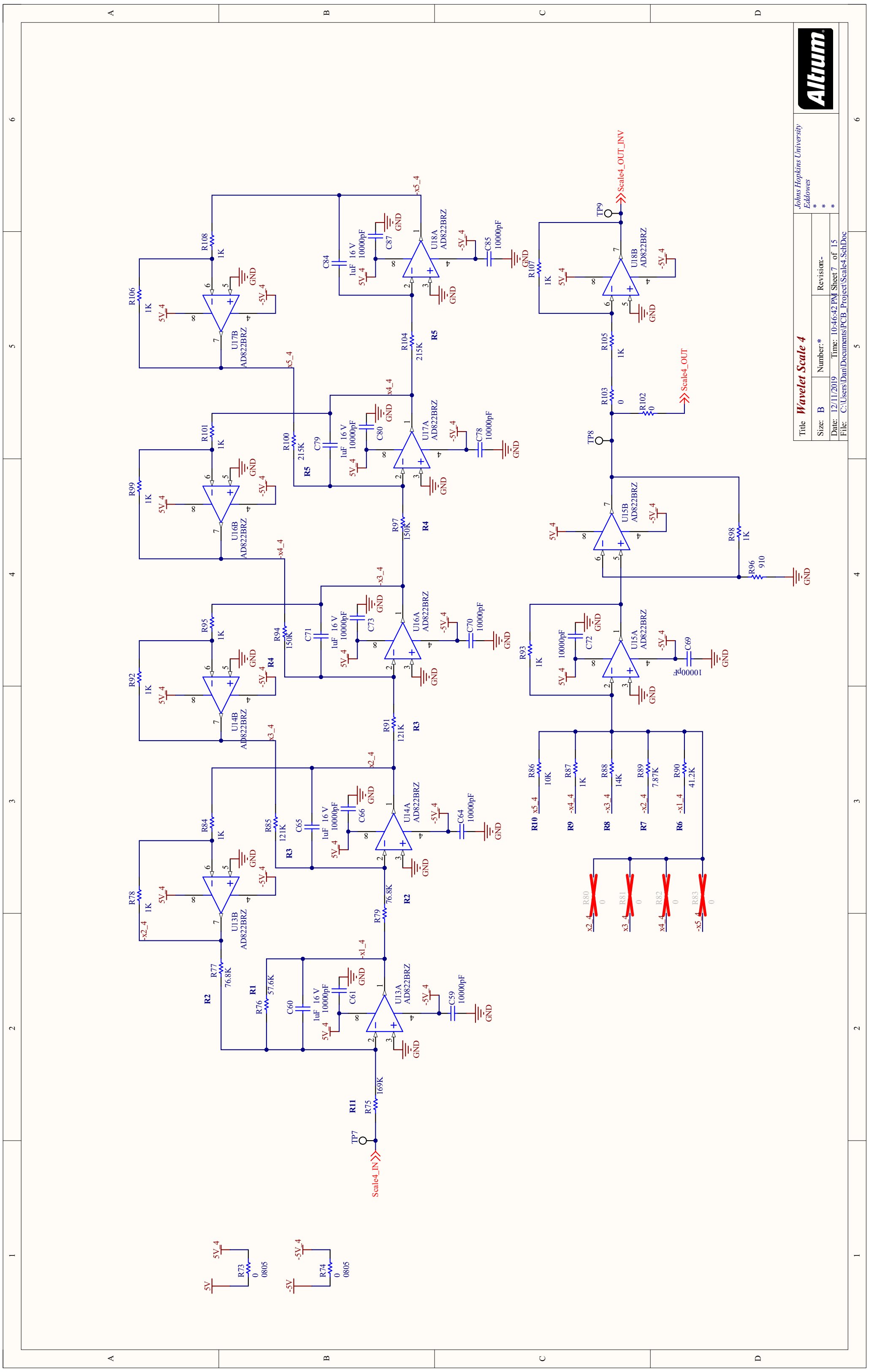


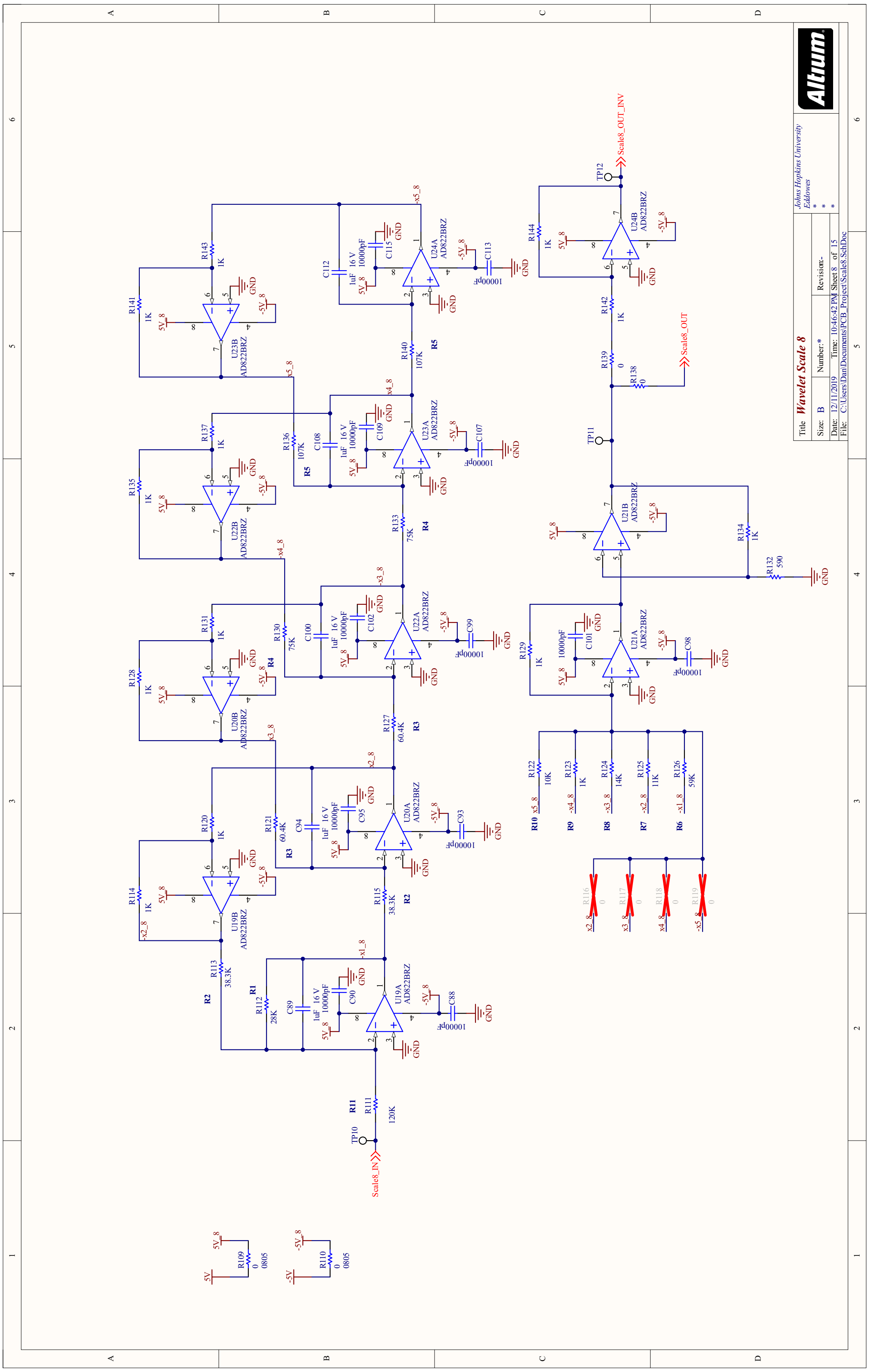

Title Signal Distribution
Size: B
Number:
Revision: -
Date: 12/11/2019
Time: 10:46:41 PM
Sheet 3 of 15
File: C:\Users\Dan\Documents\PCB_Project\Signal_Distribution.SchDoc
Johns Hopkins University
Eddowes
Altium
T Attenuator if Necessary, DNI shunt resistor
Total parallel input impedance will be on the order of 10kOhm, should not need to buffer
J4
5-1814832-1
R475 0
R476 0
R477 0
GND
Scale1_IN
Scale2_IN
Scale4_IN
Scale8_IN
Scale16_IN
Scale32_IN
Scale64_IN
Scale128_IN
Scale256_IN
Scale512_IN
Scale1024_IN
R478 0
R479 0
R480 0
R481 0
R482 0
R483 0
R484 0
R485 0
R486 0
R487 0
R488 0

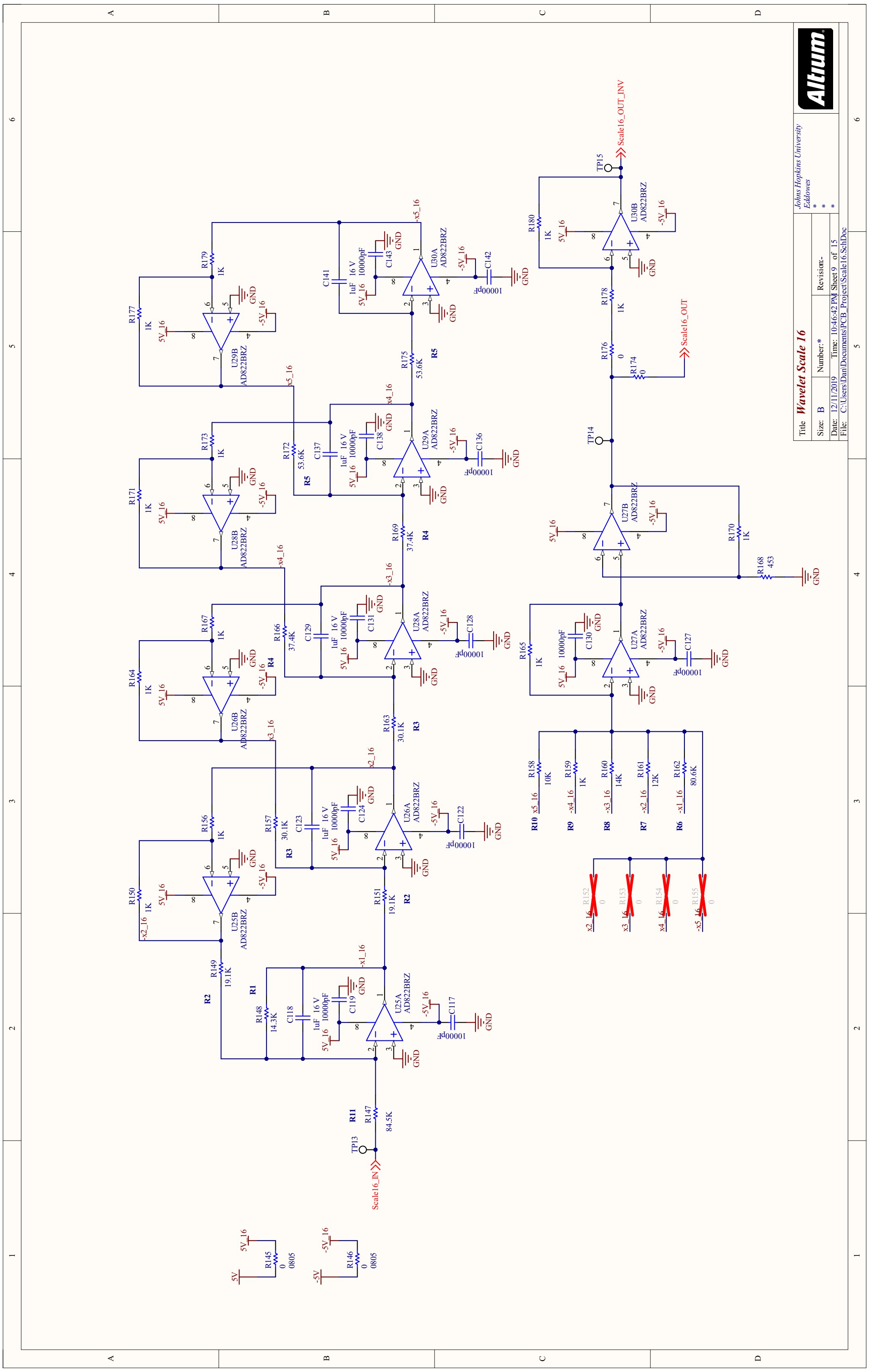

Non-Inverted Inputs
Inverted Inputs
Summation
If saturating reduce value of R441
Gain
Default 10 dB Power gain, Pot for adjustability
Bipolar Output
Convert Bipolar to Unipolar for ADC
Divide by 2
DC Bias
Unipolar Output
AD7476AAKSZ-500RL7
Title Digital
Johns Hopkins University
Eddowes
Date: 12/11/2019
Sheet 4 of 15

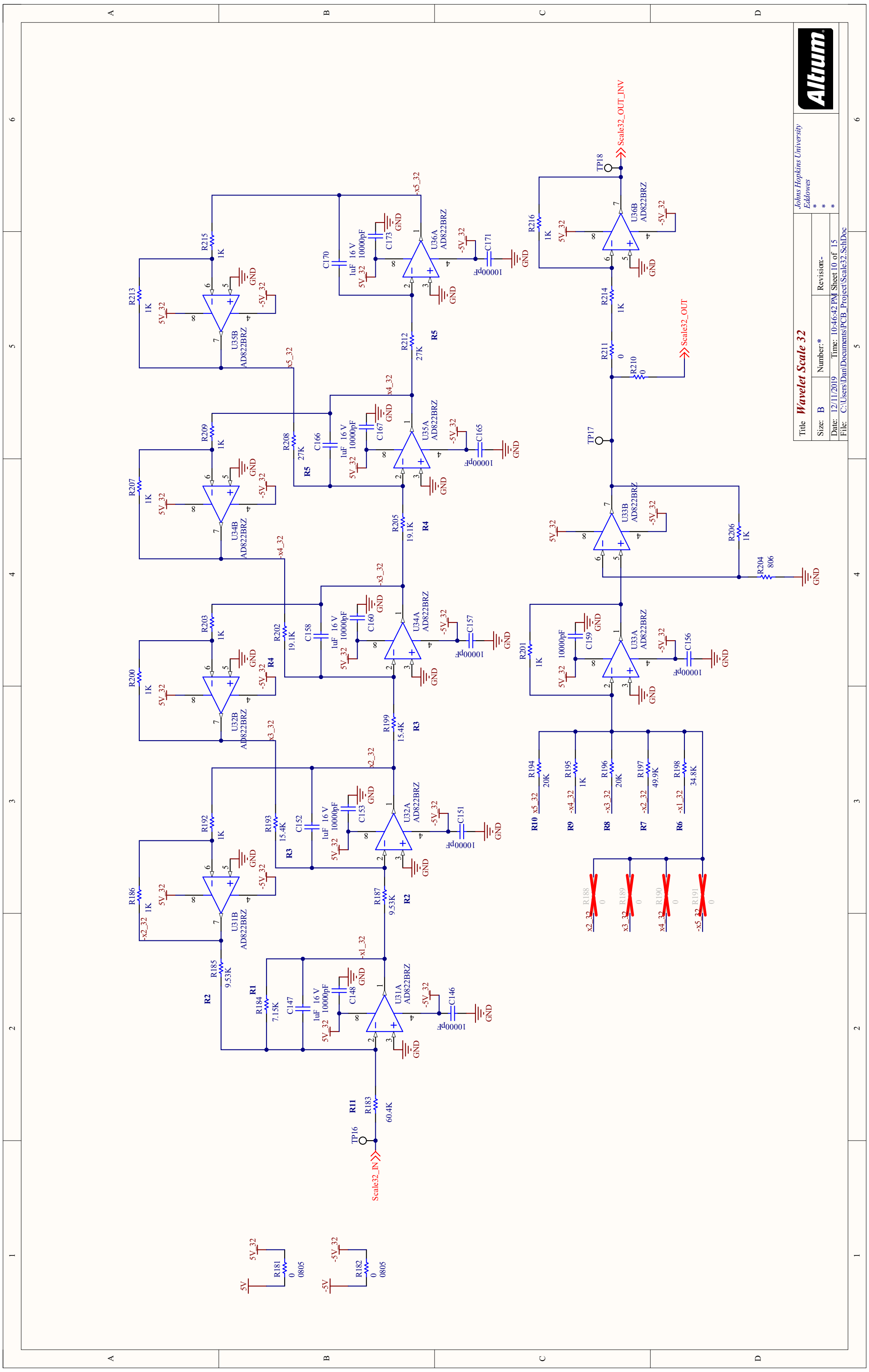

Wavelet Scale 1
Johns Hopkins University
Scale1_IN
Scale1_OUT
Scale1_OUT_INV

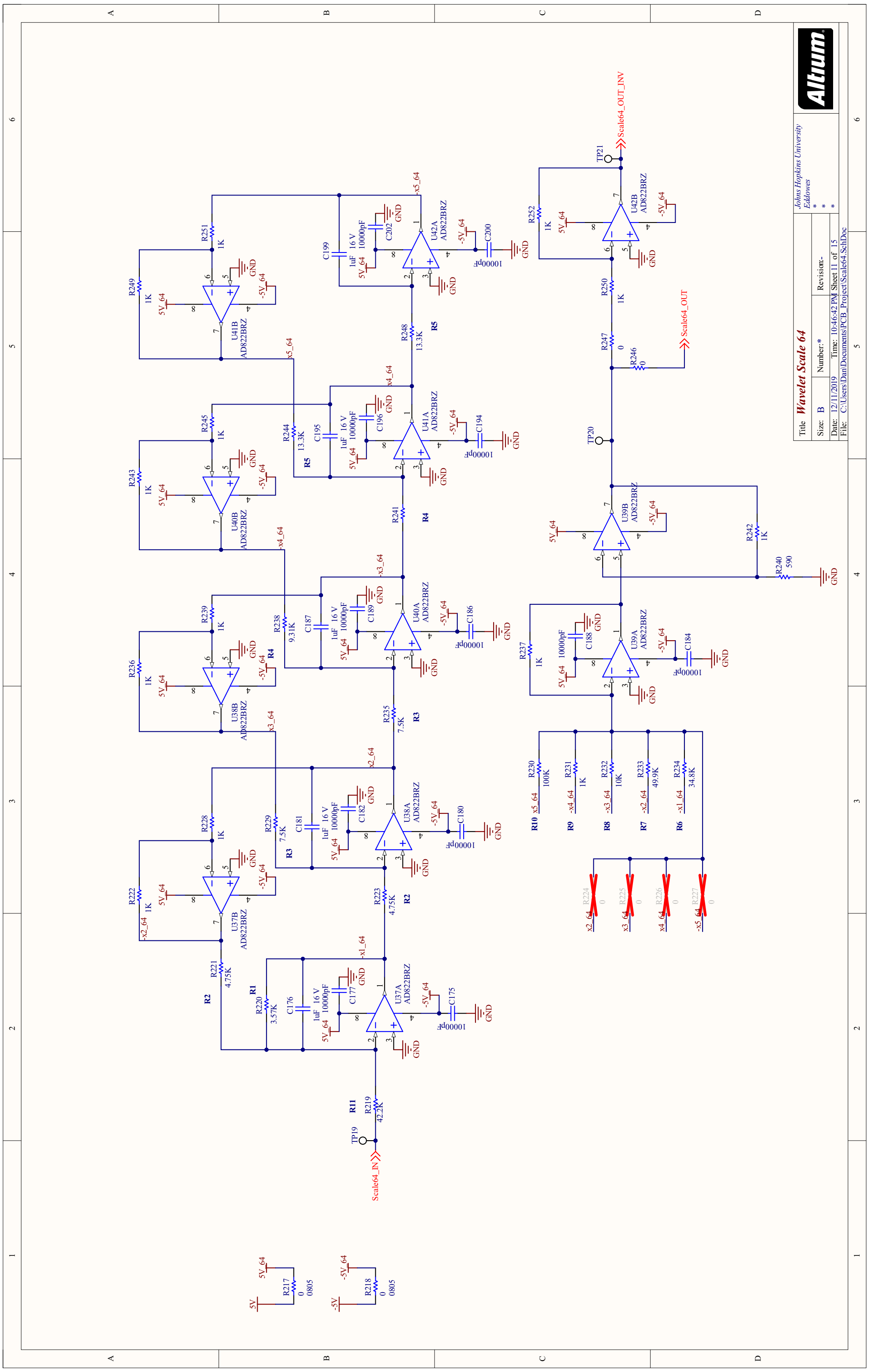


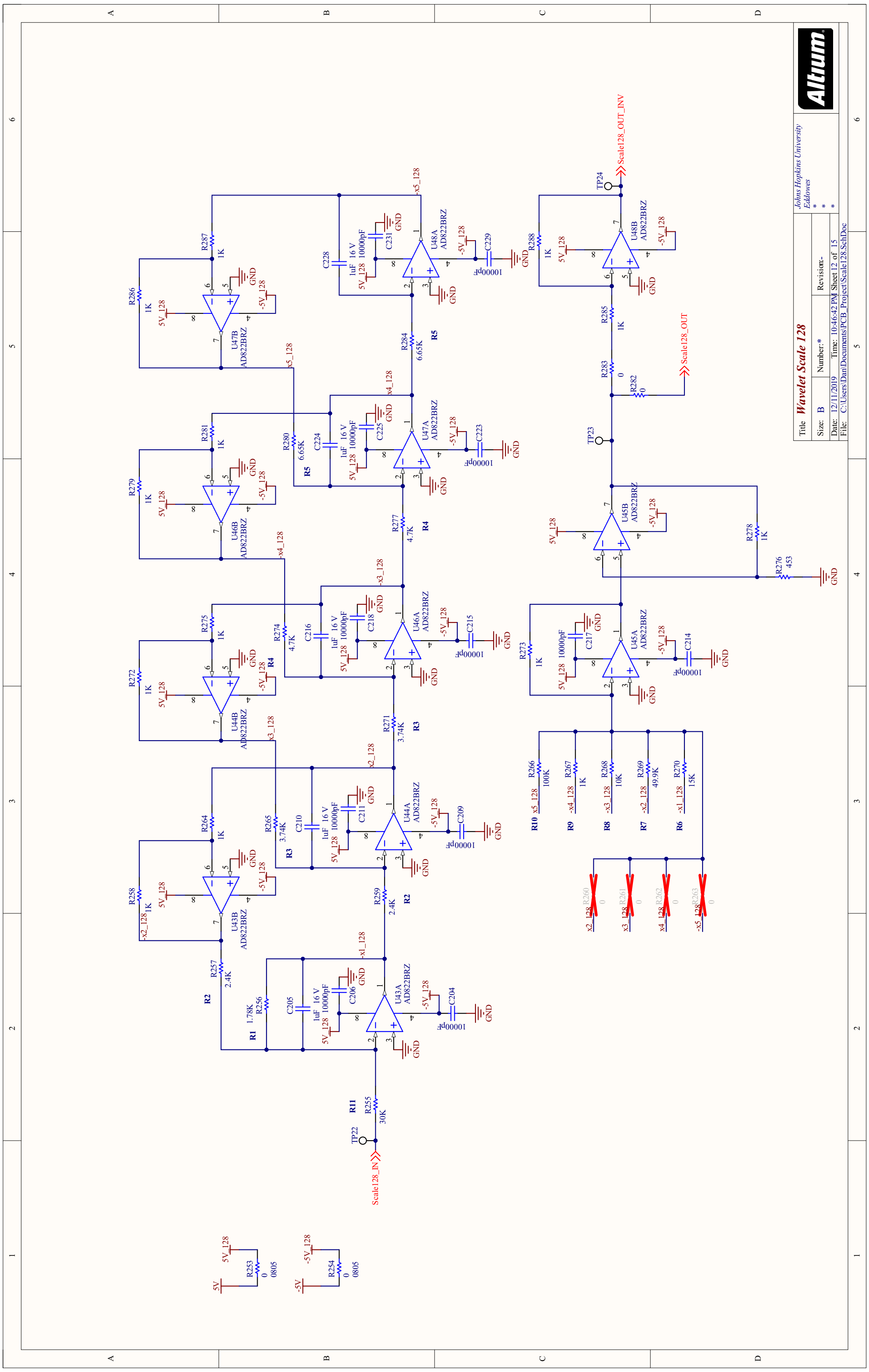

Title Wavelet Scale 4
Size: B
Number: *
Revision: -
Date: 12/11/2019
Time: 10:46:42 PM
Sheet 7 of 15
File: C:\Users\Dan\Documents\PCB_Project\Scale4.SchDoc
Johns Hopkins University
Eddowes
Altium
Scale4_IN
Scale4_OUT
Scale4_OUT_INV

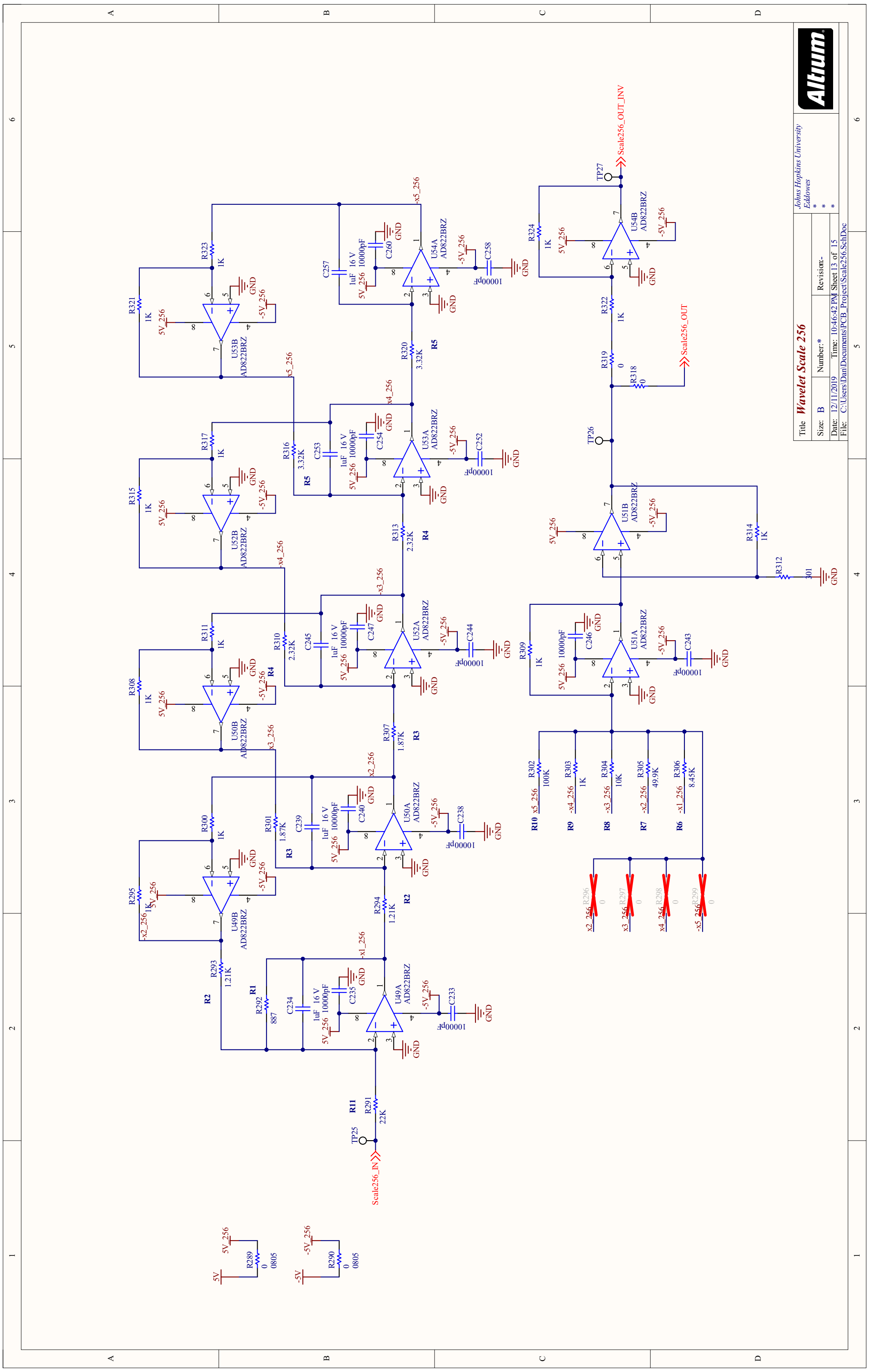


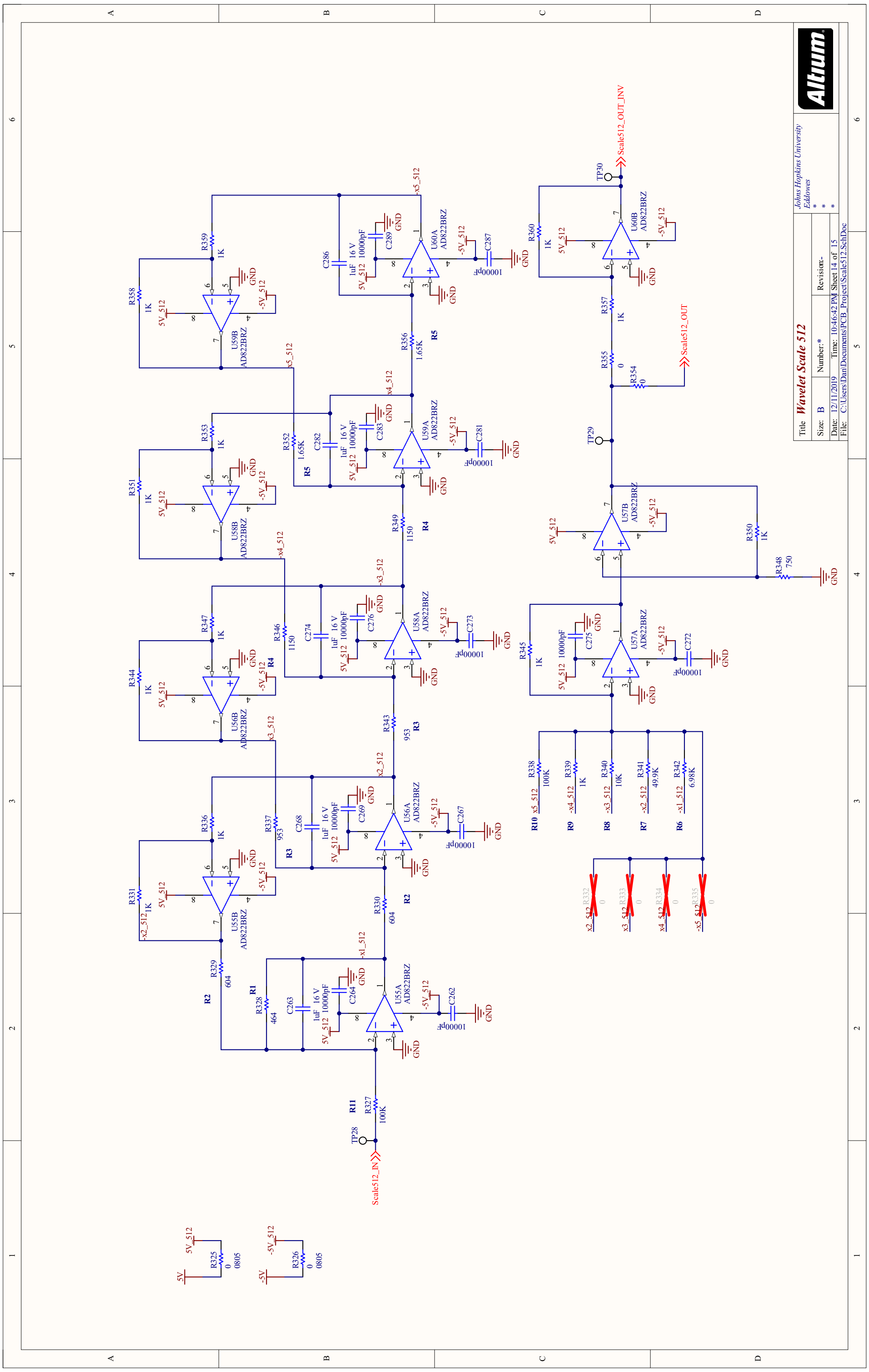

Wavelet Scale 16
Johns Hopkins University
Eddowes
Scale16_IN
Scale16_OUT
Scale16_OUT_INV

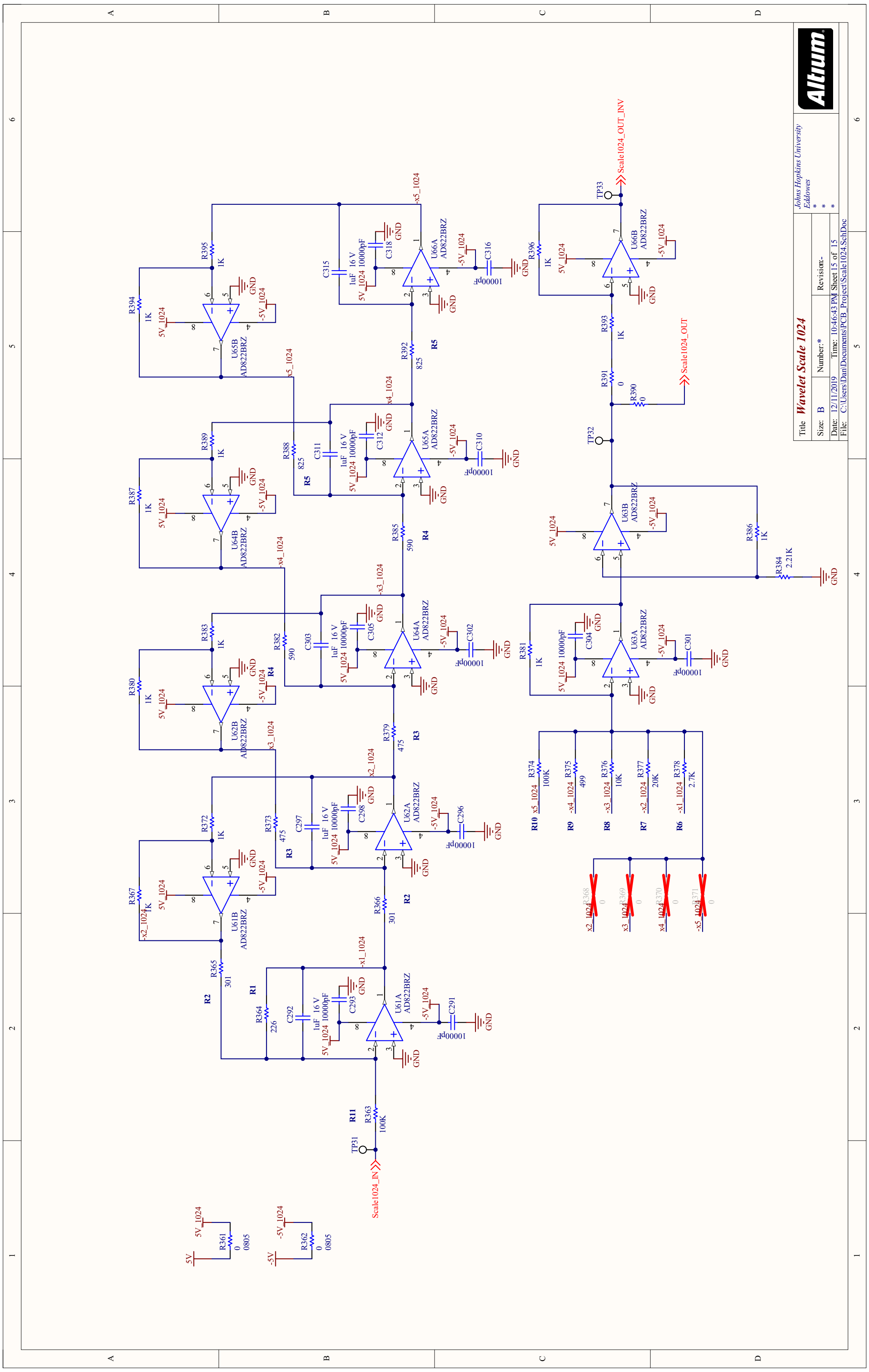



Title Wavelet Scale 64
Size: B
Number: *
Revision: -
Date: 12/11/2019
Time: 10:46:42 PM
Sheet 11 of 15
File: C:\Users\Dan\Documents\PCB_Project\Scale64.SchDoc
Johns Hopkins University
Eddowes
Altium


Altium
Johns Hopkins University
Eddowes
Title Wavelet Scale 128
Size: B
Number: *
Revision: -
Date: 12/11/2019
Time: 10:46:42 PM
Sheet 12 of 15
File: C:\Users\Dan\Documents\PCB_Project\Scale128.SchDoc
Scale128_IN
Scale128_OUT
Scale128_OUT_INV
AD822BRZ


Wavelet Scale 256
Johns Hopkins University
Eddowes
Date: 12/11/2019
Sheet 13 of 15
Scale256_IN
Scale256_OUT
Scale256_OUT_INV
AD822BRZ




Scale1024_IN
Scale1024_OUT
Scale1024_OUT_INV
Title Wavelet Scale 1024
Johns Hopkins University
Eddowes
Size: B
Number: *
Revision: -
Date: 12/11/2019
Time: 10:46:43 PM
Sheet 15 of 15
File: C:\Users\Dan\Documents\PCB_Project\Scale1024.SchDoc

Appendix B

Software

This appendix contains software written for the wavelet simulations and digitization.

B.1 MATLAB Code

MATLAB code is used for the wavelet simulation and numerical approximation, as well as an aid to support LTSpice simulations and BER functions. It includes custom written code and cited numerical approximation functions [17]. Additional functions, *optimize_SS.m*, *orth_ss.m*, *pade_approx.m*, and *schwarz-form.m* all support the numerical approximations and developing the final state space representations of the wavelets and are found in the literature [17].

Wavelet Simulation

```
% wavelet simulation using state space representation, ideal wavelet transform, and actual data with generated bpsk waveform
% eddowes 12/4/19

tic
close all
clear all

plotting = 1;   % plotting =1, generate output plots, not recommended for a large number of bits (> 10^4)
mode = 3;       % mode =1 for Pade, =2 for SVD, =3 for Circuit Sim, =4 for ideal transform from wavelet toolbox

fs = 1e7;               % sample rate
fc = 1e4;               % carrier frequency for bpsk waveform
fp = 1e3;       % bit rate
nbits = 1e1;    % number of bits to send

Ts = 1/fs;
Tp = 1/fp;
t_end = nbits*Tp;       % time of simulation
t = (0:Ts:t_end)';      % simulation time vector
SNR = 10;                         % snr to be used in simulation, snr=0 will set the noise power equal to the signal power

%%% bpsk waveform generation %%%
[data_stream,data,mod_noise,mod] = bpsk_gen(fs,fc,fp,nbits,SNR);

if plotting
    figure
    subplot(2,2,1)
    plot(t(1:end-1),data);       % plot bit stream
    grid on;
    title('Input Data Signal');
    xlabel('Time [s]');
    ylabel('Data [V]');

    subplot(2,2,2)
    plot(t(1:end-1),mod,'LineWidth',1.5);       % plot bpsk waveform
    grid on;
    title('Input BPSK Modulated Signal');
    xlabel('Time [s]');
    ylabel('Signal [V]');

    subplot(2,2,[3 4]);
    plot(t(1:end-1),mod_noise);          % plot bpsk waveform with noise
    title('Modulated Signal with Noise');
    xlabel('Time [s]');
    ylabel('Signal [V]');
end

```

```
disp('Bits Generation Complete');
toc

if plotting
    figure
end

wname = 'gaus1';
scales = [1,2,4,8,16,32,64,128,256,512,1024];   % wav scales used in
    simulation
t_len = 7;      % time length of first scale impulse response
ts = .005;      % time step in impulse response
y = zeros(length(t_len/ts),length(scales));
t_imp = zeros(length(t_len/ts),length(scales));

for i = 1:length(scales)
    t_vec = 0:ts:t_len/scales(i);

    if mode == 1                    % pade
        EQ = str2sym(['-2*(',num2str(scales(i)),'*t-1.7)*exp(-(',num2str
            (scales(i)),'*t-1.7)^2)']);      % symbolic representation of
             gaussian wavelet
        N = 2;  % numerator order
        D = 4;  % denominator order
        SS = wavelet_Pade_to_SS(EQ,N,D);
    elseif mode == 2     % svd
        F = (-2*(scales(i)*t_vec-1.8).*exp(-(scales(i)*t_vec-1.8).^2));
            % vector representing Gaussian wavelet
        [SS,Wc,Wo] = wavelet_SVD_to_SS(ts,t_vec,F);
    elseif mode == 3     % circuit simulation, gets data from .txt or .
        csv files
        fname = [pwd,'\Circuit Sims\',num2str(scales(i)),'.txt'];
            % file name containing Spice simulation or measured data
        M = readmatrix(fname);
        elseif mode == 4          % perform simulation with ideal wavelet
            transform from the MATLAB wavelet toolbox
                len = length(mod);
                CWTcoeffs = cwt(mod_noise,scales,wname,fs);
    end

        % find impulse response from state space representation
    if mode == 1 || mode ==2
        [y_temp,t_temp] = impulse(SS,t_vec);
    elseif mode == 3
        y_temp = M(:,2);
        t_temp = M(:,1);
    end

    for j = 1:length(t_temp)
        y(j,i) = y_temp(j);
        t_imp(j,i) = t_temp(j);
    end

        % plot impulse response
```

```
    if plotting
        plot(t_imp(:,i),y(:,i));
        hold on
        grid on
    end

    wavelet_CWT_approx(:,i) = conv(y(:,i),mod_noise);   % convolution, find the output of the wavelet transform
end

% plot wavelet output
wavelet_CWT_approx = wavelet_CWT_approx.';
if plotting
        if mode == 4    % special colormap plot using wavelet toolbox
                figure
                cwt(mod_noise,scales,wname,'plot');
                colormap jet; colorbar;
                hold on
                [cone,PL,PR,Pmin,Pmax] = conofinf(wname,scales,len,'plot');
                set(gca,'Xlim',[1 len])
        end

        x_lim_end = fs/100;

        figure
        for i = 1:length(scales)
                subplot(length(scales),1,i)
                plot(wavelet_CWT_approx(i,:));
                title(['Scale ',num2str(scales(i))]);
                xlim([0 x_lim_end])
                grid on;
        end
end

disp('Wavelet Transform Complete');
toc

%%% bit demodulation %%%
for i = 1:length(wavelet_CWT_approx)    % bit summation
   dec(i) = sum(wavelet_CWT_approx([9,10,11],i)); % can change what scales are used based on output of wavelet transform
end

if plotting
    figure
    plot(dec)   % plot summed wavelet output, for debugging purposes
    xlim([0 x_lim_end])
    grid on
end

thresh = 500;   % threshold for bit detection, dependent on convolution vector length
j = 1;
```

```
rate = Tp/Ts;
k = 0;
bit = 0;
n = 1;

% comparison to threshold, set demodulate bit based on exceeding pos or neg threshold
for i = 1:1:length(dec)
    if k == 0
        if dec(i) > thresh
            j = round(i/rate);
            bit(n:j) = 1;
            k = 1;
        elseif dec(i) < -thresh
            j = round(i/rate);
            bit(n:j) = 0;
            k = 1;
        end
        p = i;
    end

    if i == p+80        % after bit detection, ignore the next 80 samples, software debouncing
        k = 0;
        n = length(bit) + 1;
    end
end

% plot input bits, output bits, and demodulation errors
if plotting
    figure
    plot(bit,'o')
    hold on
    plot(data_stream,'o')
    plot(bit - data_stream(1:length(bit)))
    grid
end

err_per = sum(abs(bit - data_stream(1:length(bit))))/length(bit);    % calculate error percentage
fprintf('Decoding Complete. Error Percentage: %1.5f \n',err_per');
toc
```

BPSK Waveform Generation

```
function [data_stream,data,mod_noise,mod] = bpsk_gen(fs,fc,fp,nbits,SNR)
    % this function generates a random steam of n data bits
    % from these bits it then generates a BPSK waveform without noise
    % and with noise per the provides SNR.
    % fs is the sample frequency, fc is the carrier frequency, and fp is
```

```
        the bit rate

    Ts = 1/fs;
    Tp = 1/fp;
    t_end = nbits*Tp;
    t = (0:Ts:t_end)';
    A = 1;

    warning('off','MATLAB:colon:nonIntegerIndex');
    for j = 1:t_end/Tp
        data_stream(j) = randi([0 1]); % generate random bit stream
        data(((j-1)*Tp/Ts+1):(j*Tp/Ts)) = data_stream(j);
    end
    data = data.';
    warning('on','MATLAB:colon:nonIntegerIndex');

    for j = 2:length(data)-1   % prevent glitches, unsure why they are occurring
        if data(j) ~= data(j-1) && data(j) ~= data(j+1)
            data(j) = data(j+1);
        end
    end

    for j = 1:length(data)
        mod(j) = cos(2*pi*fc*t(j) + pi*(data(j)));     % create ideal bpsk waveform
    end

        % add noise to bpsk waveform
    mod_rms = 1/sqrt(2);
    noise_dev = mod_rms/(10^(SNR/10));
    mod_noise = mod + noise_dev.*randn(1,length(mod));
end
```

Pade to State Space Approximation

Sections of code taken from Grashuis [17]

```
function SSN = wavelet_Pade_to_SS(EQ,N,D)
    % this function generates a wavelet state-space representation using the Pade approximation
    % Refernce: M. Grashuis, A fully differential switched capacitor waveletfilter, Masters thesis, Daft University of Technology, 2009.

    sys_a = pade_approx(EQ, N, D); % the PADE approx.
    SS = orth_ss(sys_a);          % orthonomal approx

    [a,b,c,d,ts] = ssdata(SS);
    N = length(b);
    for t = 1:N
        if imag(b(t)) ~= 0
```

```
            b(t) = b(t) * 1i;
        end
        if imag(c(t)) ~= 0
            c(t) = c(t) / 1i;
        end
    end

    % noise scaling
    [Ks, Ws] = grams(SS);
    sumAr = sum(abs(a'));
    sumAc = sum(abs(a));
    alfaWsr = sumAr .* (diag(Ws) .* diag(Ks))';
    alfaWsc = sumAc .* (diag(Ws) .* diag(Ks))';
    Copt = sqrt(alfaWsr) ./ sum(sqrt(abs(alfaWsc)));
    SSN = ss(a, b, c, d, ts);
end
```

SVD to State Space Approximation

Sections of code taken from Grashuis [17]

```
function [SS,Wc,Wo] = wavelet_SVD_to_SS(ts,x,F)
    % this function generates a wavelet state-space representation using
        the SVD approximation
    % Refernce: M. Grashuis, A fully differential switched capacitor
        waveletfilter, Masters thesis, Daft University of
        Technology, 2009.

    r = zeros(1,length(x));     % column vector of zero's
    r(1,1) = F(1,1);    % first entry of r is first entry of F
    T = toeplitz(F,r);  % toeplitz matrix of F (lower triangular)
    H = hankel(T(:,1)); % hankel matrix of T
    [U S V] = svd(H);   % calculate the SVD

    % resize U, S and V according to the approximation.
    U = U(:, 1:1:5);
    S = S(1:1:5, 1:1:5);
    V = V(:, 1:1:5);
    H_appr = U*S*V'; % the new approximated hankel matrix.

    C = S^(1/2)*V';             % controllability matrix
    O = U*S^(1/2);              % observability matrix
    Ot = O(1:1:end-1, 1:1:end); % the upper part of the observability
    Ob = O(2:1:end, 1:1:end);   % the lower part of the observability
    a = Ot\Ob;                  % calculate the A matrix
    b = C(:,1);                 % calculate the B matrix
    c = O(1,:);                 % calculate the C matrix
    SS = ss(a,b,c,0,ts);        % create the State-space system
    SS = SS*ts;

```

```
    Wc = gram(SS,'c');  % controllability grammian
    Wo = gram(SS,'o');  % observability grammian
end
```

Circuit Value Calculator

```
%% generate resistor values for each scale
scales = [1,2,4,8,16,32,64,128,256,512,1024];
t_len = 7;
delay = 1.7;    % delay used  for Gaussian wavelet (can not be centered
    around zero)

for i=1:length(scales)
    % generate SVD approximation
    ts = .005/scales(i);
    t_vec = 0:ts:t_len/scales(i);
    F = (-2*(scales(i)*t_vec-delay).*exp(-(scales(i)*t_vec-delay).^2));
        % Gaus 1
    [SS,Wc,Wo] = wavelet_SVD_to_SS(ts,t_vec,F);
    SSc = d2c(SS);      % convert SS from discrete to continueous
        SSconical = canon(SSc,'companion');              % concial form
    SSc_sparse = schwarzform(SSconical);        % schwarz form
    SSc_sparse  % print SS

        % calculate resistor values, correspondence between R# and
            location in circuit in LTSpice
    div = 1;    % unused experimental factor
    C = 1e-6;   % capactitor value, 1uF
    R = 1000;   % feedback resistor value
    R1 = round(abs(1/(SSc_sparse.A(1,1)*C)));
    R2 = round(1/(SSc_sparse.A(2,1)*C));
    R3 = round(1/(SSc_sparse.A(3,2)*C));
    R4 = round(1/(SSc_sparse.A(4,3)*C));
    R5 = round(1/(SSc_sparse.A(5,4)*C));
    R6 = round(abs(R/SSc_sparse.C(1))/div);
    R7 = round(R/SSc_sparse.C(2)/div);
    R8 = round(abs(R/SSc_sparse.C(3))/div);
    R9 = round(R/SSc_sparse.C(4)/div);
    R10 = round(abs(R/SSc_sparse.C(5))/div);
    R11 = round(1/(SSc_sparse.B(1)*C));

    n(i) = log10(R7)/log10(i);

        % print LTSpice param line, used in circuit simulations
    param = ['.param C={1u} R1={',num2str(R1),'}  ','R2={',num2str(R2),'}
        ','R3={',num2str(R3),'}  ',...
      'R4={',num2str(R4),'}  ','R5={',num2str(R5),'}  ','R6={',num2str(R6),
        '}  ',...
```

```
        'R7={',num2str(R7),'} ','R8={',num2str(R8),'} ','R9={',num2str(R9),
           '} ',...
        'R10={',num2str(R10),'} ','R11={',num2str(R11),'} ','gain={800}'];
      clipboard('copy',param) % copy param line for easy pasting into
         LTSpice

          % print resistor values
          fprintf('%4.2f\n',R1)
          fprintf('%4.2f\n',R2)
          fprintf('%4.2f\n',R3)
          fprintf('%4.2f\n',R4)
          fprintf('%4.2f\n',R5)
          fprintf('%4.2f\n',R6)
          fprintf('%4.2f\n',R7)
          fprintf('%4.2f\n',R8)
          fprintf('%4.2f\n',R9)
          fprintf('%4.2f\n',R10)
          fprintf('%4.2f\n',R11)
end


%% reading values from resistor part list sheet, used for organization
    when generating all the scales
sheet='Sheet1';
cell = 'AH3:AH14';
M = xlsread('Resistor Parts List1.xlsx',sheet,cell);
param = ['.param C={1u} R1={',num2str(M(1)),'} ','R2={',num2str(M(2)),'}
       ','R3={',num2str(M(3)),'} ',...
        'R4={',num2str(M(4)),'} ','R5={',num2str(M(5)),'} ','R6={',num2str(
           M(6)),'} ',...
        'R7={',num2str(M(7)),'} ','R8={',num2str(M(8)),'} ','R9={',num2str(
           M(9)),'} ',...
        'R10={',num2str(M(10)),'} ','R11={',num2str(M(11)),'} ','gain={',
           num2str(M(12)),'} '];
clipboard('copy',param)
```

Measured BER Calculator

```
%% generate resistor values for each scale
scales = [1,2,4,8,16,32,64,128,256,512,1024];
t_len = 7;
delay = 1.7;     % delay used  for Gaussian wavelet (can not be centered
    around zero)

for i=1:length(scales)
    % generate SVD approximation
    ts = .005/scales(i);
    t_vec = 0:ts:t_len/scales(i);
    F = (-2*(scales(i)*t_vec-delay).*exp(-(scales(i)*t_vec-delay).^2));
       % Gaus 1
```

```
    [SS,Wc,Wo] = wavelet_SVD_to_SS(ts,t_vec,F);
    SSc = d2c(SS);      % convert SS from discrete to continueous
        SSconical = canon(SSc,'companion');           % concial form
    SSc_sparse = schwarzform(SSconical);        % schwarz form
    SSc_sparse  % print SS

        % calculate resistor values, correspondence between R# and location in circuit in LTSpice
    div = 1;    % unused experimental factor
    C = 1e-6;   % capactitor value, 1uF
    R = 1000;   % feedback resistor value
    R1 = round(abs(1/(SSc_sparse.A(1,1)*C)));
    R2 = round(1/(SSc_sparse.A(2,1)*C));
    R3 = round(1/(SSc_sparse.A(3,2)*C));
    R4 = round(1/(SSc_sparse.A(4,3)*C));
    R5 = round(1/(SSc_sparse.A(5,4)*C));
    R6 = round(abs(R/SSc_sparse.C(1))/div);
    R7 = round(R/SSc_sparse.C(2)/div);
    R8 = round(abs(R/SSc_sparse.C(3))/div);
    R9 = round(R/SSc_sparse.C(4)/div);
    R10 = round(abs(R/SSc_sparse.C(5))/div);
    R11 = round(1/(SSc_sparse.B(1)*C));

    n(i) = log10(R7)/log10(i);

        % print LTSpice param line, used in circuit simulations
    param = ['.param C={1u} R1={',num2str(R1),'}  ','R2={',num2str(R2),'}  ','R3={',num2str(R3),'}  ',...
      'R4={',num2str(R4),'}  ','R5={',num2str(R5),'}  ','R6={',num2str(R6),'}  ',...
      'R7={',num2str(R7),'}  ','R8={',num2str(R8),'}  ','R9={',num2str(R9),'}  ',...
      'R10={',num2str(R10),'}  ','R11={',num2str(R11),'}  ','gain={800}'];
    clipboard('copy',param) % copy param line for easy pasting into LTSpice

        % print resistor values
        fprintf('%4.2f\n',R1)
        fprintf('%4.2f\n',R2)
        fprintf('%4.2f\n',R3)
        fprintf('%4.2f\n',R4)
        fprintf('%4.2f\n',R5)
        fprintf('%4.2f\n',R6)
        fprintf('%4.2f\n',R7)
        fprintf('%4.2f\n',R8)
        fprintf('%4.2f\n',R9)
        fprintf('%4.2f\n',R10)
        fprintf('%4.2f\n',R11)
end


%% reading values from resistor part list sheet, used for organization when generating all the scales
```

```
sheet='Sheet1';
cell = 'AH3:AH14';
M = xlsread('Resistor Parts List1.xlsx',sheet,cell);
param = ['.param C={1u} R1={',num2str(M(1)),'} ','R2={',num2str(M(2)),'}
    ','R3={',num2str(M(3)),'} ',...
      'R4={',num2str(M(4)),'} ','R5={',num2str(M(5)),'} ','R6={',num2str(
         M(6)),'} ',...
      'R7={',num2str(M(7)),'} ','R8={',num2str(M(8)),'} ','R9={',num2str(
         M(9)),'} ',...
      'R10={',num2str(M(10)),'} ','R11={',num2str(M(11)),'} ','gain={',
         num2str(M(12)),'} '];
clipboard('copy',param)
```

B.2 Embedded C Code

The following embedded C code is used to control the wavelet digitization and threshold demodulation. The code is targeted to a SAMD21 processor on an Arduino MKR Zero. Microchip (formerly Atmel) and Arduino forums used in designing code [25,26].

Digitization Code

```
// C program for SAMD21 processor on Arudino Mkr Zero
// Takes ADC input, uses DMA controller to access ADC data, main CPU
   computes bits off
// ADC data and sends to PC over serial connection
// Reference http://www.atmel.com/Images/Atmel-42258-ASF-Manual-SAM-
   D21_AP-Note_AT07627.pdf pg 73
// Reference MartinL at https://forum.arduino.cc/index.php?topic
   =518461.0

#define ADCPIN A1              // pin A1 on Arudino Mkr Zero
#define HWORDS 4               // number of samples the ADC takes per DMA
   access
uint16_t adcbuf[HWORDS];

typedef struct {          // DMA structure
    uint16_t btctrl;
    uint16_t btcnt;
```

```
    uint32_t srcaddr;   // source address
    uint32_t dstaddr;   // data address
    uint32_t descaddr;  // destination address
} dmacdescriptor ;

volatile dmacdescriptor wrb[12] __attribute__ ((aligned (16)));
dmacdescriptor descriptor_section[12] __attribute__ ((aligned (16)));
dmacdescriptor descriptor __attribute__ ((aligned (16)));

static uint32_t chnl = 0;    // DMA channel
volatile uint32_t dmadone;  // DMA transfer complete flag

void DMAC_Handler() {    // DMA interrupt handler
    uint8_t active_channel;

    __disable_irq();     // disable interrupts
    active_channel =  DMAC->INTPEND.reg & DMAC_INTPEND_ID_Msk; // get channel number
    DMAC->CHID.reg = DMAC_CHID_ID(active_channel);
    dmadone = DMAC->CHINTFLAG.reg;   // check if DMA transfer is complete
    DMAC->CHINTFLAG.reg = DMAC_CHINTENCLR_TCMPL;
    DMAC->CHINTFLAG.reg = DMAC_CHINTENCLR_TERR;
    DMAC->CHINTFLAG.reg = DMAC_CHINTENCLR_SUSP;
    __enable_irq();      // enable interrupts
}

void dma_init() {    // DMA initialization
    PM->AHBMASK.reg |= PM_AHBMASK_DMAC ;
    PM->APBBMASK.reg |= PM_APBBMASK_DMAC ;
    NVIC_EnableIRQ( DMAC_IRQn ) ;

    DMAC->BASEADDR.reg = (uint32_t)descriptor_section;
    DMAC->WRBADDR.reg = (uint32_t)wrb;
    DMAC->CTRL.reg = DMAC_CTRL_DMAENABLE | DMAC_CTRL_LVLEN(0xf);
}

void adc_dma(void *rxdata,  size_t hwords) {
    uint32_t temp_CHCTRLB_reg;

    DMAC->CHID.reg = DMAC_CHID_ID(chnl);
    DMAC->CHCTRLA.reg &= ~DMAC_CHCTRLA_ENABLE;   // enable DMA
    DMAC->CHCTRLA.reg = DMAC_CHCTRLA_SWRST;
    DMAC->SWTRIGCTRL.reg &= (uint32_t)(~(1 << chnl));
    temp_CHCTRLB_reg = DMAC_CHCTRLB_LVL(0) |
      DMAC_CHCTRLB_TRIGSRC(ADC_DMAC_ID_RESRDY) | DMAC_CHCTRLB_TRIGACT_BEAT;
    DMAC->CHCTRLB.reg = temp_CHCTRLB_reg;
    DMAC->CHINTENSET.reg = DMAC_CHINTENSET_MASK ; // enable DMA interrupts
    dmadone = 0;
    descriptor.descaddr = 0;
    descriptor.srcaddr = (uint32_t) &ADC->RESULT.reg;    // source address is ADC result
    descriptor.btcnt =  hwords;
```

```
    descriptor.dstaddr = (uint32_t)rxdata + hwords*2;   // end address
    descriptor.btctrl =  DMAC_BTCTRL_BEATSIZE_HWORD | DMAC_BTCTRL_DSTINC | DMAC_BTCTRL_VALID;
    memcpy(&descriptor_section[chnl],&descriptor, sizeof(dmacdescriptor));

    // block of code below prevents hangups where DMA flag is never set
    while (ADC->INTFLAG.bit.RESRDY == 0);
    uint16_t value = ADC->RESULT.reg;
    ADCsync();

    // start channel
    DMAC->CHID.reg = DMAC_CHID_ID(chnl);
    DMAC->CHCTRLA.reg |= DMAC_CHCTRLA_ENABLE;
}

static __inline__ void ADCsync() __attribute__((always_inline, unused));
static void   ADCsync() {
  while (ADC->STATUS.bit.SYNCBUSY == 1); //wait till the ADC is free
}

void adc_init(){
  analogRead(ADCPIN);                   // initialize ADC input pin
  ADC->CTRLA.bit.ENABLE = 0x00;         // Disable ADC
  ADCsync();
  ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_DIV2_Val;    // divide ADC input by 2
  ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_AREFA_Val;   // use analog reference A
  ADCsync();    //  ref 31.6.16
  ADC->INPUTCTRL.bit.MUXPOS = g_APinDescription[ADCPIN].ulADCChannelNumber;
  ADCsync();
  ADC->AVGCTRL.reg = 0x00 ; //no averaging
  ADC->SAMPCTRL.reg = 0x00; //sample length in 1/2 CLK_ADC cycles
  ADCsync();
  ADC->CTRLB.reg = ADC_CTRLB_PRESCALER_DIV32 | ADC_CTRLB_FREERUN | ADC_CTRLB_RESSEL_12BIT;  // free running, clk div 32, 12 bit output
  ADCsync();
  ADC->CTRLA.bit.ENABLE = 0x01;         // enable ADC
  ADCsync();
}

void setup(){
  Serial.begin(500000); // 500 kbaud serial interface
  adc_init();
  dma_init();
}

void loop() {   // arduino equivalent of while(1) inside main{}
  uint16_t val;
  uint16_t thresh_l = 750; //650;
  uint16_t thresh_h = 980; //1050;

```

```
  adc_dma(adcbuf,HWORDS);    // get ADC samples
  while(!dmadone);  // wait till DMA transfer is done
  val = (adcbuf[0] + adcbuf[1] + adcbuf[2] + adcbuf[3])/4;
  //Serial.println(val);

  if (val < thresh_l) {
    Serial.println(0);  // send bit zero if less than threshold
    delay(10);          // delay 10 ms, software pause to prevent false
    positives
  }
  else if (val > thresh_h) {
    Serial.println(1);  // send bit one if greater than threshold
    delay(10);
  }
}
```

Appendix C

Wavelet Resistor Selection

This appendix contains a table of all resistor values used in the analog wavelet implementations. R1 through R12 refer to the resistors as noted in figure 4.4. All resistors used for the wavelet scales are from the Panasonic ERJ-2RKF series. All resistors have an 0402 footprint and a 1% tolerance. Table C.1 below contains the values:

Table C.1: This table contains all resistor values (in Ohms) used in each analog wavelet scale.

Scale	1	2	4	8	16	32	64	128	256	512	1024
R1	232000	115000	57600	28000	14300	7150	3570	1780	887	464	226
R2	309000	154000	76800	38300	19100	9530	4750	2400	1210	604	301
R3	499000	243000	121000	60400	30100	15400	7500	3740	1870	953	475
R4	590000	301000	150000	75000	37400	19100	9310	4700	2320	1150	590
R5	845000	430000	215000	107000	53600	27000	13300	6650	3320	1650	825
R6	31600	29400	41200	59000	80600	34800	34800	15000	8450	6980	2700
R7	4530	4990	7870	11000	12000	49900	49900	49900	49900	49900	20000
R8	14000	14000	14000	14000	14000	20000	10000	10000	10000	10000	10000
R9	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	499
R10	8060	8060	10000	10000	10000	20000	100000	100000	100000	100000	100000
R11	340000	237000	169000	120000	84500	60400	42200	30000	22000	100000	100000
R12	1150	1240	910	590	453	806	590	453	301	750	2210

Appendix D

Bill of Materials

This appendix contains the bill of materials for the entire project, including parts, PCB fabrication, and assembly equipment. Table D.1 below summarizes the total cost of the project:

Item	**Cost**	**Note**
Printed Circuit Board Fabrication	$66	Cost per PCB. Student program at Advance Circuits used for PCB fabrication.
Assembly Cost	$82.32	Hand assembled. Cost listed is for solder paste and stencil
Parts	$580.39	Includes passive parts overage. Part cost is about $470 without overages.
TOTAL	$728.71	Four PCBs were ordered and extra solder paste was purchased, total amount spent: $942.71

Table D.1: Cost Breakdown

Table D.2 below contains the electrical bill of materials:

APPENDIX D. BILL OF MATERIALS

Comp	Part Number	Designator	Value	Mfr.	Qty.
Capacitor	GRM155R71E103JA01J	C1, C3, C6, C8, C11, C12, C14, C15, C20, C22, C27, C29, C30, C32, C35, C37, C40, C41, C43, C44, C49, C51, C55, C57, C59, C61, C64, C66, C69, C70, C72, C73, C78, C80, C85, C87, C88, C90, C93, C95, C98, C99, C101, C102, C107, C109, C113, C115, C117, C119, C122, C124, C127, C128, C130, C131, C136, C138, C142, C143, C146, C148, C151, C153, C156, C157, C159, C160, C165, C167, C171, C173, C175, C177, C180, C182, C184, C186, C188, C189, C194, C196, C200, C202, C204, C206, C209, C211, C214, C215, C217, C218, C223, C225, C229, C231, C233, C235, C238, C240, C243, C244, C246, C247, C252, C254, C258, C260, C262, C264, C267, C269, C272, C273, C275, C276, C281, C283, C287, C289, C291, C293, C296, C298, C301, C302, C304, C305, C310, C312, C316, C318, C320, C321, C326, C327, C346	10000pF	Murata	137
Capacitor	CC0603JRX7R6BB105	C2, C7, C13, C21, C26, C31, C36, C42, C50, C54, C60, C65, C71, C79, C84,C89, C94, C100, C108, C112, C118, C123, C129, C137, C141, C147, C152, C158, C166, C170, C176, C181, C187, C195, C199, C205, C210, C216, C224, C228, C234, C239, C245, C253, C257, C263, C268, C274, C282, C286, C292, C297, C303, C311, C315	1uF	Yageo	55
Capacitor	C0805C684K4RACTU	C328	0.68uF	Kemet	1
Capacitor	C0805C106K4PACTU	C329, C339,	10uF	Kemet	6
Capacitor	C0603C104J4RACTU	C330, C333,	0.1uF	Kemet	4
Capacitor	TPSE686K020R0150	C331, C336	68uF	AVX	2
Capacitor	C0805C104Z5VACTU	C332, C335,	0.1uF	Kemet	4
Capacitor	T491A105K016AT	C334		Kemet	1
Diode	B3100-13-F	D1		Diodes Incorporated	1
LED	ASMT-RF45-AN002	DS1, DS2		Avago	2
Fuse	0154001.DR	F1		Littelfuse	1
Fuse	0451.500MRL	F2, F3		Littelfuse	2
Connector	5227699-2	J1, J2, J3		TE Connectivity	3
Connector	5-1814832-1	J4		TE Connectivity	1
Inductor	7445720	L1		Wirewound	1
Inductor	744031004	L2, L3		Wirewound	2
Pin	TSW-102-07-S-S	P1, P2		Samtec	2
Pin	TSW-104-08-L-S	P3		Samtec	1

Resistor	CRCW08050000Z0EAHP	R1, R2, R37, R38, R73, R74, R109, R110, R145, R146, R181, R182, R217, R218, R253, R254, R289, R290, R325, R326, R361, R362, R450, R451, R452, R453, R454, R461, R462, R463, R464, R465, R466, R467, R468, R471, R472, R473, R474, R493	0	Vishay Dale	40
Resistor	ERJ-2RKF3403X	R3	340K	Panasonic	1
Resistor	ERJ-2RKF2323X	R4	232K	Panasonic	1
Resistor	ERJ-2RKF1001X	R6, R12, R15, R20, R21, R23, R26, R27, R29, R32, R34, R35, R36, R42, R48, R51, R56, R57, R59, R62, R63, R65, R68, R70, R71, R72, R78, R84, R92, R93, R95, R98, R99, R101, R104, R106, R107, R108, R114, R120, R128, R129, R131, R134, R135, R137, R140, R142, R143, R144, R150, R156, R164, R165, R167, R170, R171, R173, R176, R178, R179, R180, R186, R192, R200, R201, R203, R206, R207,R209, R212, R214, R215, R216, R222, R228, R236, R237, R239, R242, R243, R245, R248, R250, R251, R252, R258, R264, R272, R273, R275, R278, R279, R281, R284, R286, R287, R288, R295, R300, R308, R309, R311, R314, R315, R317, R320, R322, R323, R324, R330, R336, R344, R345, R347, R350, R351, R353, R356, R358, R359, R360, R366, R372, R380, R381, R383, R386, R387, R389, R392, R394, R395, R396, R419, R420, R421, R422, R423, R424, R425, R426, R427, R428, R429, R430, R431, R432, R433, R434, R435, R436, R437, R438, R439, R440, R441, R443, R446, R458, R459	1K	Panasonic	161
Resistor	RC0402JR-070RL	R8, R9, R10, R11, R30, R31, R44, R45, R46, R47, R66, R67, R80, R81, R82, R83, R102, R103, R116, R117, R118, R119,R138, R139, R152, R153, R154, R155, R174, R175, R188, R189, R190, R191, R210, R211, R224, R225, R226, R227, R246, R247, R260, R261, R262, R263, R282, R283, R296, R297, R298, R299, R318, R319, R332, R333, R334, R335, R354, R355, R368, R369, R370, R371, R390, R391	0	Panasonic	66
Resistor	ERJ-2RKF3093X	R5, R7	309K	Panasonic	2
Resistor	ERJ-2RKF4993X	R13, R19	499K	Panasonic	2
Resistor	ERJ-2RKF8061X	R14, R50	8.06K	Panasonic	2
Resistor	ERJ-2RKF1402X	R16, R52, R	14K	Panasonic	5

Resistor	ERJ-2RKF4531X	R17	4.53K	Panasonic	1
Resistor	ERJ-2RKF3162X	R18	31.6K	Panasonic	1
Resistor	ERJ-2RKF5903X	R22, R25	590K	Panasonic	2
Resistor	ERJ-2RKF1051X	R24	1.15K	Panasonic	1
Resistor	ERJ-2RKF8453X	R28, R33	845K	Panasonic	2
Resistor	ERJ-2RKF2373X	R39	237K	Panasonic	1
Resistor	ERJ-2RKF1153X	R40	115K	Panasonic	1
Resistor	ERJ-2RKF1543X	R41, R43	154K	Panasonic	2
Resistor	ERJ-2RKF2433X	R49, R55	243K	Panasonic	2
Resistor	ERJ-2RKF4991X	R53	4.99K	Panasonic	1
Resistor	ERJ-2RKF2942X	R54	29.4K	Panasonic	1
Resistor	ERJ-2RKF3013X	R58, R61	301K	Panasonic	2
Resistor	ERJ-2RKF1241X	R60	1.24K	Panasonic	1
Resistor	ERJ-2RKF4303X	R64, R69	430K	Panasonic	2
Resistor	ERJ-2RKF1693X	R75	169K	Panasonic	1
Resistor	ERJ-2RKF5762X	R76	57.6K	Panasonic	1
Resistor	ERJ-2RKF7682X	R77, R79	76.8K	Panasonic	2
Resistor	ERJ-2RKF1213X	R85, R91	121K	Panasonic	2
Resistor	ERJ-2RKF1002X	R86, R122,	10K	Panasonic	8
Resistor	ERJ-2RKF1001X	R87, R123,	1K	Panasonic	8
Resistor	ERJ-2RKF7871X	R89	7.87K	Panasonic	1
Resistor	ERJ-2RKF4122X	R90	41.2K	Panasonic	1
Resistor	ERJ-2RKF1503X	R94, R97	150K	Panasonic	2
Resistor	ERJ-2RKF9100X	R96	910	Panasonic	1
Resistor	ERJ-2RKF2153X	R100, R105	215K	Panasonic	2
Resistor	ERJ-2RKF1203X	R111	120K	Panasonic	1
Resistor	ERJ-2RKF2802X	R112	28K	Panasonic	1
Resistor	ERJ-2RKF3832X	R113, R115	38.3K	Panasonic	2
Resistor	ERJ-2RKF6042X	R121, R127,	60.4K	Panasonic	3
Resistor	ERJ-2RKF1102X	R125	11K	Panasonic	1
Resistor	ERJ-2RKF5902X	R126	59K	Panasonic	1
Resistor	ERJ-2RKF7502X	R130, R133	75K	Panasonic	2
Resistor	ERJ-2RKF5900X	R132, R240,	590	Panasonic	4
Resistor	ERJ-2RKF1073X	R136, R141	107K	Panasonic	2
Resistor	ERJ-2RKF8452X	R147	84.5K	Panasonic	1
Resistor	ERJ-2RKF1432X	R148	14.3K	Panasonic	1
Resistor	ERJ-2RKF1912X	R149, R151,	19.1K	Panasonic	4
Resistor	ERJ-2RKF3012X	R157, R163	30.1K	Panasonic	2
Resistor	ERJ2RKF1202X	R161	12K	Panasonic	1
Resistor	ERJ-2RKF8062X	R162	80.6K	Panasonic	1
Resistor	ERJ-2RKF3742X	R166, R169	37.4K	Panasonic	2
Resistor	ERJ-2RKF4530X	R168, R276	453	Panasonic	2
Resistor	ERJ-2RKF5362X	R172, R177	53.6K	Panasonic	2
Resistor	ERJ-2RKF7151X	R184	7.15K	Panasonic	1
Resistor	ERJ-2RKF9531X	R185, R187	9.53K	Panasonic	2
Resistor	ERJ-2RKF1542X	R193, R199	15.4K	Panasonic	2
Resistor	ERJ-2RKF2002X	R194, R196,	20K	Panasonic	3
Resistor	ERJ-2RKF4992X	R197, R233,	49.9K	Panasonic	5
Resistor	ERJ-2RKF3482X	R198, R234	34.8K	Panasonic	2
Resistor	ERJ-2RKF8060X	R204	806	Panasonic	1
Resistor	ERJ2RKF2702X	R208, R213	27K	Panasonic	2
Resistor	ERJ-2RKF4222X	R219	42.2K	Panasonic	1
Resistor	ERJ-2RKF3571X	R220	3.57K	Panasonic	1
Resistor	ERJ-2RKF4751X	R221, R223	4.75K	Panasonic	2
Resistor	ERJ-2RKF7501X	R229, R235	7.5K	Panasonic	2
Resistor	ERJ-2RKF1003X	R230, R266,	100K	Panasonic	7

Resistor	ERJ-2RKF9311X	R238, R241	9.31K	Panasonic	2
Resistor	ERJ-2RKF1332X	R244, R249	13.3K	Panasonic	2
Resistor	ERJ-2RKF3002X	R255	30K	Panasonic	1
Resistor	ERJ-2RKF1781X	R256	1.78K	Panasonic	1
Resistor	ERJ-2RKF2401X	R257, R259	2.4K	Panasonic	2
Resistor	ERJ-2RKF3741X	R265, R271	3.74K	Panasonic	2
Resistor	ERJ-2RKF1502X	R270	15K	Panasonic	1
Resistor	ERJ-2RKF4701X	R274, R277	4.7K	Panasonic	2
Resistor	ERJ-2RKF6651X	R280, R285	6.65K	Panasonic	2
Resistor	ERJ-2RKF2202X	R291	22K	Panasonic	1
Resistor	ERJ-2RKF8870X	R292	887	Panasonic	1
Resistor	ERJ-2RKF1211X	R293, R294	1.21K	Panasonic	2
Resistor	ERJ-2RKF1871X	R301, R307	1.87K	Panasonic	2
Resistor	ERJ-2RKF8451X	R306	8.45K	Panasonic	1
Resistor	ERJ-2RKF2321X	R310, R313	2.32K	Panasonic	2
Resistor	ERJ-2RKF3010X	R312, R365,	301	Panasonic	3
Resistor	ERJ-2RKF3321X	R316, R321	3.32K	Panasonic	2
Resistor	ERJ-2RKF4640X	R328	464	Panasonic	1
Resistor	ERJ-2RKF6040X	R329, R331	604	Panasonic	2
Resistor	ERJ-2RKF9530X	R337, R343	953	Panasonic	2
Resistor	ERJ-2RKF6981X	R342	6.98K	Panasonic	1
Resistor	ERJ-2RKF1151X	R346, R349	1150	Panasonic	2
Resistor	ERJ-2RKF7500X	R348	750	Panasonic	1
Resistor	ERJ-2RKF1651X	R352, R357	1.65K	Panasonic	2
Resistor	ERJ-2RKF2260X	R364	226	Panasonic	1
Resistor	ERJ-2RKF4750X	R373, R379	475	Panasonic	2
Resistor	ERJ-2RKF4990X	R375	499	Panasonic	1
Resistor	ERJ2RKF2701X	R378	2.7K	Panasonic	1
Resistor	ERJ-2RKF2201X	R384	2.2K	Panasonic	1
Resistor	ERJ-2RKF8250X	R388, R393	825	Panasonic	2
Resistor	CRCW04020000Z0EDHP	R397 - R418, R444, R445, R469, R470, R491, R492	0	Vishay Dale	28
Resistor	ERJ-2RKF10R0X	R442	10	Panasonic	1
Resistor	CRCW04020000Z0EDHP	R475- R486	0	Vishay Dale	14
Resistor	ERJ-2RKF2001X	R447	2K	Panasonic	1
Resistor	ERA2AEB102X	R448, R449	1K	Panasonic	2
Resistor	ERJ-2RKF1002X	R455, R460		Panasonic	2
Resistor	ERJ-6ENF3833V	R456	384K	Panasonic	1
Resistor	ERJ-6ENF1004V	R457	1M	Panasonic	1
Resistor	3214W-1-103E	R489		Bourns	1
Resistor	ERJ-2RKF1000X	R490	100	Panasonic	1
Switch	418121160812	SW1, SW2		Wuerth Elektronik	2
Test Point	5015	TP1 - TP48		45	
Op-Amp	AD822	U1 - U69		68	
ADC	AD7476AAKSZ-500RL7	U70		Analog Devices	1
Precision Reference	REF195ESZ	U71		Analog Devices	1
Buck Regulator	MAX5035BASA+T	U72		Maxim Integrated	1

Split Rail Regulator	TPS65133DPDR	U73		Texas Instruments	1

Table D.2: Electrical Bill of Materials

Bibliography

[1] C. Valens. (1999) A really friendly guide to wavelets. University of New Mexico. [Online]. Available: agl.cs.unm.edu/~williams/cs530/arfgtw.pdf

[2] C. Torrence and G. Compo. (1998) A practical guide to wavelet analysis. University of Colorado. [Online]. Available: https://paos.colorado.edu/research/wavelets/bams_79_01_0061.pdf

[3] H. Yunyan, P. Minfang, T. Chenglai, and T. Hu, "Analog circuit fault diagnosis using multi-wavelet transform and svm," in *2012 Third International Conference on Digital Manufacturing Automation*, 2012, pp. 214–217.

[4] S. A. P. Haddad, R. Houben, and W. A. Serdijn, "Analog wavelet transform employing dynamic translinear circuits for cardiac signal characterization," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. 1, 2003, pp. I–I.

[5] A. J. Casson, "An analog circuit approximation of the discrete wavelet

transform for ultra low power signal processing in wearable sensor nodes," *Sensors*, 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4721816/

[6] *IEEE Std 802.11b*, IEEE Std., 1999.

[7] L. Fang, L. Wu, and Y. Zhang, "A novel demodulation system based on continuous wavelet transform," *Mathematical Problems in Engineering*, 2015. [Online]. Available: https://www.hindawi.com/journals/mpe/2015/513849/#copyright

[8] C. Lowrie and F. M. Ham, "Wavelet signal demodulation for wireless communications," *Nonlinear Analysis: Theory, Methods Applications*, vol. 63, no. 5, pp. e839 – e845, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0362546X05001628

[9] R. Polikar, *The Wavelet Tutorial, Second Edition, Part I*. [Online]. Available: http://web.iitd.ac.in/~sumeet/WaveletTutorial.pdf

[10] A. Najmi, *Wavelets A Concise Guide*. Baltimore: Johns Hopkins University Press, 2012.

[11] B. Vidakovic. (2004) Basics of wavelets. Georgia Tech. [Online]. Available: https://www2.isye.gatech.edu/~brani/isyebayes/bank/handout20.pdf

[12] E. Cheever. (2019) State space representations of linear physical

systems. Swarthmore. [Online]. Available: https://lpsa.swarthmore.edu/Representations/SysRepSS.html

[13] *State-space Orthonormal Filters: Synthesis and Design*, 2009, chapter 6. [Online]. Available: https://pdfs.semanticscholar.org/cc92/942fd5e4ea396c8e60cb0dd213a47d2a0d5a.pdf

[14] Padé approximation. University of Colorado Boulder. [Online]. Available: https://www.colorado.edu/amath/sites/default/files/attached-files/pade_2.pdf

[15] G. H. Golub and C. F. V. Loan, *Matrix Computations, 2nd Edition*. Baltimore: Johns Hopkins University Press, 1989.

[16] D. Mårtensson, "Estimate a state space model by using singular value decomposition," Mathematics Stack Exchange. [Online]. Available: https://math.stackexchange.com/q/2580893

[17] M. Grashuis, "A fully differential switched capacitor wavelet filter," Master's thesis, Daft University of Technology, 2009. [Online]. Available: https://pdfs.semanticscholar.org/9e4a/305515d9f595314695c9e68ecdd6c5311eec.pdf

[18] R. Ober, "Asymptotically stable all-pass transfer functions: Canoni-

cal form, parametrization and realization," *IFAC Proceedings Volumes*, vol. 20, pp. 181–185, 07 1987.

[19] R. Ober and D. Mcfarlanen, "Balanced canonical forms for minimal systems: A normalized coprime factor approach," *Linear Algebra and its Applications*, vol. s 122–124, p. 23–64, 09 1989.

[20] H. Qingxiu and H. Yigang, "Analog cmos high-frequency continuous wavelet transform implemented by instantaneous companding circuits," in *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003*, vol. 1, 2003, pp. 154–159 vol.1.

[21] E. W. Justh and F. J. Kub, "Analog cmos high-frequency continuous wavelet transform circuit," in *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, 1999, pp. 188–191 vol.2.

[22] *AD822, Single-Supply, Rail-to-Rail Low Power FET-Input Op Amp*, Analog Devices, datasheet. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD822.pdf

[23] *AD7476A, 2.35 V to 5.25 V, 1 MSPS,12-/10-/8-Bit ADCs in 6-Lead SC70*, Analog Devices, datasheet. [Online]. Available: https://www.mouser.com/datasheet/2/609/AD7476A_7477A_7478A-1501904.pdf

[24] *Atmel SAM D21E / SAM D21G / SAM D21J SMART*

ARM-Based Microcontroller, Atmel/Microchip, datasheet. [Online]. Available: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/Atmel-42181-SAM-D21_Datasheet.pdf

[25] *Application Note AT07627: ASF Manual (SAM D21), ASF Programmers Manual*, Atmel/Microchip, pg. 73-80. [Online]. Available: http://ww1.microchip.com/downloads/en/AppNotes/Atmel-42258-ASF-Manual-SAM-D21_AP-Note_AT07627.pdf

[26] M. L., "Using adc with dma," Arduino Forums, 2017. [Online]. Available: https://forum.arduino.cc/index.php?topic=518461.0

[27] *MKRZero_V5.0*, Arduino, 2016, schematic. [Online]. Available: https://www.arduino.cc/en/uploads/Main/ArduinoMKRZero-schematic.pdf

[28] A. Molnar, "Ece 5540 advanced analog ic design: Lecture 14-15 continuous time filter design," Cornell University, 2017.

Vita

Daniel Eddowes is a member of the professional staff at the Johns Hopkins University Applied Physics Laboratory. He works primarily in the Space Sector RF Engineering Group, where he is an RF and analog hardware engineer. He is working on next-generation, low-power, high-reliability radio products. He received a B.S. in Electrical and Computer Engineering from Cornell University in 2017.