Polarization Modulation for Application in Terahertz Sensing

By
Matthew McLean Dwyer

A thesis submitted in part for the degree of
Master of Science (Electrical Engineering)

School of Electrical and Computer Engineering,
University of Wisconsin - Madison.

May 2011

Acknowledgments

I owe a great deal of thanks to Dan van der Weide for taking me on many years ago as a study abroad student, and then as a graduate student. I also owe a great deal of thanks to John Grade who fabricated the antennas used in this project, and provided invaluable microfabrication advice.

Abstract

Providing defense against asymmetric threats under many different environmental conditions requires broad spectrum sensing and presently underutilized polarization sensitivity. In this thesis a novel pulsed radar sensing modulation scheme is proposed, exploiting pseudorandomly switching polarization and polarity. The scheme was designed for use in the terahertz (THz) regime while a scale model demonstration has provided encouraging results. Polarization sensitivity and ranging exceeding the pulsed radar range ambiguity limit have been achieved. Further, it has been shown that multiple transmitters can operate simultaneously using different PRBS with receiver rejection at the experiment noise floor. Antenna improvements have been investigated with suggestions provided for THz implementation. Finally, a novel demodulation algorithm is outlined that would enable demodulation in the THz regime while also characterizing an object's RCS over frequency and polarization.

Contents

List of Figures

CHAPTER 1

Background

This chapter is intended to briefly cover topics relevant to the theory presented in this thesis.

1.1. Antenna Performance Metrics

Gain is a measure of the efficiency and directivity of an antenna. An antenna with more gain will direct more energy on boresight than an antenna with less gain. Return loss is an easily measured metric that can be interpreted as the fraction of energy that is not accepted by an antenna. However, accepted energy is not necessarily radiated - it could also dissipated through metallic or dielectric losses. The bandwidth of an antenna is the frequency range over which the antenna meets certain criteria, i.e. a minimum gain or return loss. [1]

1.2. Polarization & Sinclair Matrix

Polarization refers to the orientation of the electric field vector, which can be represented by the weighted sum of two orthogonally polarized components. These orthogonal components could be, for instance, left and right circular, or vertical and horizontal linear as used here. The reflected electric field from an object, E^r, can be related to the incident electric field, E^i, by the Sinclair matrix multiplied by a distance scaling factor, shown in equation (1.1) [2].

$$\begin{bmatrix} E_x^r \\ E_y^r \end{bmatrix} = \frac{1}{\sqrt{4 \cdot \pi} \cdot r} \begin{bmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{bmatrix} \begin{bmatrix} E_x^i \\ E_y^i \end{bmatrix} \tag{1.1}$$

By taking into account polarization, here x and y polarized, we now have four object identifiers instead of one.

1.3. RADAR

Radio direction and ranging (RADAR) is a technique heavily developed during World War II which allows, as the acronym implies, object direction and ranging, as well as speed and altitude. Imaging of terrain is also possible through synthetic aperture RADAR.

The signal to noise ratio (SNR) in RADAR is directly proportional to the transmitted power. While this transmitted power can be high in continuous wave

RADAR, the average transmitted power is much lower in pulsed RADAR as the high peak power duration is typically much less than the pulse repetition period, determined by the pulse repetition frequency, f_{pr}. However, reducing f_{pr} also reduces the unambiguous target range. In pulsed radar, in order to unambiguously range a target, the reflected pulse must be received before the next pulse can be transmitted. This ambiguity range is given by (1.2) [3].

$$R_u = \frac{c}{2 \cdot f_{pr}} \tag{1.2}$$

If the RADAR pulse stream is somehow modulated, then the above equation is no longer valid. The ambiguity range would then depend on repetition rate of the modulation, independently enabling high f_{pr} and thus high SNR.

1.4. Matched Filtering

Matched filtering is the correlation of an incoming signal with a known waveform to detect that waveform in the signal. The matched filter output, *y[n]*, can be found from the convolution of the known waveform, *h[n]*, and the incoming signal, *x[n]*, shown in (1.3) [4]:

$$y[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k] \tag{1.3}$$

We see here that the known waveform is being used as a linear filter. Matched filtering can be implemented in the mathematical software package MATLAB using the function *xcorr* on h and x.

CHAPTER 2

Theory

2.1. Polarization Modulation Theory

We propose a polarization modulation approach for encoding a stream of pulses that would be emitted by a time-domain terahertz (THz) coherent sensing system. Our approach involves driving dual-polarized, dual-polarity ultrawideband antennas with channels having identical pulse streams whose relative time delays (pulse repetition phase) are modulated by a PRBS or other code. The resultant interferogram is a summation of the polarization vectors from each antenna channel, giving a polarization-coded pulse stream, pseudo-randomly switching both polarity and polarization. An identical antenna in a balanced configuration is used to receive both polarizations. The scattered modulated signals are correlated with the expected response (measured or calculated using the transmitted code) to produce two matched filter responses. The modulation scheme affords an arbitrarily long range ambiguity independent of the pulse repetition frequency. Signal to noise ratio (SNR) is significantly improved due to a long duty cycle and resultant high average power. Because of source coding, many transmitters can operate in an array, with receivers able to select which transmitter's scatter to detect.

The polarization modulation states are categorized as horizontal and vertical with positive/negative polarity as shown in Figure 2.1. Here, two antenna channels are driven in phase to produce the desired polarization. The antenna can also be oriented at 45° and driven one channel at a time to produce a desired polarization/polarity, as we have done in this 20x scale proof-of-concept.

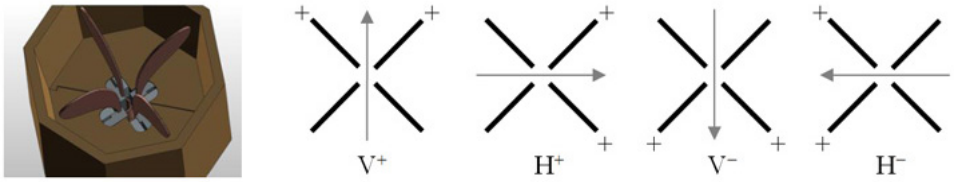

FIGURE 2.1. Polarization/Polarity States

The complex scatter from a target [5] can be known a priori, either from previous measurement in an anechoic environment or from computation with, e.g., the finite difference time domain (FDTD) method. This known response will be correlated,

or match-filtered, with the receiver output to detect the object in the presence of noise. Such noise could be another transmitter using a different PRBS.

Analyzing the effectiveness of the matched filter is best done by considering all cases of the 'pulse-wise' multiplication done during correlation. Consider detecting an object with a dominant polarization, and looking at the matched filter output of that dominant polarization on the receiver. For a positive polarity pulse on the dominant polarization, there is equal probability that the matched filter, unless matched at that delay, will have either dominant or subordinate polarization and either positive or negative polarity at that cycle. Pulse-wise multiplication for these cases, respectively, produces a large positive value, an equally large negative value, a small positive value or an equally small negative value. Statistically, the expected value when the received signal is not matched is zero. Thus in the ideal case and as PRBS length gets large, the matched filter output will produce a peak at the cycle when the received signal aligns with the filter, and zero otherwise. Following the same logic, if the matched filter PRBS differs from the received signal, then that unwanted scatter will not be detected, and will not interfere with the desired PRBS response.

2.2. Dual-polarized Dual-polarity Antennas

In order to implement this modulation scheme we required a dual-polarized, dual-polarity antenna. The antenna used in these measurements was designed and fabricated by Prof. Daniel van der Weide and Dr. John Grade during an earlier phase of the project. It consisted of two orthogonal pairs of vanes, which were themselves driven with orthogonal feeds. These vanes were housed in a shell to reduce crosstalk between neighboring antennas. Improvements to this approach of dual-polarity dual-polarization are discussed in Ch. 5: Future Work.

2.3. Implementation at THz frequencies

While the scale-model demonstration's modulation rate can run at the pulse repetition rate (f_{pr}), this is not possible with the at-scale implementation. The at-scale pulse train is generated by nonlinear transmission lines (NLTL) [6] sharpening the falling edge of the negative half-cycle of a continuous wave (CW) source operating at upwards of 10 GHz. The polarization modulation is fundamentally achieved by switching the phase of the CW source feeding each port of the antenna by 180^{o}, shown in Figure 2.2. The phase shifting circuit is equivalent to a binary phase shift key (BPSK) modulator, which have achieved switching speeds well over 50 GHz [7].

The at-scale f_{pr} cannot feasibly be switched at the same high rate due to timing issues. By using a lower modulation rate the polarity of the radiated pulses will oscillate at f_{pr}, while the polarization will switch at the modulation rate. This

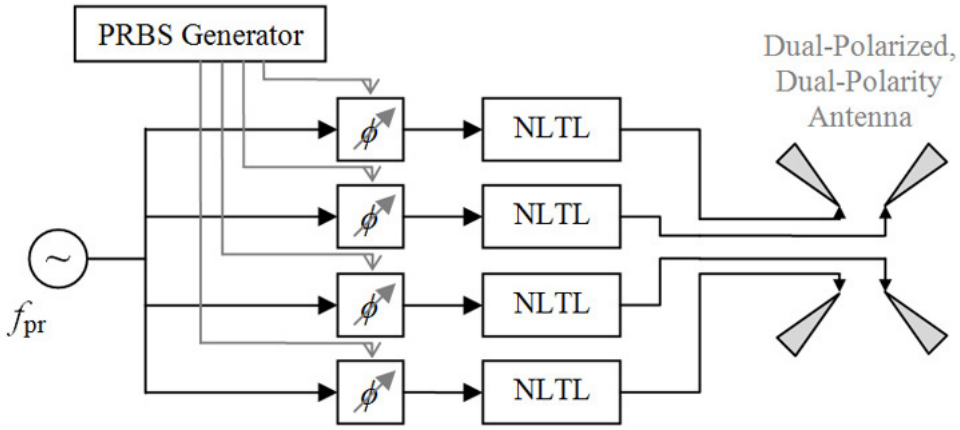

FIGURE 2.2. Transmitter architecture

means that there will be pristine pulses within a clock cycle, even if the beginning and end pulses adjacent to a transition are noisy. This approach is visualized in Figure 2.3, where we can see that pulses can be shifted by 180° at the transition point, resulting in a changed polarization or polarity vector.

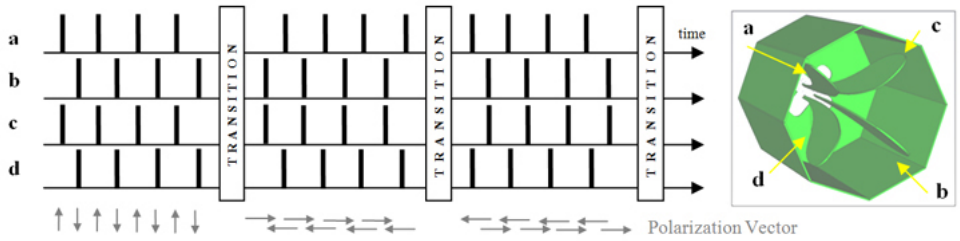

FIGURE 2.3. Pulse stream delay to change polarization vector

Proposed but untested enhancements to this theory are discussed in Ch 5: Future Work.

CHAPTER 3

Experiment and Implementation

3.1. Introduction

The goal of the experimental setup was the confirm that the theory presented in the previous chapter could be applied in practice. Since a THz system driving NLTLs was not immediately practical, a proof-of-concept 20x scale experiment was devised.

3.2. Setup

The THz implementation has the NLTL output modulated by a pseudorandom number generator, as discussed in the previous chapter. This was emulated using pulse generators with patterns stored in memory. The data was then recorded using an oscilloscope and processed by the computer over GPIB using MATLAB. The target chosen was a parallel wire grid polarizer to enhance polarizaton contrast, where the wire spacing was chosen to be much smaller than the smallest wavelength of interest. The idealized experiment setup is shown in Figure 3.1.

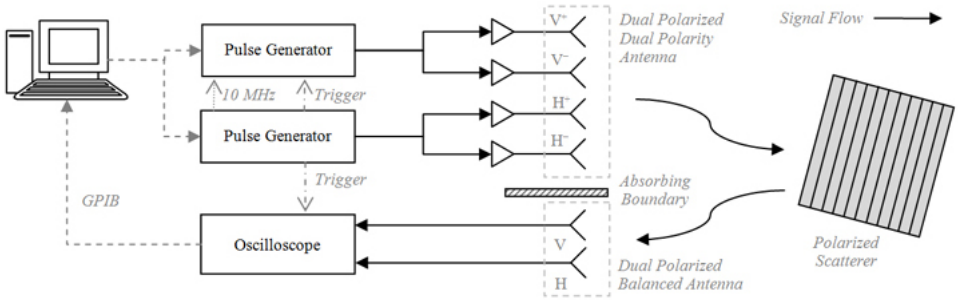

FIGURE 3.1. System overview of 20x proof of concept (idealized)

3.2.1. Issue & workaround. Four independent pulse sources were required to generate polarization and polarity modulated output. One of the pulse sources, an Anritsu data generator, was found to be temporally unstable. That is, despite being synchronized to the other pulse sources, every few seconds it would offset its pattern output from the other sources by a few bits.

As only three distinct PRBS pulse sources/channels were available, the data generator's output was split with one line delayed to decorrelate the output and emulate two distinct PRBSs. Simulations showed that the matched filter response

was still sufficient, but with negative correlation either side of the match, shown in Figure 3.2.

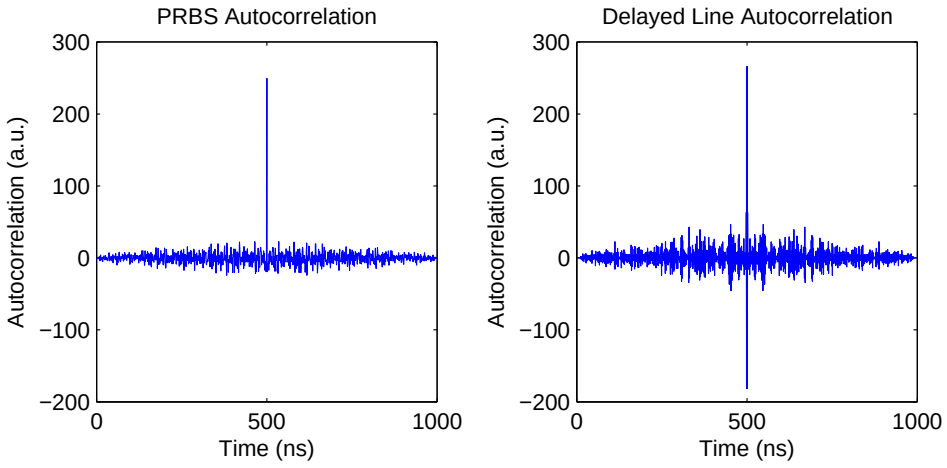

FIGURE 3.2. Autocorrelation of simulated response for PRBS (left) and PRBS with delay line (right)

3.2.2. Final implementation. The 20x proof-of-concept used a pattern generator operating at 500 MHz and 10% duty cycle to simulate the delay shifted output of the at-scale time-domain baseband pulses of a THz sensing system. The PRBSs were generated in MATLAB using the *rand* function and sent to the instruments over GPIB. An Agilent 81134A Pulse Pattern Generator and an Agilent 70843B Data Generator were used to generate the PRBS pulse trains. The latter instrument featured the delay line work around, shown in Figure 3.3. The four pulse streams were amplified by Agilent 83020A 26.5 GHz power amplifiers, and radiated by a Tera-X 20x scale dual-polarized dual-polarity antenna. The scattered response was recorded using an Agilent 86100A oscilloscope with 50 GHz sampling module, triggered on the start of the PRBS.

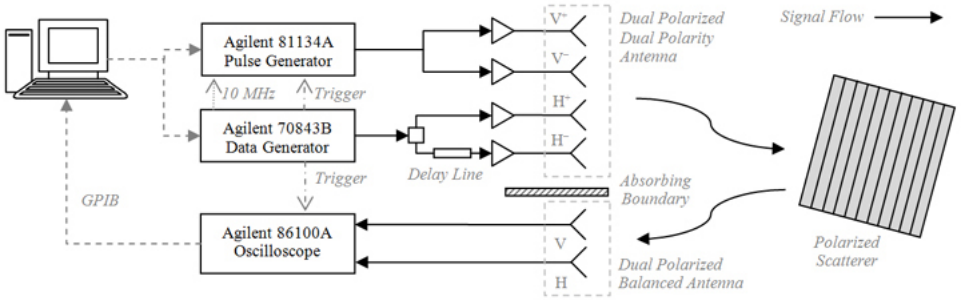

FIGURE 3.3. System overview of 20x proof of concept (with work around)

All instruments were setup remotely using MATLAB over GPIB/LAN. MATLAB was also used to download the waveform data from the oscilloscope. As the

modulated waveform was significantly longer than allowed by the oscilloscope, a script was written to step through a waveform in readable chunks, then appended together to give the full result. The code for this can be found in Ch. 7: Appendix.

3.3. Physical Setup

The experiment was set up in an anechoic chamber to reduce noise caused by undesired reflections. Further, the Rx and Tx antennas were separated by radio absorbing material (RAM) to reduce direct cross talk. The polarizer was mounted on a dielectric stand surrounded by RAM. Images of the setup are shown in Figure 3.4.

FIGURE 3.4. Experiment setup (clockwise from top left): Rx antenna and oscilloscope; Agilent 70843B Data Generator; wire grid polarizer; splitter and delay line; Rx and Tx antennas; Agilent 81134A Pulse Pattern Generator; Agilent 83020A 26.5 GHz power amplifiers.

CHAPTER 4

Measurements

4.1. Matched Filtering

To emulate matched filtering, the received waveform was either autocorrelated, or correlated with another previously saved waveform. This signal processing was done in MATLAB using the *xcorr* function. In actual use, the received waveform would be run through a matched filter corresponding to the expected response for a given PRBS and object.

4.2. Results

We used the system of Figure 3.3 to detect a 1 ft^2 polarizer in an anechoic environment with varying bit sequence lengths. The target's polarization was rotated some 20° off the receiving polarization, and was situated one meter from the antennas. The correlation calculations used the vertically polarized received signal, since that was closer to the target's orientation. The 20-bit and 100-bit sequences resulted in correlation sidebands at ~ -7 dB, while the 400 bit sequence had sidebands at ~ -10 dB, seen in Figure 4.1. As would be expected, the sidebands reduced with increasing sequence length.

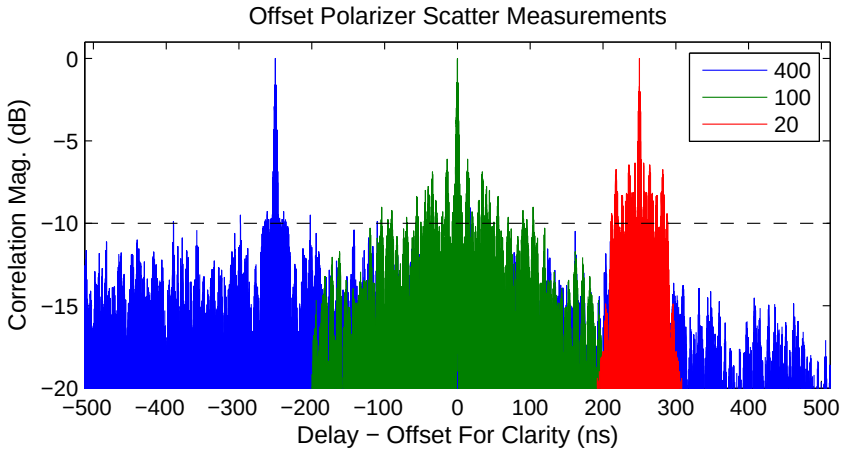

FIGURE 4.1. Scatter measurements with varying PRBS lengths.

Comparing the autocorrelation of the scattered response against the cross-correlation with an alternate PRBS shows a 10 dB differential when using a 400 bit sequence (Figure 4.2). This would allow multiple transmitters to operate simultaneously, with

receivers having the ability to choose which transmitter's scatter to detect. As the side bands on the autocorrelation are at the level of the cross-correlation peak, we can assume that the alternative PRBS rejection level is greater than the noise floor.

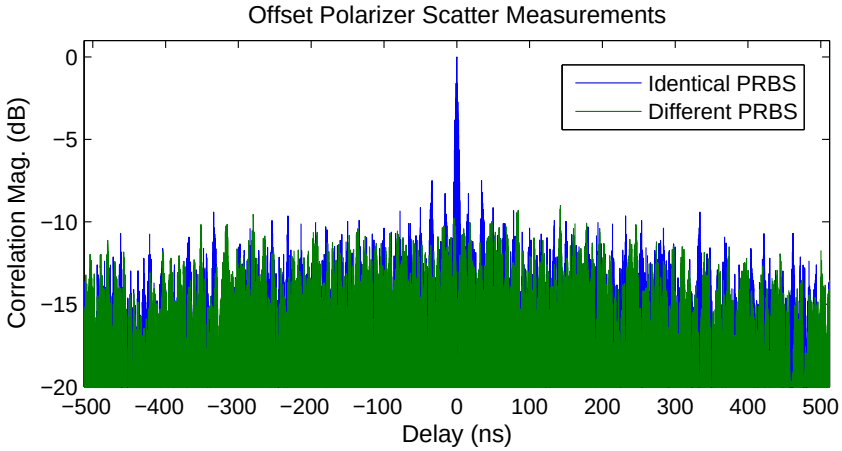

FIGURE 4.2. Scatter measurements with matched vs. unmatched (different) PRBS.

The polarizer was then rotated to highlight the benefit of polarization sensitivity (Figure 4.3). When the polarizer is rotated at 45° to the receiving antenna, the correlation magnitude is reduced by 3 dB, as would be expected. When the polarizer is rotated orthogonally to the receiving antenna, the correlation magnitude is some 10 dB down, around the experiment's noise floor. Theoretically the reflection from the orthogonal polarizer should be nil, thus the -10 dB peak shown is likely due to leakage from the Tx antenna to Rx antenna by a more direct path through the absorbing foam, or backlobe radiation.

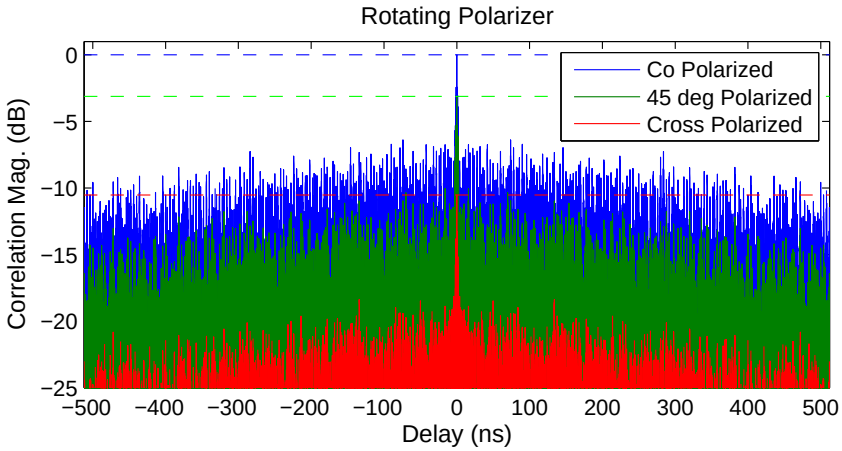

FIGURE 4.3. Scatter measurements with rotating polarizer.

Successful ranging was achieved, with a target distance change of 3 ft resulting in a 6 ns delay difference (Figure 4.4). Given there is a 6 ft round trip differential and

light travels at roughly 1 ft/ns, this is expected. Unmodulated pulsed radar would have a range ambiguity (given by $R_u = \frac{c}{2 \cdot f_{pr}}$ [3]) of only 1 ft at the experiment's pulse repetition frequency.

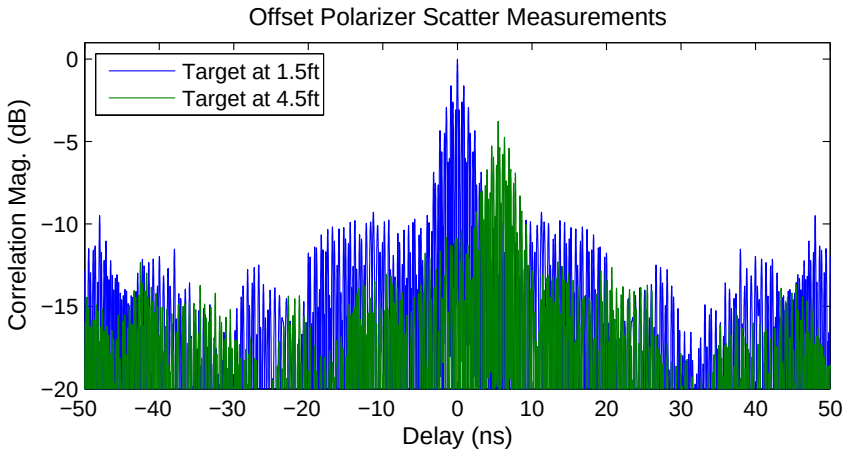

FIGURE 4.4. Scatter measurements with polarizer at varying range.

CHAPTER 5

Future Work

5.1. Implementation at THz frequencies

The future work relating to this project is largely concerned with implementing the technique at THz frequencies. Given the promising results of the scaled model, the emphasis is now on replicating these results using an NLTL-based system. Drawbacks of current NLTLs largely stem from the gallium arsenide (GaAs) substrate that must be used for diode fabrication. Here alternatives are proposed that could potentially lead to faster NLTLs and more efficient, higher gain antennas. This work will form the basis of the author's PhD research.

5.2. Antenna Improvements

As discussed in the next section, vivaldi antenna performance suffers when the substrate used is electrically thick. However, when the substrate is thinned, the microstrip feed becomes impractically small. For implementation in the scale model, we proposed a novel balun feed that uses a thick substrate for the microstrip feed, but a thin substrate for the slotline fed antenna, shown in Figure 5.1. The microstrip short circuit was achieved with a via, while the slotline open circuit was achieved with a radial slot. This enabled us to investigate electrically thin vivaldi antennas at the scaled frequencies (up to 20 GHz), for comparison with the project's original antenna. To implement a dual-polarized, dual-polarity vivaldi antenna, four equivalent antennas should be arrange in a cross shape, each representing a polarization/polarity state.

Four prototypes were simulated, each with varying dielectric arrangements at the output (Figure 5.2). Prototype A was a typical vivaldi, while B had a thin dielectric lens of sorts. Prototypes C and D were two approaches to removing dielectric material.

Comparing the return loss of these four prototypes, we see in Figure 5.3 that performance is largely constant, though prototypes A and B show slightly stronger performance.

The gain of the four antennas was calculated over the designed frequency range, with prototype B the strongest performer, shown in Figure 5.4. This is possibly due to the slight collimating effect of the lens-like dielectric, particularly at higher frequencies.

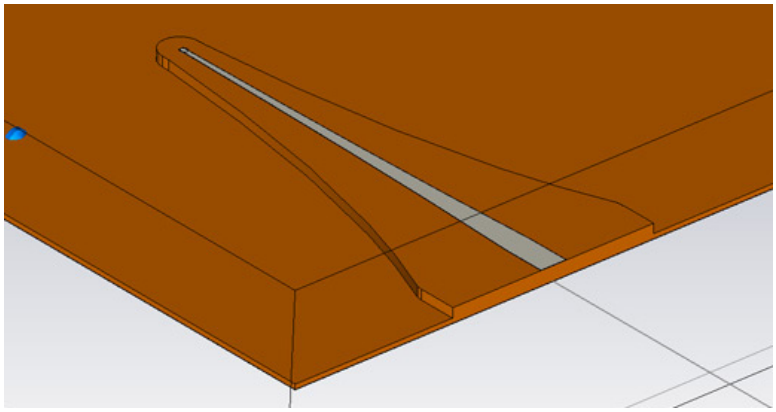

FIGURE 5.1. Novel vivaldi balun feed

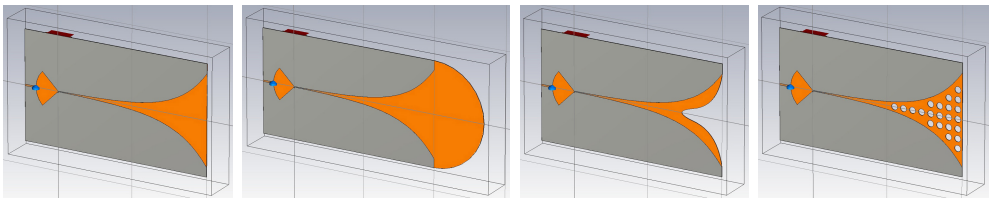

FIGURE 5.2. Prototype vivaldi antennas (L to R) A, B, C and D

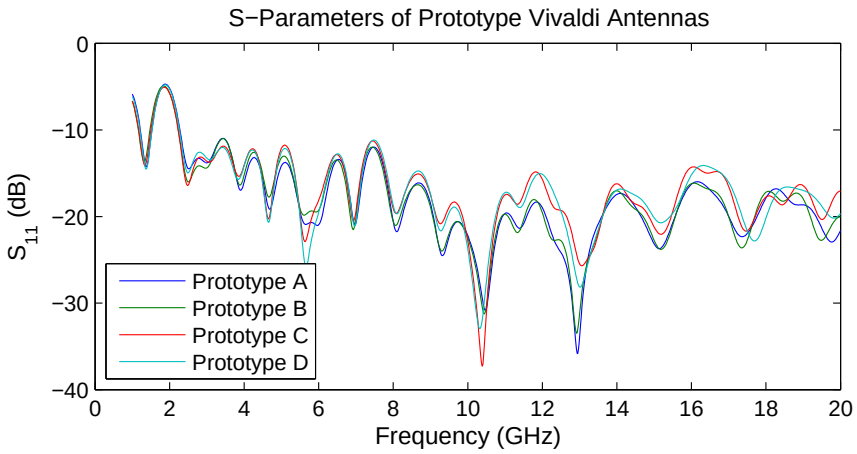

FIGURE 5.3. Simulated S-parameters of vivaldi prototypes

The measured S-parameters of prototype B were compared with that of the original antenna (Figure 5.5). It is evident that the prototype performed significantly better over almost the entire frequency range. This provides strong motivation to integrate a vivaldi antenna with an NLTL over the original antenna, and is discussed further in the next section.

5.3. NLTL/Vivaldi Improvements at THz Frequencies

The critical drawback of using NLTLs as pulse sources is that they must be fabricated on gallium arsenide (GaAs). GaAs has a permittivity of ~12.9, thus the

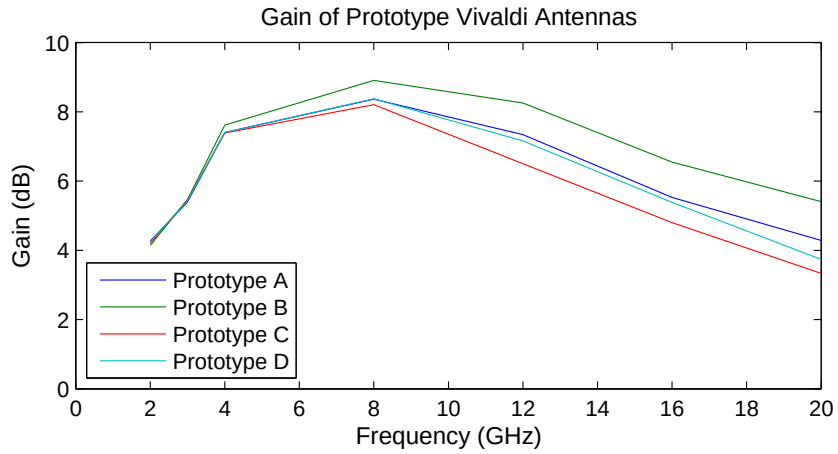

FIGURE 5.4. Simulated gain of vivaldi prototypes

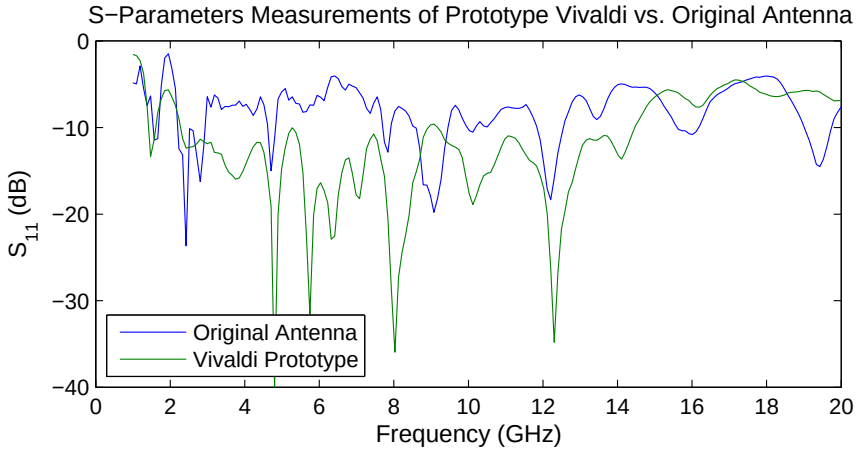

FIGURE 5.5. Measured S-parameters of prototype B and original antenna

0.5 mm substrate becomes electrically very thick. Using another metric, we see that the speed of the pulse wave in the circuit will be significantly slower than freespace, resulting in a poor match. We propose using a GaAs membrane on a thin diamond scaffold to fabricate the NLTL, thus allowing the integrated antennas to be virtually suspended in free space. This will allow antennas with higher gain, efficiency and bandwidth.

Further, NLTLs will benefit from the removal of the high permittivity GaAs through increased nonlinearity, compression and lower losses. Also, different GaAs epitaxial structures on the one device will be possible, allowing for very nonlinear diodes to be designed separately from ultra fast sampling diodes.

5.4. Averaging Algorithm to Obtain Pulse Shape

The modulation technique proposed in this paper relies on knowing wideband and polarization dependent RCS of an object in order to create the matched filter.

A more robust approach that requires only knowledge of the PRBS is to switch the polarity of the receiving samplers. That is, when expecting a positive pulse, we output samples from the positive polarity antenna. Conversely, we output negative samples when expecting a negative pulse. This effectively corrects the polarity of the negative pulses. This sampling is shown idealized in Figure 5.6, where the red cross represents a sample point. In reality, the sampling points are further apart.

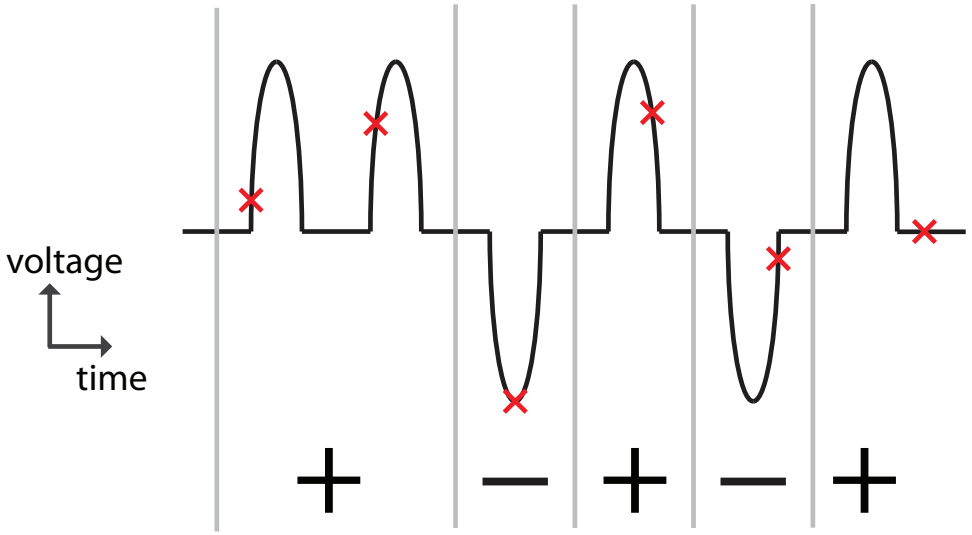

FIGURE 5.6. Idealized pulse sampling

An idealized antenna/circuit implementation of this approach is shown in Figure 5.7.

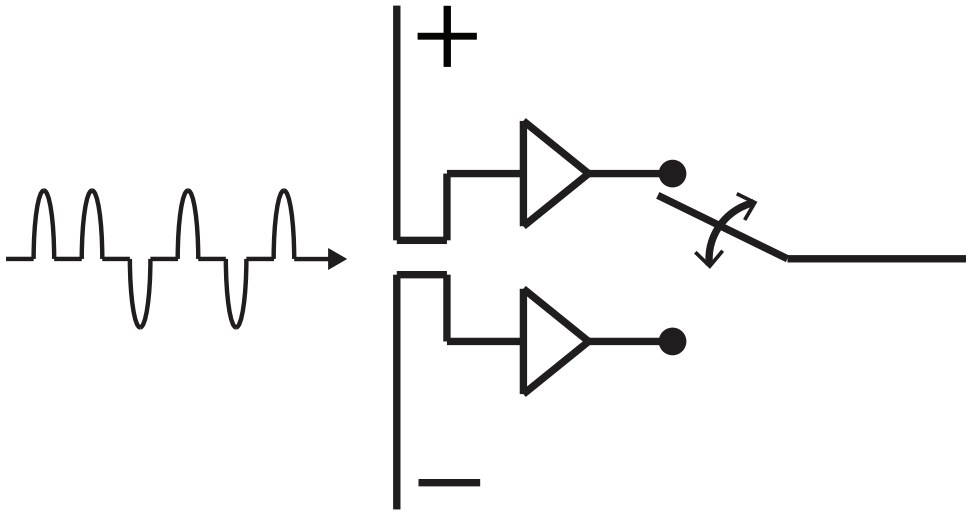

FIGURE 5.7. Idealized sample switching antenna/circuit

Further, through averaging of these sampled pulses, undesired PRBS pulse trains will be rejected (averaged out). When implemented in the THz regime, the pulse

will in fact be a group of pulses that are generated in one cycle of the modulation frequency. While this pulse train will not always be greater than zero volts, it will be the polar opposite of a negative modulation cycle. The switching of the outputs will then be done at the modulation frequency, which could be on the order of 1 GHz. Such a switching speed is certainly realizable.

A key benefit to this approach is that it does not require a priori knowledge of an object's RCS. This method will provide all four elements of the Sinclair matrix, and over a wide bandwidth, thus enabling object identification in addition to detection.

CHAPTER 6

Conclusion

The initial results of the scaled polarization modulation demonstration are promising. It has been shown that the scheme allows the detection of objects that exceed the ambiguity range associated with the pulse repetition frequency, and that by changing the PRBS, multiple transmitters can operate simultaneously. Further, the range resolution of 3 ft is expected to scale down linearly from the 20x demonstration to an at-scale THz system. Through rotating the polarizer, we have also shown that the scheme is indeed polarization-sensitive.

Antenna improvements have been proposed, investigated and shown to improve on the original vaned antenna. THz-scale NLTL and antenna improvements have been outlined for further investigation in the author's PhD research. Finally, a novel demodulation algorithm has been proposed that could be implemented in the THz-regime.

Bibliography

[1] C. Balanis, *Antenna theory: analysis and design*, 1982, vol. 1.

[2] H. Mott, *Antennas for Radar and Communications: A Polarimetric Approach*, K. Chang, Ed. John Wiley & Sons, Inc, 1992.

[3] B. Mahafza, "Radar systems analysis and design using MATLAB," 2000.

[4] G. Turin, "An introduction to matched filters," *Information Theory, IRE Transactions on*, vol. 6, no. 3, pp. 311–329, 1960.

[5] M. Hurtado and A. Nehorai, "Polarization diversity for detecting targets in inhomogeneous clutter," *Waveform Diversity and Design Conference*, vol. 1, pp. 382–386, 2007.

[6] D. van der Weide, "Delta-doped Schottky diode nonlinear transmission lines for 480-fs, 3.5-V transients," *Applied physics letters*, vol. 65, no. 7, pp. 881–883, 1994.

[7] T. Lodhi, D. Edgar, H. McLelland, S. Ferguson, K. Elgaid, C. Stanley, and I. Thayne, "A 77 GHz coplanar waveguide MMIC BPSK vector modulator realised using InP technology," pp. 183–186, 2000.

CHAPTER 7

Appendix

Appendix Preface

This appendix consists of the MATLAB code that controlled the experiment. Of particular interest would be the algorithm that allowed the capture of large waveforms beyond the capacity of the Agilent 86100A oscilloscope. This was done by stepping through a long waveform section by section, then appending these sections to give the entire waveform.

7.1. MATLAB: PRBS Setup

```
%% Pseudo Random Bit Sequence Setup
% Author: Matt Dwyer

%% 0. Initial Variables
trigDiv = 512;  % PRBS Length (inc padding)
%% 0a. PRBS length = 20
padding = (512-20)/2;    % Leading & trailing zeros
%% 0a. PRBS length = 100
padding = (512-100)/2;    % Leading & trailing zeros
%% 0a. PRBS length = 400
padding = (512-400)/2;    % Leading & trailing zeros

%% 1b. Generate a PRBS

PRBSa = zeros(trigDiv,1);
PRBSb = zeros(trigDiv,1);
PRBSc = zeros(trigDiv,1);
PRBSd = zeros(trigDiv,1);
%
% % Marker bit for alignment
% PRBSa(1) = 1;
% PRBSb(1) = 1;
% PRBSc(1) = 1;
% PRBSd(1) = 1;    % anritsu
```

```
skip = 0;
for i = 1:(trigDiv-2*padding),
    ranBit = rand(1);
    if skip == 0,
        if ranBit < 0.3333,
            PRBSa(i+padding) = 1;    % Agilent Pulse Gen Ch1
        elseif ranBit < 0.6666,
            PRBSb(i+padding) = 1;    % Agilent Pulse Gen Ch2
        else
            if i == (trigDiv-2*padding),
                PRBSa(i+padding) = 1;    % don't choose DataGen for last
                    bit
            else
                PRBSc(i+padding) = 1;    % Agilent Data Gen
                PRBSd(i+padding+1) = 1;    % Redundant
                skip = 1;
            end
        end
    else
        skip = 0;
    end
end

PRBS_Vp = char(1:(trigDiv/4));        % hex
PRBS_Vn = char(1:(trigDiv/4));        % hex
PRBS_Hp = zeros(0,(10*trigDiv/8));    % binary (byte/decimal)
PRBS_Hn = char(1:(10*trigDiv/4));     % hex

i = 0;
for p = 1:4:trigDiv,
    i = i + 1;
    PRBS_Vp(i) = num2str(dec2hex(8*PRBSa(p)+4*PRBSa(p+1)+2*PRBSa(p+2)
        +1*PRBSa(p+3)),'%X');
    PRBS_Vn(i) = num2str(dec2hex(8*PRBSb(p)+4*PRBSb(p+1)+2*PRBSb(p+2)
        +1*PRBSb(p+3)),'%X');
end

% Add extra zeros to data gen patterns to simulate pulse
zPRBSc = zeros(10*trigDiv,1);
zPRBSd = zeros(10*trigDiv,1);
for p = 1:10:(10*trigDiv),
    zPRBSc(p) = PRBSc((p-1)/10 + 1);
```

```
    zPRBSd(p) = PRBSd((p-1)/10 + 1);
end
PRBSc = zPRBSc;
PRBSd = zPRBSd;

% For the agilent data generator (Hp), keep in binary format
i = 0;
for p = 1:8:(10*trigDiv),
    i = i + 1;
    PRBS_Hp(i) = bin2dec(cellstr(num2str(PRBSc(((p-1)+1):((p-1)+8))')))
        ;
end

% Each block of 4 bits must be flipped (then blocks in a page must be
% flipped)
i = 0;
for p = 1:4:(10*trigDiv),
    i = i + 1;
%       PRBS_Hp(i) = num2str(dec2hex(8*PRBSc(p)+4*PRBSc(p+1)+2*PRBSc(p+2)
    +1*PRBSc(p+3)),'%X');
    PRBS_Hn(i) = num2str(dec2hex(1*PRBSd(p)+2*PRBSd(p+1)+4*PRBSd(p+2)
        +8*PRBSd(p+3)),'%X');
end

%

%% 1a. Load PRBS
saveFile = 'dataPRBS.mat';
dataFile = load(saveFile);
PRBS_V = dataFile.PRBS_V;
PRBS_H = dataFile.PRBS_H;
trigDiv = 4 * size(PRBS_V,2);
clear dataFile
clear saveFile


%% 1c. Generate a Single Pulse

PRBSa = zeros(trigDiv,1);
PRBSb = zeros(trigDiv,1);

PRBSa(round(trigDiv/2)) = 1;
```

```
PRBSb(round(trigDiv/2+15)) = 1;

PRBS_V = '';
PRBS_H = '';
for p = 1:4:size(PRBSa,1),
    PRBS_V = [PRBS_V num2str(dec2hex(PRBSa(p)+2*PRBSa(p+1)+4*PRBSa(p+2)
        +8*PRBSa(p+3)),'%X')];
    PRBS_H = [PRBS_H num2str(dec2hex(PRBSb(p)+2*PRBSb(p+1)+4*PRBSb(p+2)
        +8*PRBSb(p+3)),'%X')];
end

%% 3. Save PRBS
savefile = 'dataPRBS.mat';
save(savefile,'PRBS_V','PRBS_H');%% 4e
clear all;
close all;
f = 10^9 .* 10; % freq. in Hz
lambda = 3e8 ./ f;
k = 2*pi./lambda;
d = 0.01; % 10 mm
w = 0.001 .* (0.01:0.01:4); % 1 mm
phi = 0*pi/180;
theta = 0*pi/180;
eta = 377;
Z = 50;
```

7.2. MATLAB: Pulse generator setup

```
%% Send patterns to the 3 pulse generators

%% Agilent 31184A
prf = 0.5;
pulseVolt = 300;

pulseGen = instrfind('Type', 'gpib', 'BoardIndex', 7, 'PrimaryAddress',
    13, 'Tag', '');
if isempty(pulseGen)
    pulseGen = gpib('AGILENT', 7, 13);
else
    fclose(pulseGen);
    pulseGen = pulseGen(1);
end
```

```
fopen(pulseGen);

% Setting up the Pulse Pattern Generator (using Channel 2, output-bar)
fprintf(pulseGen,sprintf('*RST'));
fprintf(pulseGen,sprintf('*CLS'));
fprintf(pulseGen,sprintf('SYSTem:HEADer OFF'));
fprintf(pulseGen,sprintf([':SOURce:FREQuency ' num2str(prf) 'GHz']));
fprintf(pulseGen,sprintf([':OUTPut0:DIVider ' num2str(trigDiv)]));
fprintf(pulseGen,sprintf(':SOURce:FUNCtion PATTern'));
fprintf(pulseGen,sprintf(':SOURce:FUNCtion:MODe2 DATa'));          %
    Uncomment for PRBS Pulse Stream
fprintf(pulseGen,sprintf(':SOURce:FUNCtion:MODe1 DATa'));          %
    Uncomment for PRBS Pulse Stream
fprintf(pulseGen,sprintf(':SOURce:PULSe:DCYCle2 20'));
fprintf(pulseGen,sprintf(':SOURce:PULSe:DCYCle1 20'));
fprintf(pulseGen,sprintf(':SOURce:VOLTage2:HIGH 0V'));
fprintf(pulseGen,sprintf(':SOURce:VOLTage1:HIGH 0V'));
fprintf(pulseGen,sprintf([':SOURce:VOLTage2:LOW ' num2str(pulseVolt) '
    MV']));
fprintf(pulseGen,sprintf([':SOURce:VOLTage1:LOW ' num2str(pulseVolt) '
    MV']));
fprintf(pulseGen,sprintf(':SOURce:VOLTage0:HIGH 100MV'));
fprintf(pulseGen,sprintf(':SOURce:VOLTage0:LOW -100MV'));
fprintf(pulseGen,sprintf([':DIGital2:PATTern:LENGth ' num2str(trigDiv)
    ]));
fprintf(pulseGen,sprintf([':DIGital1:PATTern:LENGth ' num2str(trigDiv)
    ]));
fprintf(pulseGen,sprintf(':DIGital2:SIGNal:FORMat RZ'));
fprintf(pulseGen,sprintf(':DIGital1:SIGNal:FORMat RZ'));
fprintf(pulseGen,sprintf(':OUTPut:CENTral ON'));
fprintf(pulseGen,sprintf(':OUTPut2:NEG ON'));
fprintf(pulseGen,sprintf(':OUTPut1:NEG ON'));
fprintf(pulseGen,sprintf(':OUTPut0 ON'));
tempLength = length(num2str(size(PRBS_Vp,2)));
% length(num2str(size(PRBS_Vn,2))), length(num2str(size(PRBS_Vp,2)))
fprintf(pulseGen,sprintf([':DIGital2:PATTern #' num2str(tempLength)
    num2str(size(PRBS_Vp,2)) PRBS_Vp ', HEX']));
fprintf(pulseGen,sprintf([':DIGital1:PATTern #' num2str(tempLength)
    num2str(size(PRBS_Vn,2)) PRBS_Vn ', HEX']));

fclose(pulseGen);
delete(pulseGen);
```

```
clear pulseGen

%% Agilent 70843B Pattern Generator

pulseGen = instrfind('Type', 'gpib', 'BoardIndex', 7, 'PrimaryAddress',
    18, 'Tag', '');
if isempty(pulseGen)
    pulseGen = gpib('AGILENT', 7, 18);
else
    fclose(pulseGen);
    pulseGen = pulseGen(1);
end

pulseGen.OutputBufferSize = 3000;

fopen(pulseGen);

% send the pattern in ASCII binary

fprintf(pulseGen,sprintf(':SOURce1:PATTern:UPATtern0:USE␣STRaight'));
fprintf(pulseGen,sprintf('PATTern:FORMat␣PACKed,␣8'));
fprintf(pulseGen,sprintf(['':SOURce1:PATTern:UPATtern0:LENGth␣' num2str
    (10*trigDiv)]));
% tempLength = length(num2str(size(PRBS_Hp,2)));
% fprintf(pulseGen,sprintf(['':SOURce1:PATTern:UPATtern1:DATA #' num2str
    (tempLength) num2str(size(PRBS_Hp,2)) PRBS_Hp]));
binblockwrite(pulseGen,PRBS_Hp,'uint8','':SOURce1:PATTern:UPATtern0:DATA
    ␣');

% Need to take 8 bits, convert to decimal
% binblockwrite(pulseGen,binPattern,'uint8',':SOURce1:PATTern:UPATtern0
    :DATA ');
% fprintf(pulseGen,sprintf('PATTern:FORMat PACKed, 8'));
% fprintf(pulseGen,sprintf(':SOURce1:PATTern:UPATtern0:LENGth 16'));
% binblockwrite(pulseGen,[49 16],'uint8',':SOURce1:PATTern:UPATtern0:
    DATA ');

fclose(pulseGen);
delete(pulseGen);
clear pulseGen
```

```
%% Anritsu Pattern Generator

pulseGen = instrfind('Type', 'gpib', 'BoardIndex', 7, 'PrimaryAddress',
    24, 'Tag', '');
if isempty(pulseGen)
    pulseGen = gpib('agilent', 7, 24);
else
    fclose(pulseGen);
    pulseGen = pulseGen(1);
end

set(pulseGen, 'EOIMode', 'off');
set(pulseGen, 'EOSMode', 'read&write');
fopen(pulseGen);

% testing ----------------------------------
% query(pulseGen, '*IDN?')
% fprintf(pulseGen,sprintf('LGC 0'));
% fprintf(pulseGen,sprintf('PTS 1'));
% fprintf(pulseGen,sprintf('DLN 16'));
% fprintf(pulseGen,sprintf('PAG 1'));
% fprintf(pulseGen,sprintf('BIT #HF0F0'));
% /testing ---------------------------------

fprintf(pulseGen,sprintf('LGC 0'));
fprintf(pulseGen,sprintf('PTS 0'));
fprintf(pulseGen,sprintf('ALT 0'));
fprintf(pulseGen,sprintf(['DLN ' num2str(5*trigDiv)]));

for j = 1:(5*trigDiv/16),
    % REMEMBER TO FLIP THE BLOCKS
    fprintf(pulseGen,sprintf(['PAG ' num2str(j)]));
    fprintf(pulseGen,sprintf(['BIT #H' num2str(fliplr(PRBS_Hn((1+4*(j
        -1)):(4+4*(j-1)))))]));
end



fprintf(pulseGen,sprintf('LGC 0'));
fprintf(pulseGen,sprintf('PTS 0'));
fprintf(pulseGen,sprintf('ALT 1'));
fprintf(pulseGen,sprintf(['DLN ' num2str(5*trigDiv)]));
for j = 1:(5*trigDiv/16),
```

```
    % REMEMBER TO FLIP THE BLOCKS
    fprintf(pulseGen,sprintf(['PAG ' num2str(j)]));
    fprintf(pulseGen,sprintf(['BIT #H' num2str(fliplr(PRBS_Hn((1+4*(j
        -1)):(4+4*(j-1)))))]));
end

fclose(pulseGen);
delete(pulseGen);
clear pulseGen
```

7.3. MATLAB: Oscilloscope setup

```
%% Oscilloscope and Pulse Pattern Generator Setup
% Author: Matt Dwyer

% trigDiv set by a_PRBS_Setup.m
prf = 0.5;              % Pulse repition frequency [GHz]
yDiv = 100e-3;          % Y-Division on oscilloscope [volts]
pulseVolt = -300;    % Pulse height [millivolts]

oscilloscope = instrfind('Type', 'gpib', 'BoardIndex', 7, '
    PrimaryAddress', 7, 'Tag', '');
if isempty(oscilloscope)
    oscilloscope = gpib('AGILENT', 7, 7);
else
    fclose(oscilloscope);
    oscilloscope = oscilloscope(1);
end

oscilloscope.InputBufferSize = 200000;

% Connect to instruments
fopen(oscilloscope);

fprintf(oscilloscope,sprintf('*RST'));                    % reset scope
fprintf(oscilloscope,sprintf('*CLS'));                    % clear status
     registers & output queue
fprintf(oscilloscope,sprintf('SYSTem:HEADer OFF'));

% Channel Settings (Channel 3)
fprintf(oscilloscope,sprintf(':CHANnel3:BANDwidth HIGH'));        % set
    to LOW?
fprintf(oscilloscope,sprintf(':CHANnel3:OFFSet 0'));
```

```
fprintf(oscilloscope,sprintf([':CHANnel3:SCALe ' num2str(yDiv,'%E')]));
fprintf(oscilloscope,sprintf(':CHANnel3:UNITs VOLT'));
fprintf(oscilloscope,sprintf(':CHANnel3:DISPlay ON'));
fprintf(oscilloscope,sprintf(':CHANnel1:DISPlay OFF'));

%% Channel Settings (Channel 4)
% fprintf(oscilloscope,sprintf(':CHANnel4:BANDwidth HIGH'));      % set
    to LOW?
% fprintf(oscilloscope,sprintf(':CHANnel4:OFFSet 0'));
% fprintf(oscilloscope,sprintf([':CHANnel4:SCALe ' num2str(yDiv,'%E')])
    );
% fprintf(oscilloscope,sprintf(':CHANnel4:UNITs VOLT'));
% fprintf(oscilloscope,sprintf(':CHANnel4:DISPlay ON'));

%% Trigger Settings
% fprintf(oscilloscope,sprintf(':TRIGger:SOURce FPANel'));
% fprintf(oscilloscope,sprintf(':TRIGger:SLOPe POSitive'));
% fprintf(oscilloscope,sprintf(':TRIGger:LEVel -0.002'));

% Waveform Acquisition Settings
fprintf(oscilloscope,sprintf(':ACQuire:AVERage 1'));          % Turn
    averaging on
fprintf(oscilloscope,sprintf(':ACQuire:COUNt 25'));           % Average
    count
% fprintf(oscilloscope,sprintf(':WAVeform:SOURce CHANnel3'));
fprintf(oscilloscope,sprintf(':WAVeform:FORMat ASCii'));

% Timebase Settings
fprintf(oscilloscope,sprintf(':TIMebase:REFerence LEFT'));
fprintf(oscilloscope,sprintf(':TIMebase:UNITs TIME'));
```

7.4. MATLAB: Oscilloscope measurement

```
%% Oscilloscope Measurement
% Author: Matt Dwyer


%% 1. Perform measurement

Ts = 1e-11;                    % sampling period, given by (1 / 100 ghz)
samplesReq = trigDiv * (prf * 1e9)^-1 / (Ts);        % eg 12800
numSegments = ceil(samplesReq / 4000);               % eg 4
numPoints = floor(samplesReq / numSegments);         % eg 3200
```

```
lenSegments = numPoints * 1e-11;                    % eg 32e-9

fprintf(oscilloscope,sprintf(':WAVeform:SOURce CHANnel3'));
fprintf(oscilloscope,sprintf([':TIMebase:RANGe ' num2str(lenSegments,'%
    E')]));      % 3.2ns X-Div
fprintf(oscilloscope,sprintf([':TIMebase:POSition ' num2str((trigDiv /
    (prf*1e9)),'%E')]));
fprintf(oscilloscope,sprintf([':ACQuire:POINts ' num2str(numPoints)]));

xData = Ts / 1e-9 * (1:(numPoints*numSegments))';     % in ns
yData = zeros(numPoints*numSegments,1);

for i = 1:numSegments,
    delayStr = [':TIMebase:POSition ' num2str(50e-9 + (i-1)*lenSegments
        ,'%E')];      % Delay requires fudging
    fprintf(oscilloscope,sprintf(delayStr));
    fprintf(oscilloscope,sprintf(':DIGitize CHANnel3'));
    fprintf(oscilloscope,sprintf(':CHANnel3:DISPlay ON'));
    fwrite(oscilloscope,sprintf(':WAVeform:DATA?'));
    yData(((i-1)*numPoints + 1):(((i)*numPoints))) = ...
        (str2double(regexp(fscanf(oscilloscope),',','split')));
end

close all
yData = yData - mean(yData);                    % remove dc component

%% Channel 4
% fprintf(oscilloscope,sprintf(':WAVeform:SOURce CHANnel4'));
% yData2 = zeros(numPoints*numSegments,1);
%
% for i = 1:numSegments,
%     delayStr = [':TIMebase:POSition ' num2str(50e-9 + (i-1)*
    lenSegments,'%E')];      % Delay requires fudging
%     fprintf(oscilloscope,sprintf(delayStr));
%     fprintf(oscilloscope,sprintf(':DIGitize CHANnel4'));
%     fprintf(oscilloscope,sprintf(':CHANnel4:DISPlay ON'));
%     fwrite(oscilloscope,sprintf(':WAVeform:DATA?'));
%     yData2(((i-1)*numPoints + 1):(((i)*numPoints))) = ...
%         (str2double(regexp(fscanf(oscilloscope),',','split')));
% end
%
% close all
```

```
% yData2 = yData2 - mean(yData2);                % remove dc component

plot(xData,yData,'--rs','MarkerSize',3);
% figure
% plot(xData,yData2,'--rs','MarkerSize',3);
```

7.5. MATLAB: Oscilloscope save

```
%% Save yData to Appropriate Variable
% Author: Matt Dwyer

%% PRBS Length Comparisons
% For each length we want to compare autocorrelation and also
% cross-correlation with a response from a different sequence.

%% 2a. PRBS Length = 20
ScatterBG20 = yData;
ScatterR20 = yData;

%% 2a. PRBS Length = 100
ScatterBG100 = yData;
ScatterR100 = yData;

ScatterBG100H = yData;
ScatterR100H = yData;

%% 2a. PRBS Length = 400
ScatterBG400 = yData;
ScatterR400 = yData;

%% 2a. PRBS Length = 400 // Alternate PRBS
ScatterBG400B = yData;
ScatterR400B = yData;

%% Correct Measurements
Scatter20 = ScatterR20 - ScatterBG20;
Scatter100 = ScatterR100 - ScatterBG100;
Scatter400 = ScatterR400 - ScatterBG400;
Scatter100H = ScatterR100H - ScatterBG100H;
Scatter400B = ScatterR400B - ScatterBG400B;

%% 2b. Autocorrelation
subplot(1,2,1)
```

```
plot( ...
    0.01.*(-1*(size(Scatter400)-1):(size(Scatter400))-1),20*log10(abs(
        xcorr(Scatter400))/max(xcorr(Scatter400))), ...
    0.01.*(-1*(size(Scatter100H)-1):(size(Scatter100H))-1),20*log10(abs
        (xcorr(Scatter100H))/max(xcorr(Scatter100H))), ...
    [-512 512],[-10 -10],'k');
title('Gun Silhouette Scatter Measurements');
xlabel('Delay (ns)');
ylabel('Correlation Mag. (dB)');
legendtitle('PRBS Length');
axis([-1000 1000 -20 0]);

%% 2b. Autocorrelation
subplot(1,2,1)
plot( ...
    0.01.*(-1*(size(Scatter20)-1):(size(Scatter20))-1),20*log10(abs(
        xcorr(Scatter20))/max(xcorr(Scatter20))), ...
    0.01.*(-1*(size(Scatter100)-1):(size(Scatter100))-1),20*log10(abs(
        xcorr(Scatter100))/max(xcorr(Scatter100))), ...
    0.01.*(-1*(size(Scatter400)-1):(size(Scatter400))-1),20*log10(abs(
        xcorr(Scatter400))/max(xcorr(Scatter400))), ...
    [-512 512],[-10 -10],'k');
title('Gun Silhouette Scatter Measurements');
xlabel('Delay (ns)');
ylabel('Correlation Mag. (dB)');
legend('20','100','400');
legendtitle('PRBS Length');
axis([-512 512 -20 0]);

%% 2b. Cross-correlation for different PRBS
subplot(1,2,2)
plot(0.01.*(-1*(size(Scatter400)-1):(size(Scatter400))-1),20*log10(abs(
    xcorr(Scatter400))/max(xcorr(Scatter400))), ...
    0.01.*(-1*(size(Scatter400)-1):(size(Scatter400))-1),20*log10(abs(
        xcorr(Scatter400,Scatter400B))/max(xcorr(Scatter400))), ...
    [-512 512],[-12 -12],'k');
title('Gun Silhouette Scatter Measurements');
xlabel('Delay (ns)');
ylabel('Correlation Mag. (dB)');
axis([-512 512 -20 0]);
legend('Identical PRBS','Different PRBS');
```

```
%% 5b. Save All Measurements To File
savefile = 'dataScopeMeasRad0406.mat';
save(savefile, 'ScatterBG20','ScatterBG100','ScatterBG400','
    ScatterBG400B', ...
     'ScatterR20','ScatterR100','ScatterR400','ScatterR400B',...
     'Scatter20','Scatter100','Scatter400','Scatter400B');

%%
%% %% 2b. Autocorrelation of ScatterDBS
% plot(0.01.*(-1*(size(ScatterDBS)-1):(size(ScatterDBS))-1),20*log10(
    abs(xcorr(ScatterDBS))/max(xcorr(ScatterDBS))));
% title('Autocorrelation of Gun Silhouette Scatter Response');
% xlabel('Delay (ns)');
% ylabel('Autocorrelation Magnitude (dB)');
% axis([-512 512 -30 0]);

%% 2c. Scattering w/ Noise
ScatterNoise = yData;

%% 2d. Correlation
plot(0.01.*(-1*(size(Scatter)-1):(size(Scatter))-1),20*log10(abs(xcorr(
    Scatter,ScatterNoise)/max(xcorr(Scatter,ScatterNoise)))));
title('Scatterer Correlated with Scatterer + Noise');
xlabel('Delay (ns)');
axis([-512 512 -60 0]);


%% 5a. Save All Measurements To File
savefile = 'dataScopeMeasAtt.mat';
save(savefile,'AttenuatedCh1','AttenuatedCh2','AttenuatedDG','
    AttenuatedDGdelay');

%% 5b. Save All Measurements To File
savefile = 'dataScopeMeasRad.mat';
save(savefile,'Scatter','ScatterNoise');


%% 6. Archive Measurements File
copyfile('dataScopeMeasAtt.mat',['dataScopeMeasAtt.' datestr(now,'mm.dd
    .yyyy') '.mat']);
copyfile('dataScopeMeasRad.mat',['dataScopeMeasRad.' datestr(now,'mm.dd
    .yyyy') '.mat']);
```

```
%% 1a. Agilent Pulse Gen Ch1 [Attenuated]
AttenuatedCh1 = yData;

%% 1b. Agilent Pulse Gen Ch2 [Attenuated]
AttenuatedCh2 = yData;

%% 1c. Agilent Data Gen [Attenuated]
AttenuatedDG = yData;

%% 1d. Agilent Data Gen Delayed [Attenuated]
AttenuatedDGdelay = yData;

%% 1e. Plot Attenuated Signals
% shift AttenuatedDGdelay by 1ns?!)
plot(xData,AttenuatedCh1,xData,AttenuatedCh2,xData,AttenuatedDG,xData,
    AttenuatedDGdelay)
title('Channel Alignment');
xlabel('time (ns)');

% FFT of Pulse
Fs = 100;
T = 1/Fs;
L = size(AttenuatedCh1(5000:45000),1);
y = AttenuatedCh1(5000:45000) + AttenuatedCh2(5000:45000) ...
    + AttenuatedDG(5000:45000) + AttenuatedDGdelay(5000:45000);
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);

plot(f,20*log10(abs(Y(1:NFFT/2+1))/max(abs(Y(1:NFFT/2+1)))))
title('Single-Sided Amplitude Spectrum of y(t)')
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')


plot(xData,AttenuatedCh1,xData,AttenuatedCh2,xData,AttenuatedDG,xData,
    circshift(AttenuatedDGdelay,-100))
title('Pattern Spread');
xlabel('time (ns)');
```

```
plot(0.01.*(-1*(size(AttenuatedCh1)-1):(size(AttenuatedCh1))-1), ...
    20*log10(abs(xcorr(AttenuatedCh1-AttenuatedCh2+AttenuatedDG-
        AttenuatedDGdelay)) ...
    /max(xcorr(AttenuatedCh1-AttenuatedCh2+AttenuatedDG-
        AttenuatedDGdelay))));
title('Autocorrelation of Signal Superposition');
xlabel('Delay (ns)');
axis([-512 512 -60 0]);

%% Filter Attenuated Data 5 to 30 ghz (fs = 100 ghz)
% Wp = [5 30]/50; Ws = [3 35]/50;
% Rp = -20*log10(0.9); Rs = 40;
% [n,Wp] = ellipord(Wp,Ws,Rp,Rs);
% [b,a] = ellip(n,Rp,Rs,Wp);
% freqz(b,a,512,100);

%% Att and Rad CrossCorrelation
plot(0.01.*(-1*(size(AttenuatedCh1)-1):(size(AttenuatedCh1))-1), ...
    20*log10(abs(xcorr(AttenuatedCh1-AttenuatedCh2+AttenuatedDG-
        AttenuatedDGdelay,Scatter)) ...
    /max(xcorr(AttenuatedCh1-AttenuatedCh2+AttenuatedDG-
        AttenuatedDGdelay,Scatter))));
title('Autocorrelation of Signal Superposition');
xlabel('Delay (ns)');
axis([-512 512 -60 0]);
```

7.6. MATLAB: Disconnect from oscilloscope/pulsegen

```
%% Disconnect from Oscilloscope and Pulse Gen
% Author: Matt Dwyer

fclose(oscilloscope);
delete(oscilloscope);
clear oscilloscope
fclose(pulseGen);
delete(pulseGen);
clear pulseGen
```

7.7. MATLAB: Measurement workaround simulation verification

```
%% Disconnect from Oscilloscope and Pulse Gen
% Author: Matt Dwyer

fclose(oscilloscope);
```

```
delete(oscilloscope);
clear oscilloscope
fclose(pulseGen);
delete(pulseGen);
clear pulseGen
```