MODELING AN INTERFERENCE-TOLERANT LIDAR SYSTEM

An Undergraduate Research Scholars Thesis

by

DERRICK BRUCE KNOX

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Samuel Palermo

May 2018

Major: Electrical Engineering

TABLE OF CONTENTS

ABSTRACT

Modeling an Interference-Tolerant LIDAR System

Derrick Bruce Knox
Department of Electrical Engineering
Texas A&M University

Research Advisor: Dr. Samuel Palermo
Department of Electrical Engineering
Texas A&M University

LIDAR – essentially laser radar – is a key technology in the emerging field of autonomous vehicles. It allows the vehicle to detect any obstacles around it and calculate its distance from them, which allows it to build a real-time map of its surroundings. Issues arise, however, when multiple LIDAR-equipped vehicles are on the road at the same time, as their transmitted lasers may strike other vehicles' receivers. This interference gives the receiving car a faulty view of its surroundings, which could be dangerous if the LIDAR is being used to help control the car. Our research team found a way to mitigate this issue, using a pseudo-random algorithm to vary the time at which the lasers are sent. This spreads the laser energy around and makes it less likely to create a faulty detection in any cars that may accidentally receive it.

My work on this project was to create a software model of this algorithm to assist with the creation of this hardware, and then to test the functionality of this system in hardware using an actual laser system. To do this, I created and optimized a program in MATLAB for the software portion. Once that software portion was completed and tested to be accurate, a Verilog script was written so that tests could be conducted with actual laser drivers on FPGA boards.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Palermo, for admitting me onto his project and for his direction, Po-Hsuan Chang for providing me with the initial MATLAB model as far as debugging support, and my brother, Dillon Knox, for reminding me of a couple program-critical research deadlines I had forgotten about. None of this research would have been completed without them.

NOMENCLATURE

LFSR	Linear feedback shift register
LIDAR	Light Imaging, Detecting And Ranging
PRBS	Pseudo-random bit sequencing
SNR	Signal-to-noise ratio
SPAD	Single-photon avalanche diode

CHAPTER I

INTRODUCTION

On LIDAR

LIDAR technology allows machines to detect obstacles around them. A laser pulse is sent out, it strikes an object, and some of the laser photons are reflected back. A very small number of these will be reflected straight back to the source.

These can be detected by the use of single-photon avalanche diodes (SPADs), which can detect as little as one photon that strikes them and use that impulse to trigger a much larger electrical signal. A few particles of light can thus be used control a voltage large enough to be measured by a computer. Ideally, a SPAD can be used to detect an object as far as 100 meters away (Richardson).

A timer is started when the laser is fired, and when the SPAD signal is seen, the timer is stopped. Under real-world conditions, light can be assumed to always travel at a constant speed, so by knowing the laser's velocity and the time the laser traveled, the distance from the transmitter to the object can be calculated. This functionality is handled in hardware with a time-to-digital converter connected to a SPAD (Richardson).

The overarching goal of this research project is to develop a LIDAR system that fits on a single IC – making it simpler to add LIDAR systems to vehicles and reducing cost. A rough outline for this system has previously been made (Chen), and the current work on the project is on how to ultimately implement that into hardware. To our knowledge, this has only been done once before (Niclass), and it’s never been done with an interference-reduction feature like ours has, which will be crucial if the technology is to reach a mass scale.

Interference Reduction and Pseudo-Random Binary Sequencing

The concern my team has, and the motivation for this research, is that if LIDAR technology is to become widespread, there may be hundreds of cars on a road, all firing lasers in all directions. This will inevitably end up striking the LIDAR receivers on another vehicle, causing it to create a false reading of seeing an object that isn't truly there – or more dangerously, possibly ignoring an object that is.

To solve this, our team's idea is to pseudo-randomly spread the laser signal around. A pulsed laser will be repeatedly fired, but the time will randomly vary. Sometimes it may be a few fractions of a second ahead of its normal period, sometimes it may be a few fractions of a second behind its normal period. The timers, likewise, start counting when the laser is fired. This allows the energy to be spread around over time and minimize the energy that is sent to other vehicles, while ideally, the clocks still accurately measure the lasers' overall travel time since they are synchronized with all the individual pulses.

The generation of random numbers in hardware is what is known as a pseudo-random binary sequence (PRBS). The output is created by a repeatable mathematical process and is deterministic, but it appears to be random over long stretches. A common way to create these sequences is through linear feedback shift registers (LFSR). A series of digital shift registers with a beginning “seed” bit sequence is implemented. XOR gates are tapped to the outputs of certain registers, and the output of those XORs in turn to the beginning of the sequence. An example of this is shown in figure 1. In this way, the input of the sequence is always being “scrambled”, and that scrambled input is continuously scrambled more as the shifting progresses. With enough bits and an unknown seed, the process becomes unpredictable enough to be considered random.

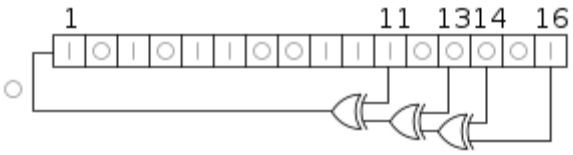

Figure 1. An example of the shifting registers and XOR logic used to create a pseudo-random binary sequence.

MATLAB Simulations

MATLAB models were constructed to simulate the LIDAR laser pulses being sent out, and how they interfered with each other. This let us determine how many photons our pulses could be expected to return from a signal sent out, how much our random time-shifting of pulses reduced interference compared to an unshifted signal, and what degree of randomization we would need to optimize interference reduction while using a minimal amount of hardware.

The model was built up incrementally over time. A rough existing model, using two sources interfering with each other using a Gaussian randomization process created in MATLAB, already existed – this was used as the foundation for the rest of the work. The model originally only had two sources in it – this was expanded to three. After that, a flaw in the model that caused it to have an excessive run time (over one hour per simulation), was located and solved, to reduce the run time to a matter of seconds. Then, pseudo-random bit sequencing was implemented to simulate the actual hardware we would be using, instead of the software-defined Gaussian bell-curve that could not be replicated by a circuit. The model was then verified for accurate function.

Hardware Testing

The hardware portion of the circuit was to be implemented after the MATLAB model was perfected. A Verilog script was created to drive actual, physical laser transmitters on an FPGA board, using the PRBS time-shifting algorithm perfected in MATLAB. These lasers were to be transmitted at targets at a distance, and their reflections would be measured by the devices. By building histograms of the photons received, we were able to see a picture of what the LIDAR machines were detecting. Ideally, these could be used have been used to verify the measurements of the software model.

Unfortunately, the FPGA boards needed were in another place, and we were not able to get them back to College Station before the end of the semester. Instead, the Verilog code was written and tested on software to show that it *would* work, had the proper equipment been available.

CHAPTER II

METHODS

Constructing The Model

Work was began on this project by using a MATLAB model that had been created by a PhD student working on this project, Po-Hsuan Chang. This model modeled two LIDAR beams interfering with each other and plotted the results on a histogram. While functioning, the model had some shortcomings. The pseudo-random time-shifting behavior it was intended to test was not yet implemented and was simulated by using Gaussian bell-curve distributions. In addition, the model had an excessive run time of over an hour per test, which made repeated testing impractical.

The first improvement made to this model was adding a third source to the other two, to help generate more data. This third source was constructed identically to the first two, and its reflections and interference were combined along with the first two sources. Testing showed that this third interferer worked properly.

The next step was to make the model run in a shorter amount of time, since one hour per run made testing very impractical. I created a series of text-printout flags in the script that would display a message when the program got to them (one example was “I am computing the received elements”), and see what parts of the code took the longest to execute.

That slow part of the code was found in a section that computed the reflections for the three different sources. This was a process than ran several nested for-loops, ultimately covering millions of elements. MATLAB software is designed to handle vector calculations very quickly,

but on the flip side, it handles traditional looping operations very slowly, and this was causing the excessive run time.

I was able to find some vector commands within MATLAB that replicated the arithmetic being done within the for-loops, and replaced the loops with these much faster functions. After this, the run time for the model fell from over an hour to about ten seconds.

With the foundation of the model finished, the next step was to implement the pseudo-random shifting algorithm that the simulation was intended to model. An implementation was used to recreate the action of the LFSR – an array of values were used, and values were passed down the chain to higher and higher indices. Some of these indices were compared to each other with a logical XOR function, and those outputs XOR'ed with others according to a user-configurable setting of taps. The output of that logic was fed back in to the beginning of the sequence.

Three separate PRBS sources were added to the model, so that each transmitter source had its own stream of random inputs. The transmitters would take a fixed amount of bits from the sequence – most of the testing was done with 8 bits – and use that to determine how much shifting would occur. The transmit time of the model was backed up by a factor of (2^(number of bits – 1)), and then all the shifting afterwards would be added to that offset. The last bit in the sequence would move the offset ahead by one time period if the bit was 1, or by none if it was zero. The second to last bit would move the offset ahead by 2 if the bit was 1, or by none if it was zero. And so on until the last bit, which would move the sequence ahead by 128 if the bit was 1 or by zero if the bit was 0. At the very most, the sequence could move 255 time periods forward if all 8 bits in the PRBS sequence turned out to be 1.

Given that signal was offset by 128 periods to start with, this gives us a maximum of +127 time periods forward shift. At the other extreme, all the PRBS bits could be zero, the shift would be zero, and that -128 offset would remain, so the signal would be shifted back by 128 time periods. Any shift between -128 and +127 was equally likely given the randomness of the sequence. This allowed us to spread the output and thus the energy of the signal equally over time. The timer of the transmitter would be shifted along with the signal, so it would always record its source accurately. But it sees the energy of all the other sources trying to interfere with it as a smooth, low-power spread across the time domain.

Once the system was proven to be functioning properly, we were able to factor a non-ideality in. The system built previously assumed that signals lost no strength with increasing distance. In reality though, the laser intensity would diffuse over increasing area. It had an elliptical lens with an X-axis half angle of 30 degrees and Y-axis half-angle of, and the diffu. The equation governing this signal strength is shown below in Figure 2. The SPAD has a diameter of 8 um, and rho*phi-LR is a constant equal to 0.7. The laser wavelength is 905 nm.

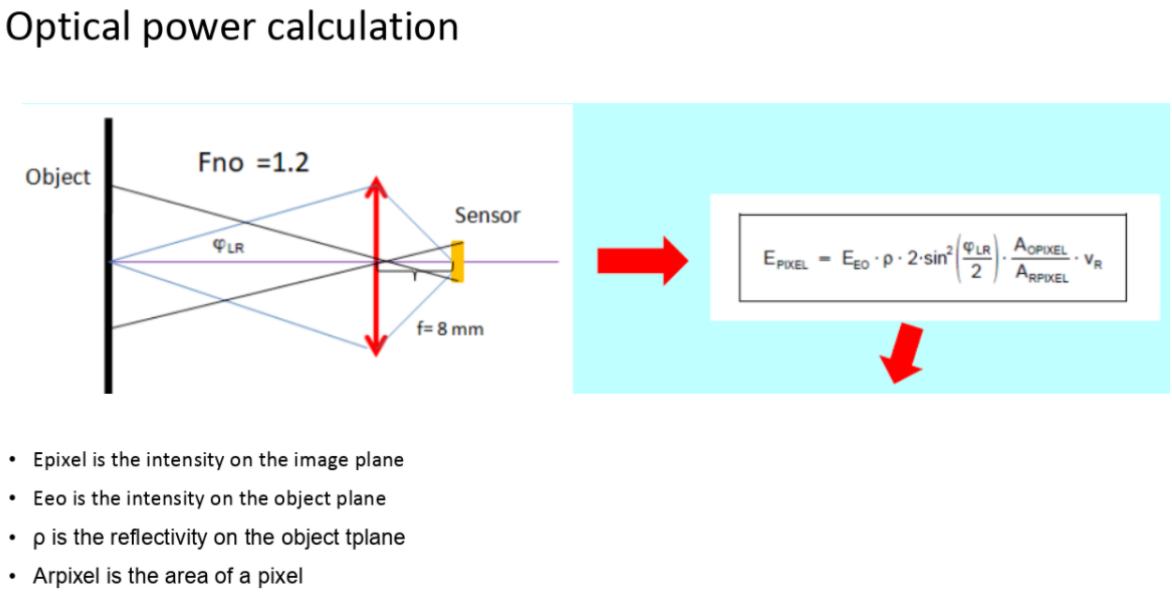

Figure 2. Optical power formula used to calculate SPAD power. Courtesy of Dr. Xiaoge Zeng.

The goal with testing this non-ideal model is to find the distances between a source and an interferer where the source would have 10 dB gain over the interferer.

Testing With The Simulation

The main purpose of the simulation was to find the strength of a LIDAR signal in a variety of conditions. Signal strength was measured in terms of signal-to-noise ratio – the maximum strength of the signal, divided by the maximum strength of the next-highest peak besides the signal, i.e. the maximum of the interference and noise.

To measure the effectiveness of LFSR shifting on reducing interference, the system was tested with shifting in place, and then the exact same code was ran again with the shifting offsets all set to zero, creating static pulses.

Simulations initially had random placement of transmitter sources, and random noise generated for every run, so to average out those parameters, ten trials were conducted and their

means were taken. For the shifting and non-shifting comparisons, the same transmitter location and noise variables were used in both tests to keep the comparisons as close as possible.

For the non-ideal tests factoring in power loss over distance, one of the two interferers was effectively eliminated from the test, and the other one was placed at a fixed distance (eg, 10 m , 20 m, 50 m). The other remaining source was shifted until its signal-to-noise ratio versus the interferer was +10 dB.

Testing With The Hardware

Hardware testing was to be achieved by driving physical lasers mounted on FPGA boards and monitoring the response of SPAD receivers, similar to the design proposed for self-driving cars. The lasers would be pulsed at pseudo-random time intervals, using LFSR shifting simulated with a software algorithm. Their received signals would be recorded and built into a histogram, much like the Matlab model did. In this way, they could be used as experimental verification of the Matlab model.

Unfortunately, the FPGA boards were not available before the end of the semester, so no physical testing could be conducted. A script to drive the FPGA boards was written, and the script was tested on a waveform simulator to prove that it worked as intended. Were the boards available, this script could have been used to successfully drive them.

CHAPTER III

RESULTS

Software Modeling Results

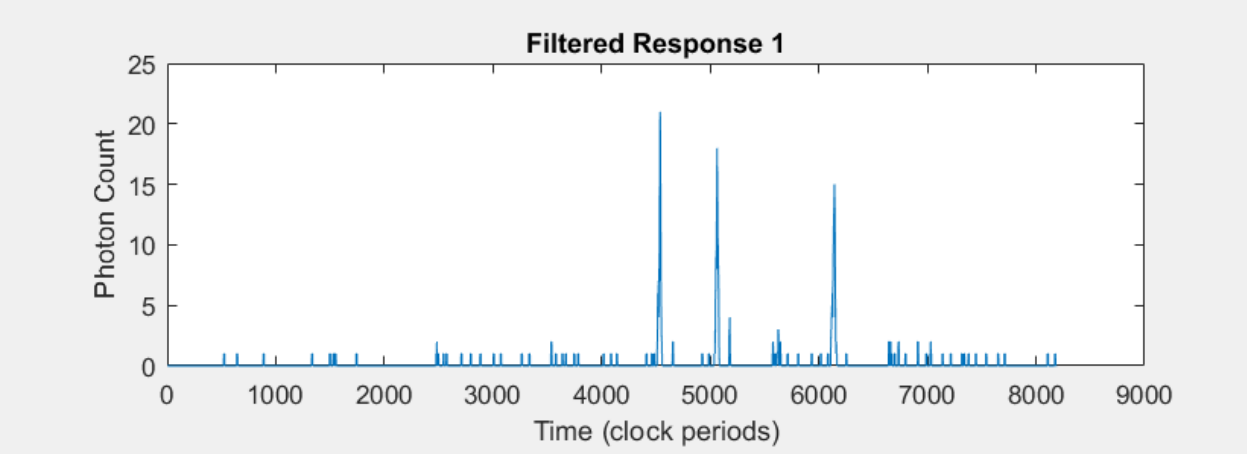

Figure 3. Sample of received signal without PRBS shifting. One source (far right) and two interfering signals.

	Source 1 - No Shifting	Source 2 - No Shifting	Source 3 - No Shifting
dBs	0.526578774	3.098212916	0.965727862
	3.712542713	0.599032841	0.496278653
	0.469987631	0.914281179	1.213956807
	0	0	1.023050449
	0.965727862	0	1.339183957
	1.492672366	2.182792878	1.160164654
	0.496278653	1.023050449	0.526578774
	3.521825181	1.086897775	4.438711997
	0.469987631	1.41183863	0
	1.93820026	2.361986242	1.339183957
Avg	1.453010868	1.322043935	1.331884953

Figure 4: Signal to noise ratio of unshifted signals (dBs)

In figure 3, intended signal to be received (in the center) has no gain over it's two interferers and is actually weaker than the left-most interferer. Figure 4's data demonstrates that the signal and the noise in this situation are practically indistinguishable.

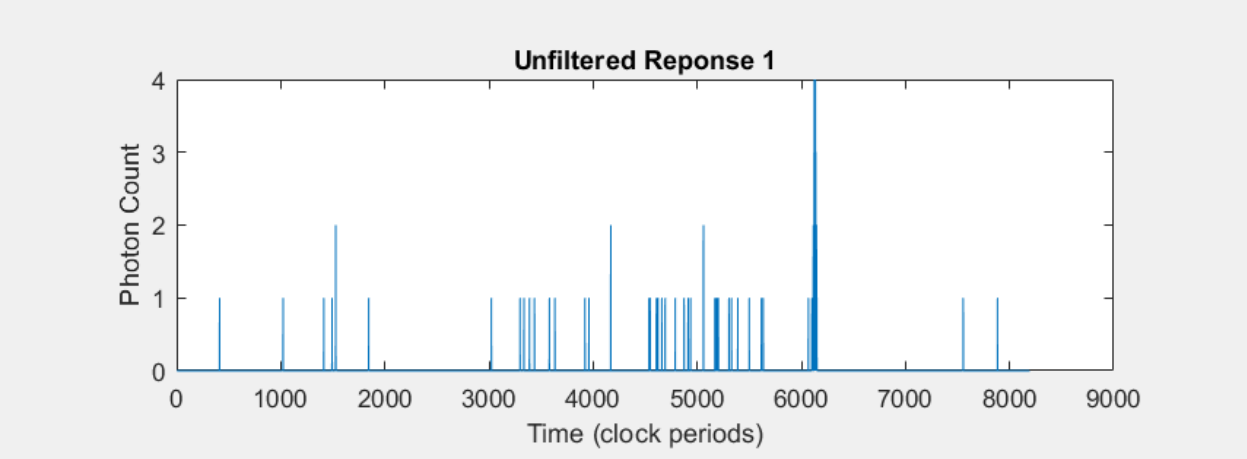

Figure 5. Sample of a received with two interferers, after PRBS shifting. Notice how the energy of the other two signals from Figure 3 has been spread around the time domain.

	Source 1 Peak - Unfiltered	Source 2 Peak - Unfiltered	Source 3 Peak - Unfiltered
Signal-to-Noise Ratios	6.026	6.0206	9.5424
(dBs)	12.0412	9.5424	12.0412
	13.9794	6.0206	15.563
	13.9794	3.5218	9.5424
	12.0412	13.9794	7.9588
	6.206	12.0412	3.5218
	13.9794	12.0412	13.9794
	9.5424	13.9794	12.0412
	9.5424	12.0412	12.0412
	16.902	13.9794	3.5218
Avg SNR (dBs)	11.60553238	10.50089279	9.966201563

Figure 6. Signal to noise ratio of PRBS shifted signals (dBs)

The PRBS shifting dramatically improved the signal-to-noise ratio for the LIDAR output. The intended signal is now clearly readable over the noise, as can be seen in Figure 5.

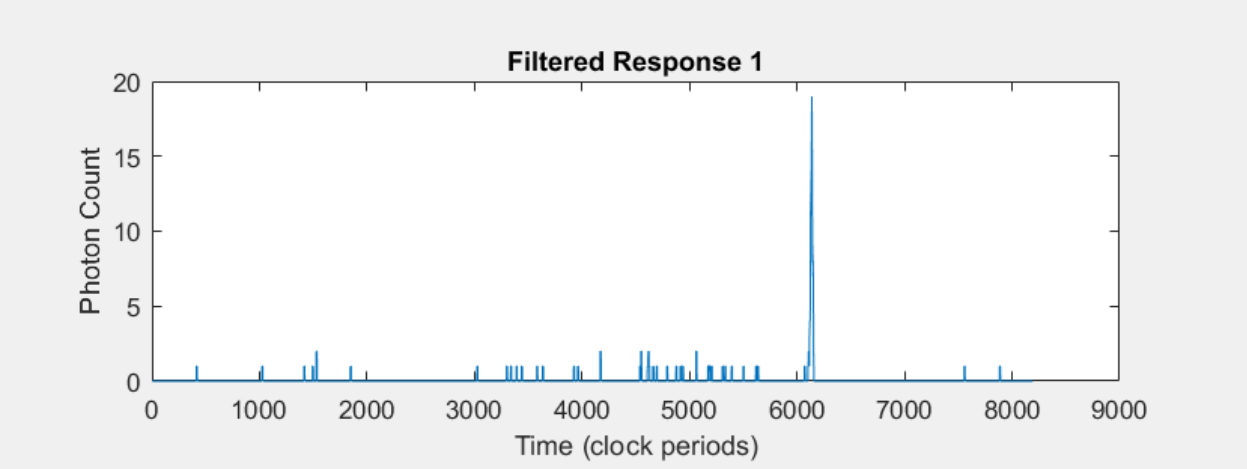

Figure 7. Received signal after PRBS shifting and the application of a digital filter.

	Source 1 Peak - Filtered	Source Peak 2 - Filtered	Source Peak 3 - Filtered
	17.5012	18.5884	13.563
	17.5012	18.5884	15.563
Signal-to-Noise Ratios	17.5012	16.902	17.5012
(dBs)	17.5012	17.5012	17.5012
	17.5012	18.5884	12.5678
	17.5012	18.0618	18.0618
	17.5012	11.2854	15.563
	17.5012	18.5884	16.902
	17.5012	19.0849	19.5545
	18.0618	19.5545	17.5012
Avg SNR (dBs)	16.58605019	16.92263463	16.02120686

Figure 8. Signal-to-noise ratio of a received signal, after PRBS shifting and digital filtering.

Total Avg SNR (dB)	
Filtered	**16.51789072**
Unfiltered	**10.71795454**
No Shifitng	**1.363717235**

Figure 9. Signal-to-noise ratio comparison, between the averages of the signal with filtering and PRBS shifting, the signal with shifting but no filtering, and the unshifted and unfiltered signals.

After applying a digital filter, the data in Figure 8 and Figure 9 shows that the target signal becomes much sharper as the noise is heavily attenuated. It stands above the noise floor by a factor of six. Figure 7 provides visual proof of this.

Inteferer at….	10 dB signal-noise point at…	Discrepancy
10m	6.9m	3.1m (31%)
20m	12.1m	7.9m(39.5%)
30m	23.2m	6.8m(22.7%)
50m	42.4m	7.6m(15.2%)
75m	70.1m	3.9m (5.25%)
90m	85.8m	4.2m (4.6%)

Figure 10. Matlab model testing, attenuation with distance included.

Hardware Testing Results

It was also shown that Verilog script written for the FPGA laser driver would work successfully, although no FPGA was able to be used. This is the waveform output of the laser. Notice how the 8-bit LFSR value is pseudo-randomly changing between 0 and 255, and how the period of the laser firing changes along with it. The Verilog code and testbench used can be found in the appendix.

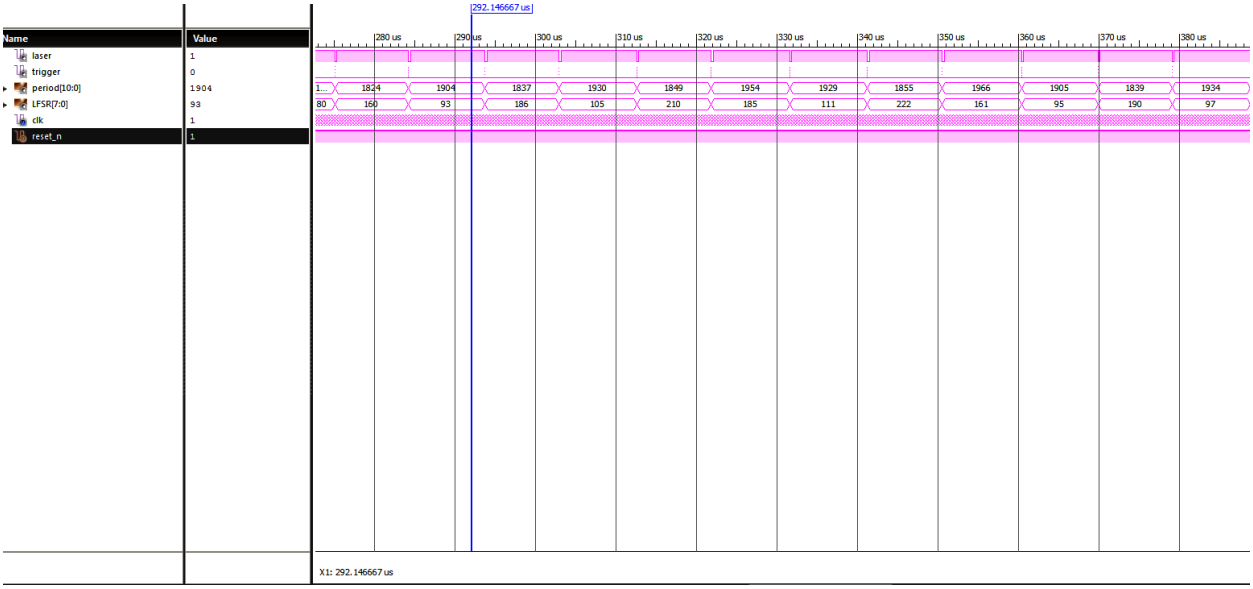

Figure 11. Waveform simulation of FPGA driver Verilog code.

Discussion

The PRBS shifting makes a great difference in reducing interference from two aggressors. Without shifting, the signal barely had any gain over the signals interfering with it. Shifting made three times stronger than the noise, and applying a digital filter created a sixfold gain over the noise.

This model proves that PRBS shifting and filtering is a valid way to reduce interference from nearby LIDAR signals. However, one thing it does not factor in is laser beam spreading over distance – this model assumes that the lasers will lose no strength over distance.

The updated model factored in signal loss over distance. The data in Figure 10 shows that, in general, the LIDAR model can determine an object that is 4-7 meters away from an interfering beam with a +3dB signal gain, all the way out to about 90 meters. 95 meters is the maximum range of the system: the 13-bit hardware time counter overflows beyond that.

Working Verilog code has also been written to test physical hardware with laser drivers, as proven by the test-bench simulation in Figure 11.

CHAPTER IV

CONCLUSION

Overall, I was able to accomplish all my intended goals for this research. The LIDAR model was proven to work successfully and our pseudo-random shifting using linear feedback shift registers was able to reduce interference by as much as a factor of six, compared to a system without PRBS shifting. The system was found to perform well even when non-idealities such as signal attenuation over distance were introduced as well, having a resolution of 4-7 meters of targets up to 90 meters away. And while the laser-driver FPGA boards needed to experimentally verify the Matlab model were unavailable, I was able to write a Verilog script that would have ran them successfully had I been able to use them.

This research paves the way for future work in implementing the system in hardware – at first through an FPGA board testing, and long term, into a tape-out on silicon. The ultimate goal is to build a single-IC LIDAR system utilizing this pseudo-random shifting technology that can be easily and cheaply installed in a vehicle. The convenience of installing such a system, along with a shifting algorithm making operation possible in heavy autonomous vehicle traffic, will be a major step forward in making self-driving vehicles an everyday reality.

REFERENCES

C.Y. Chen, “A Sub-centimeter Ranging Precision LIDAR Sensor Prototype Based on ILO-TDC”, in a thesis submitted to Texas A&M Office of Graduate and Professional Studies, August 2016. Print.

C. Niclass, M. Soga, H. Matsubara, M. Ogawa, and M. Kagami, “A 0.18-µm CMOS SoC for a 100-m-Range 10-Frame/s 200×96-Pixel Time-of-Flight Depth Sensor,” IEEE Journal of Solid-State Circuits, vol. 49, no. 1, pp. 315–330, Jan 2014. Print.

C. Niclass, M. Soga, H. Matsubara, S. Kato, and M. Kagami, “A 100-m Range 10-Frameps 340x96-Pixel Time-of-Flight Depth Sensor in 0.18-µm CMOS,” IEEE Journal of Solid-State Circuits, vol. 48, no. 2, pp. 559–572, Feb 2013.

J. Richardson, R. Walker, L. Grant, D. Stoppa, F. Borghetti, E. Charbon, M. Gersbach, and R. K. Henderson, “A 32×32 50ps resolution 10 bit time to digital converter array in 130nm CMOS for time correlated imaging,” in 2009 IEEE Custom Integrated Circuits Conference, Sept 2009, pp. 77–80. Print.

APPENDIX

Matlab model code

```
% 50khz: 20us (laser period)
% 200MHz: 5ns (TDC coarse clock cycle)
% TDC resolution 78ps as a unit
% A coarse clock cycle has 64 units
% A 13 bit TDC has 2^13= 8192 unit
% A laser period = 4000*64 = 256000 units
% PRBS variation:+- 8 clock period = +- 512 units

clear;clc

clock_period=64;
pulse_period=256000;
pulse_number=50;
tdc_total_lsb=8192;
x1=2430;
x2=2581;
x3=3900;

%x1 = 3900;
%x2 = 50;
%x3 = 100;

pulse_width=64;
photon_number=1000;
noise_number=round( (pulse_number+1)*(20e-
6)*200/(10000*256*2.94e-9) );
pde=0.05;
holdoff_time=512;
word_length=8;

disp(' ');
disp('I am generating an LFSR sequence');


LFSR_Taps = [16 14 13 11];
LFSR_Seed_1 = decimalToBinaryVector(randi(255), 16);
LFSR_Seed_2 = decimalToBinaryVector(randi(255), 16);
LFSR_Seed_3 = decimalToBinaryVector(randi(255), 16);
```

```
PRBS_Sequence_1 = LFSR(LFSR_Seed_1, LFSR_Taps);
PRBS_Sequence_2 = LFSR(LFSR_Seed_2, LFSR_Taps);
PRBS_Sequence_3 = LFSR(LFSR_Seed_3, LFSR_Taps);

if(length(PRBS_Sequence_1) < (pulse_number)+1)
PRBS_Sequence_1 = repmat(PRBS_Sequence_1,
ceil(pulse_number/length(PRBS_Sequence_1)));
PRBS_Sequence_2 = repmat(PRBS_Sequence_2,
ceil(pulse_number/length(PRBS_Sequence_2)));
PRBS_Sequence_3 = repmat(PRBS_Sequence_3,
ceil(pulse_number/length(PRBS_Sequence_3)));
end

signal_1=zeros(1,(pulse_number+1)*pulse_period);
signal_2=zeros(1,(pulse_number+1)*pulse_period);
signal_3=zeros(1,(pulse_number+1)*pulse_period);
noise_1=zeros(1,(pulse_number+1)*pulse_period);
noise_2=zeros(1,(pulse_number+1)*pulse_period);
noise_3=zeros(1,(pulse_number+1)*pulse_period);
start_1=zeros(1,pulse_number);
start_2=zeros(1,pulse_number);
start_3=zeros(1,pulse_number);

disp(' ');
disp('I am creating start arrays');


i = 6;

for m = 1:pulse_number

    if m >= 8
     start_1(m)=start_1(m-1)+pulse_period -
(clock_period*128) + (clock_period*(PRBS_Sequence_1(i-7) +
PRBS_Sequence_1(i-6)*2 + PRBS_Sequence_1(i-5)*4 +
PRBS_Sequence_1(i-4)*8 + PRBS_Sequence_1(i-3)*16 +
PRBS_Sequence_1(i-2)*32 + PRBS_Sequence_1(i-1)*64 +
PRBS_Sequence_1(i)*128 ));
     start_2(m)=start_2(m-1)+pulse_period -
(clock_period*128) + (clock_period*(PRBS_Sequence_2(i-7) +
PRBS_Sequence_2(i-6)*2 + PRBS_Sequence_2(i-5)*4 +
PRBS_Sequence_2(i-4)*8 + PRBS_Sequence_2(i-3)*16 +
PRBS_Sequence_2(i-2)*32 + PRBS_Sequence_2(i-1)*64 +
PRBS_Sequence_2(i)*128 ));
```

```
     start_3(m)=start_3(m-1)+pulse_period -
(clock_period*128) + (clock_period*(PRBS_Sequence_3(i-7) +
PRBS_Sequence_3(i-6)*2 + PRBS_Sequence_3(i-5)*4 +
PRBS_Sequence_3(i-4)*8 + PRBS_Sequence_3(i-3)*16 +
PRBS_Sequence_3(i-2)*32 + PRBS_Sequence_3(i-1)*64 +
PRBS_Sequence_3(i)*128 ));
     i = i + 8;

    elseif m ==7
     start_1(m)=start_1(m-1)+pulse_period -
(clock_period*64) + (clock_period*(PRBS_Sequence_1(i-6) +
PRBS_Sequence_1(i-5)*2 + PRBS_Sequence_1(i-4)*4 +
PRBS_Sequence_1(i-3)*8 + PRBS_Sequence_1(i-2)*16 +
PRBS_Sequence_1(i-1)*32 + PRBS_Sequence_1(i)*64));
     start_2(m)=start_2(m-1)+pulse_period -
(clock_period*64) + (clock_period*(PRBS_Sequence_2(i-6) +
PRBS_Sequence_2(i-5)*2 + PRBS_Sequence_2(i-4)*4 +
PRBS_Sequence_2(i-3)*8 + PRBS_Sequence_2(i-2)*16 +
PRBS_Sequence_2(i-1)*32 + PRBS_Sequence_2(i)*64));
     start_3(m)=start_3(m-1)+pulse_period -
(clock_period*64) + (clock_period*(PRBS_Sequence_3(i-6) +
PRBS_Sequence_3(i-5)*2 + PRBS_Sequence_3(i-4)*4 +
PRBS_Sequence_3(i-3)*8 + PRBS_Sequence_3(i-2)*16 +
PRBS_Sequence_3(i-1)*32 + PRBS_Sequence_3(i)*64));
     i = i + 7;

    elseif m==6
     start_1(m)=start_1(m-1)+pulse_period -
(clock_period*32) + (clock_period*(PRBS_Sequence_1(i-5) +
PRBS_Sequence_1(i-4)*2 + PRBS_Sequence_1(i-3)*4 +
PRBS_Sequence_1(i-2)*8 + PRBS_Sequence_1(i-1)*16 +
PRBS_Sequence_1(i)*32));
     start_2(m)=start_2(m-1)+pulse_period -
(clock_period*32) + (clock_period*(PRBS_Sequence_2(i-5) +
PRBS_Sequence_2(i-4)*2 + PRBS_Sequence_2(i-3)*4 +
PRBS_Sequence_2(i-2)*8 + PRBS_Sequence_2(i-1)*16 +
PRBS_Sequence_2(i)*32));
     start_3(m)=start_3(m-1)+pulse_period -
(clock_period*32) + (clock_period*(PRBS_Sequence_3(i-5) +
PRBS_Sequence_3(i-4)*2 + PRBS_Sequence_3(i-3)*4 +
PRBS_Sequence_3(i-2)*8 + PRBS_Sequence_3(i-1)*16 +
PRBS_Sequence_3(i)*32));
     i = i + 6;
```

```
elseif m==5
         start_1(m)=start_1(m-1)+pulse_period -
clock_period*16 + clock_period*(PRBS_Sequence_1(i-4) +
PRBS_Sequence_1(i-3)*2 + PRBS_Sequence_1(i-2)*4 +
PRBS_Sequence_1(i-1)*8 + PRBS_Sequence_1(i)*16);
         start_2(m)=start_2(m-1)+pulse_period -
clock_period*16 + clock_period*(PRBS_Sequence_2(i-4) +
PRBS_Sequence_2(i-3)*2 + PRBS_Sequence_2(i-2)*4 +
PRBS_Sequence_2(i-1)*8 + PRBS_Sequence_2(i)*16);
         start_3(m)=start_3(m-1)+pulse_period -
clock_period*16 + clock_period*(PRBS_Sequence_3(i-4) +
PRBS_Sequence_3(i-3)*2 + PRBS_Sequence_3(i-2)*4 +
PRBS_Sequence_3(i-1)*8 + PRBS_Sequence_3(i)*16);
         i = i + 1;


elseif m==4
        start_1(m)=start_1(m-1)+pulse_period -
clock_period*8 + clock_period*(PRBS_Sequence_1(i-3) +
PRBS_Sequence_1(i-2)*2 + PRBS_Sequence_1(i-1)*4 +
PRBS_Sequence_1(i)*8);
        start_2(m)=start_2(m-1)+pulse_period -
clock_period*8 + clock_period*(PRBS_Sequence_2(i-3) +
PRBS_Sequence_2(i-2)*2 + PRBS_Sequence_2(i-1)*4 +
PRBS_Sequence_2(i)*8);
        start_3(m)=start_3(m-1)+pulse_period -
clock_period*8 + clock_period*(PRBS_Sequence_3(i-3) +
PRBS_Sequence_3(i-2)*2 + PRBS_Sequence_3(i-1)*4 +
PRBS_Sequence_3(i)*8);
        i = i + 1;

elseif m==3
       start_1(m)=start_1(m-1)+pulse_period -
clock_period*4 + clock_period*(PRBS_Sequence_1(i-2) +
PRBS_Sequence_1(i-1)*2 + PRBS_Sequence_1(i)*4);
       start_2(m)=start_2(m-1)+pulse_period -
clock_period*4 + clock_period*(PRBS_Sequence_2(i-2) +
PRBS_Sequence_2(i-1)*2 + PRBS_Sequence_2(i)*4);
       start_3(m)=start_3(m-1)+pulse_period -
clock_period*4 + clock_period*(PRBS_Sequence_3(i-2) +
PRBS_Sequence_3(i-1)*2 + PRBS_Sequence_3(i)*4);
       i = i + 1;

elseif m==2
```

```
       start_1(m)=start_1(m-1)+pulse_period -
clock_period*2 + clock_period*(PRBS_Sequence_1(i-1) +
PRBS_Sequence_1(i)*2);
       start_2(m)=start_2(m-1)+pulse_period -
clock_period*2 + clock_period*(PRBS_Sequence_2(i-1) +
PRBS_Sequence_2(i)*2);
       start_3(m)=start_3(m-1)+pulse_period -
clock_period*2 + clock_period*(PRBS_Sequence_3(i-1) +
PRBS_Sequence_3(i)*2);
       i = i + 1;

    else

    start_1(m)=pulse_period;
    start_2(m)=pulse_period;
    start_3(m)=pulse_period;


%     start_1(m)=pulse_period;
%     start_2(m)=pulse_period;
%     start_3(m)=pulse_period;
    end
end

disp(' ');
disp('I am creating temp photons');

for a1=1:pulse_number
   for b1=1:photon_number

photon_temp_1=round(normrnd(start_1(a1),pulse_width/2.355))
;
       if photon_temp_1>0

signal_1(photon_temp_1)=signal_1(photon_temp_1)+1;
       end


photon_temp_2=round(normrnd(start_2(a1),pulse_width/2.355))
;
       if photon_temp_2>0

signal_2(photon_temp_2)=signal_2(photon_temp_2)+1;
       end
```

```
photon_temp_3=round(normrnd(start_3(a1),pulse_width/2.355))
;
       if photon_temp_3>0

signal_3(photon_temp_3)=signal_3(photon_temp_3)+1;
       end

   end
end

disp(' ');
disp('I am creating noise vectors');

for e1=1:noise_number
    noise_index_1=unidrnd((pulse_number+1)*pulse_period);
    noise_1(noise_index_1)=noise_1(noise_index_1)+1;

    noise_index_2=unidrnd((pulse_number+1)*pulse_period);
    noise_2(noise_index_2)=noise_2(noise_index_2)+1;

    noise_index_3=unidrnd((pulse_number+1)*pulse_period);
    noise_3(noise_index_3)=noise_3(noise_index_3)+1;
end

disp(' ');
disp('I am computing reflections');

reflect_1to1=[zeros(1,2*x1)    signal_1(1 :
(pulse_number+1)*pulse_period-(2*x1))];
reflect_1to2=[zeros(1,x1+x2) signal_1(1 :
(pulse_number+1)*pulse_period-(x1+x2))];
reflect_1to3=[zeros(1,x1+x3) signal_1(1 :
(pulse_number+1)*pulse_period-(x1+x3))];
reflect_2to1=[zeros(1,x1+x2) signal_2(1 :
(pulse_number+1)*pulse_period-(x1+x2))];
reflect_2to2=[zeros(1,2*x2)    signal_2(1 :
(pulse_number+1)*pulse_period-(2*x2))];
reflect_2to3=[zeros(1,x2+x3)    signal_2(1 :
(pulse_number+1)*pulse_period-(x2+x3))];
reflect_3to1=[zeros(1,x1+x3)    signal_3(1 :
(pulse_number+1)*pulse_period-(x1+x3))];
reflect_3to2=[zeros(1,x2+x3)    signal_3(1 :
(pulse_number+1)*pulse_period-(x2+x3))];
```

```
reflect_3to3=[zeros(1,2*x3)    signal_3(1 :
(pulse_number+1)*pulse_period-(2*x3))];

% reflect_1to2=zeros(1,(pulse_number+1)*pulse_period);
% reflect_2to1=zeros(1,(pulse_number+1)*pulse_period);

combine_1=reflect_1to1+reflect_2to1+reflect_3to1;
combine_2=reflect_1to2+reflect_2to2+reflect_3to2;
combine_3=reflect_1to3+reflect_2to3+reflect_3to3;

% received_1=zeros(1,(pulse_number+1)*pulse_period);
% received_2=zeros(1,(pulse_number+1)*pulse_period);
% received_3=zeros(1,(pulse_number+1)*pulse_period);

% for c1=1:(pulse_number+1)*pulse_period
% received_1(c1)=binornd(1,pde*combine_1(c1))+noise_1(c1);
%
% received_2(c1)=binornd(1,pde*combine_2(c1))+noise_2(c1);
%
%
received_3(c1)=binornd(1,pde*combine_3(c1))+noise_3(c1);
% end

disp(' ');
disp('I am computing the received elements');

%received_1 =
binornd(1,pde*combine_1,1,((pulse_number+1)*pulse_period))
+ noise_1;
%received_2 =
binornd(1,pde*combine_2,1,((pulse_number+1)*pulse_period))
+ noise_2;
%received_3 =
binornd(1,pde*combine_3,1,((pulse_number+1)*pulse_period))
+ noise_3;

received_1 =
binornd(1,pde*combine_1,1,((pulse_number+1)*pulse_period));
received_2 =
binornd(1,pde*combine_2,1,((pulse_number+1)*pulse_period));
received_3 =
binornd(1,pde*combine_3,1,((pulse_number+1)*pulse_period));


received_1_holdoff=zeros(1,(pulse_number+1)*pulse_period);
```

```
received_2_holdoff=zeros(1,(pulse_number+1)*pulse_period);
received_3_holdoff=zeros(1,(pulse_number+1)*pulse_period);
d1=1;
d2=1;
d3=1;

while d1<=(pulse_number+1)*pulse_period
   if(received_1(d1))>0
       received_1_holdoff(d1)=1;
       d1=d1+holdoff_time;
   else
       d1=d1+1;
   end
end

while d2<=(pulse_number+1)*pulse_period
   if(received_2(d2))>0
       received_2_holdoff(d2)=1;
       d2=d2+holdoff_time;
   else
       d2=d2+1;
   end
end

while d3<=(pulse_number+1)*pulse_period
   if(received_3(d3))>0
       received_3_holdoff(d3)=1;
       d3=d3+holdoff_time;
   else
       d3=d3+1;
   end
end

hist_1=zeros(1,tdc_total_lsb);
hist_2=zeros(1,tdc_total_lsb);
hist_3=zeros(1,tdc_total_lsb);

for m1=1:pulse_number

for n1= start_1(m1) : start_1(m1)+tdc_total_lsb

   if received_1_holdoff(n1)>0 && n1-start_1(m1)>0
       hist_1(n1-start_1(m1))=hist_1(n1-start_1(m1))+1;
   end
```

```
end

for n2= start_2(m1) : start_2(m1)+tdc_total_lsb

   if received_2_holdoff(n2)>0 && n1-start_2(m1)>0
      hist_2(n2-start_2(m1))=hist_2(n2-start_2(m1))+1;
   end

end

for n3= start_3(m1) : start_3(m1)+tdc_total_lsb

   if received_3_holdoff(n3)>0 && n3-start_3(m1)>0
       hist_3(n3-start_3(m1))=hist_3(n3-start_3(m1))+1;
   end

end

end



moving_average=ones(1,word_length);

response_1=filter(moving_average,1,hist_1);
response_2=filter(moving_average,1,hist_2);
response_3=filter(moving_average,1,hist_3);

[Ma1 In1]=max(response_1);
[Ma2 In2]=max(response_2);
[Ma3 In3]=max(response_3);

%Correct peak check
%[1st peak correct, 2nd peak correct, wrong peak that might
happen, correct
%1st peak index, correct 2nd peak index
[2*x1 2*x2 2*x3 In1 In2 In3]



figure
subplot(3,2,1)
plot(hist_1);
```

```
xlabel('Time (clock periods)')
ylabel('Photon Count')
title('Unfiltered Reponse 1')
[max_filtered_1_shifting_unfiltered MI_1_U] = max(hist_1);
max_noisefloor_source_1_unfiltered =
max(max(hist_1(1:(MI_1_U-
200))),max((hist_1(MI_1_U+200:8096))));
signalnoise_1_shifting_unfiltered
=(max_filtered_1_shifting_unfiltered /
max_noisefloor_source_1_unfiltered);

subplot(3,2,2)
plot(response_1);
xlabel('Time (clock periods)')
ylabel('Photon Count')
title('Filtered Response 1')
[max_filtered_1_shifting MI_1] = max(response_1);
max_noisefloor_source_1 = max(max(response_1(1:(2*x1-
100))),max((response_1(2*x1+100:8096))));
signalnoise_1_shifting = (max_filtered_1_shifting /
max_noisefloor_source_1);

subplot(3,2,3)
plot(hist_2);
xlabel('Distance')
ylabel('Photon Count')
title('Unfiltered Response 2')
[max_filtered_2_shifting_unfiltered MI_2_U] = max(hist_2);
max_noisefloor_source_2_unfiltered =
max(max(hist_2(1:(MI_2_U-
10))),max((hist_2(MI_2_U+200:8096))));
signalnoise_2_shifting_unfiltered =
(max_filtered_2_shifting_unfiltered /
max_noisefloor_source_2_unfiltered);

subplot(3,2,4)
plot(response_2);
xlabel('Distance')
ylabel('Photon Count')
title('Filtered Response 2')
[max_filtered_2_shifting MI_2] = max(response_2);
max_noisefloor_source_2 = max(max(response_2(1:(MI_2-
10))),max((response_2(MI_2+200:8096))));
```

```
signalnoise_2_shifting = (max_filtered_2_shifting /
max_noisefloor_source_2);


subplot(3,2,5)
plot(hist_3);
xlabel('Distance')
ylabel('Photon Count')
title('Unfiltered Response 3')
[max_filtered_3_shifting_unfiltered MI_3_U] = max(hist_3);
max_noisefloor_source_3_unfiltered =
max(max(response_2(1:(MI_3_U-
10))),max((response_2(MI_3_U+200:8096))));
signalnoise_3_shifting_unfiltered =
(max_filtered_3_shifting_unfiltered /
max_noisefloor_source_3_unfiltered);


subplot(3,2,6)
plot(response_3);
xlabel('Distance')
ylabel('Photon Count')
title('Filtered Response 3')
[max_filtered_3_shifting MI_3] = max(response_3);
max_noisefloor_source_3 = max(max(response_3(1:(MI_3-
10))),max((response_3(MI_3+200:8096))));
signalnoise_3_shifting = (max_filtered_3_shifting /
max_noisefloor_source_3);



%photon energy = h * c / lambda
%lambda = 905 nm per Xiaoge's model

photon_energy =(6.626E-34)*(2.98E8) / (905E-9);

relectivity_transmission_coeff = 0.7;

SPAD_area = pi * 4E-3 * 4E-3;

power_plot_1 = hist_1;

for i = 1:tdc_total_lsb

    %area of the object reflecting off of is an ellipse
```

```
    %area = pi * a * b
    %the x axis of the lens is at a 10 degree angle
    %the y axis of the lens is at a 30 degree angle

    power_plot_1(i) = (hist_1(i) * photon_energy *
2*sind(10) * sind(10) * relectivity_transmission_coeff /
SPAD_area) / (pi * ((tand(10) * 0.5*i * (2.98E8 * 78E-12))
* (tand(30)*0.5*i*(2.98E8* 78E-12))));


response_powerplot=filter(moving_average,1,power_plot_1);

    max_signal_powerplot = max(max(response_powerplot(2*x1-
50:2*x1),max(response_powerplot(2*x1:2*x1+50))));
    noise_powerplot = max(max(response_powerplot(2*x2-
200:2*x2),max(response_powerplot(2*x2:2*x2+200))));
    powerplot_SNR = max_signal_powerplot/noise_powerplot


end

figure
xaxis = (1:1:8192);
xaxis = xaxis *(78E-12*0.5*2.98E8);
plot(xaxis, power_plot_1);
xlabel('Distance (m)');
ylabel('Power(W)');









%
%
%
%
%
```

```
% This code was used to generate the non-random, non-
%distance affected code for SNR sims.
% %lazy copy paste below
%
% % 50khz: 20us (laser period)
% % 200MHz: 5ns (TDC coarse clock cycle)
% % TDC resolution 78ps as a unit
% % A coarse clock cycle has 64 units
% % A 13 bit TDC has 2^13= 8192 unit
% % A laser period = 4000*64 = 256000 units
% % PRBS variation:+- 8 clock period = +- 512 units
%
% % clock_period=64;
% % pulse_period=256000;
% % pulse_number=200;
% % tdc_total_lsb=8192;
% % x1=unidrnd(4096);
% % x2=unidrnd(4096);
% % x3=unidrnd(4096);
% %
% % pulse_width=64;
% % photon_number=1000;
% % noise_number=round( (pulse_number+1)*(20e-
6)*200/(10000*256*2.94e-9) );
% % pde=0.05;
% % holdoff_time=128;
% % word_length=8;
% %
% % disp(' ');
% % disp('I am generating an LFSR sequence');
% %
% %
% % LFSR_Taps = [8 6 5 4];
% % LFSR_Seed_1 = decimalToBinaryVector(randi(63), 8);
% % LFSR_Seed_2 = decimalToBinaryVector(randi(63), 8);
% % LFSR_Seed_3 = decimalToBinaryVector(randi(63), 8);
% %
% %
% % PRBS_Sequence_1 = LFSR(LFSR_Seed_1, LFSR_Taps);
% % PRBS_Sequence_2 = LFSR(LFSR_Seed_2, LFSR_Taps);
% % PRBS_Sequence_3 = LFSR(LFSR_Seed_3, LFSR_Taps);
% %
% % if(length(PRBS_Sequence_1) < (pulse_number)+1)
```

```
% % PRBS_Sequence_1 = repmat(PRBS_Sequence_1,
ceil(pulse_number/length(PRBS_Sequence_1)));
% % PRBS_Sequence_2 = repmat(PRBS_Sequence_2,
ceil(pulse_number/length(PRBS_Sequence_2)));
% % PRBS_Sequence_3 = repmat(PRBS_Sequence_3,
ceil(pulse_number/length(PRBS_Sequence_3)));
% % end
%
% signal_1=zeros(1,(pulse_number+1)*pulse_period);
% signal_2=zeros(1,(pulse_number+1)*pulse_period);
% signal_3=zeros(1,(pulse_number+1)*pulse_period);
% %noise_1=zeros(1,(pulse_number+1)*pulse_period);
% %noise_2=zeros(1,(pulse_number+1)*pulse_period);
% %noise_3=zeros(1,(pulse_number+1)*pulse_period);
% start_1=zeros(1,pulse_number);
% start_2=zeros(1,pulse_number);
% start_3=zeros(1,pulse_number);

% disp(' ');
% disp('I am creating start arrays');
%
% for m=1:pulse_number
%     if m>=2
%     start_1(m)=start_1(m-1)+pulse_period;
%     start_2(m)=start_2(m-1)+pulse_period;
%     start_3(m)=start_3(m-1)+pulse_period;
%
%     %start_1(m)=start_1(m-1)+pulse_period-
clock_period*8+clock_period*unidrnd(16);
%     %start_2(m)=start_2(m-1)+pulse_period-
clock_period*8+clock_period*unidrnd(16);
%     %start_3(m)=start_3(m-1)+pulse_period-
clock_period*8+clock_period*unidrnd(16);
% %     start_1(m)=start_1(m-1)+pulse_period;
% %     start_2(m)=start_2(m-1)+pulse_period;
% %     start_3(m)=start_3(m-1)+pulse_period;
%     else
%     start_1(m)=pulse_period-512;
%     start_2(m)=pulse_period-512;
%     start_3(m)=pulse_period-512;
% %     start_1(m)=pulse_period;
% %     start_2(m)=pulse_period;
% %     start_3(m)=pulse_period;
%     end
```

```
% end
%
% disp(' ');
% disp('I am creating temp photons');
%
% for a1=1:pulse_number
%    for b1=1:photon_number
%
photon_temp_1=round(normrnd(start_1(a1),pulse_width/2.355))
;
%        if photon_temp_1>0
%
signal_1(photon_temp_1)=signal_1(photon_temp_1)+1;
%        end
%
%
photon_temp_2=round(normrnd(start_2(a1),pulse_width/2.355))
;
%        if photon_temp_2>0
%
signal_2(photon_temp_2)=signal_2(photon_temp_2)+1;
%        end
%
%
photon_temp_3=round(normrnd(start_3(a1),pulse_width/2.355))
;
%        if photon_temp_3>0
%
signal_3(photon_temp_3)=signal_3(photon_temp_3)+1;
%        end
%
%    end
% end
%
% disp(' ');
% disp('I am creating noise vectors');
%
% for e1=1:noise_number
%     noise_index_1=unidrnd((pulse_number+1)*pulse_period);
%     noise_1(noise_index_1)=noise_1(noise_index_1)+1;
%
%     noise_index_2=unidrnd((pulse_number+1)*pulse_period);
%     noise_2(noise_index_2)=noise_2(noise_index_2)+1;
%
```

```
%     noise_index_3=unidrnd((pulse_number+1)*pulse_period);
%     noise_3(noise_index_3)=noise_3(noise_index_3)+1;
% end
%
% disp(' ');
% disp('I am computing reflections');
%
% reflect_1to1=[zeros(1,2*x1)    signal_1(1 :
(pulse_number+1)*pulse_period-(2*x1))];
% reflect_1to2=[zeros(1,x1+x2) signal_1(1 :
(pulse_number+1)*pulse_period-(x1+x2))];
% reflect_1to3=[zeros(1,x1+x3) signal_1(1 :
(pulse_number+1)*pulse_period-(x1+x3))];
% reflect_2to1=[zeros(1,x1+x2) signal_2(1 :
(pulse_number+1)*pulse_period-(x1+x2))];
% reflect_2to2=[zeros(1,2*x2)    signal_2(1 :
(pulse_number+1)*pulse_period-(2*x2))];
% reflect_2to3=[zeros(1,x2+x3)    signal_2(1 :
(pulse_number+1)*pulse_period-(x2+x3))];
% reflect_3to1=[zeros(1,x1+x3)    signal_3(1 :
(pulse_number+1)*pulse_period-(x1+x3))];
% reflect_3to2=[zeros(1,x2+x3)    signal_3(1 :
(pulse_number+1)*pulse_period-(x2+x3))];
% reflect_3to3=[zeros(1,2*x3)    signal_3(1 :
(pulse_number+1)*pulse_period-(2*x3))];
%
% % reflect_1to2=zeros(1,(pulse_number+1)*pulse_period);
% % reflect_2to1=zeros(1,(pulse_number+1)*pulse_period);
%
% combine_1=reflect_1to1+reflect_2to1+reflect_3to1;
% combine_2=reflect_1to2+reflect_2to2+reflect_3to2;
% combine_3=reflect_1to3+reflect_2to3+reflect_3to3;
%
% % received_1=zeros(1,(pulse_number+1)*pulse_period);
% % received_2=zeros(1,(pulse_number+1)*pulse_period);
% % received_3=zeros(1,(pulse_number+1)*pulse_period);
%
% % for c1=1:(pulse_number+1)*pulse_period
% %
received_1(c1)=binornd(1,pde*combine_1(c1))+noise_1(c1);
% %
% %
received_2(c1)=binornd(1,pde*combine_2(c1))+noise_2(c1);
% %
```

```
% %
received_3(c1)=binornd(1,pde*combine_3(c1))+noise_3(c1);
% % end
%
% disp(' ');
% disp('I am computing the received elements');
%
% received_1 =
binornd(1,pde*combine_1,1,((pulse_number+1)*pulse_period));
% received_2 =
binornd(1,pde*combine_2,1,((pulse_number+1)*pulse_period));
% received_3 =
binornd(1,pde*combine_3,1,((pulse_number+1)*pulse_period));
%
%
%
received_1_holdoff=zeros(1,(pulse_number+1)*pulse_period);
%
received_2_holdoff=zeros(1,(pulse_number+1)*pulse_period);
%
received_3_holdoff=zeros(1,(pulse_number+1)*pulse_period);
% d1=1;
% d2=1;
% d3=1;
%
% while d1<=(pulse_number+1)*pulse_period
%    if(received_1(d1))>0
%        received_1_holdoff(d1)=1;
%        d1=d1+holdoff_time;
%    else
%        d1=d1+1;
%    end
% end
%
% while d2<=(pulse_number+1)*pulse_period
%    if(received_2(d2))>0
%        received_2_holdoff(d2)=1;
%        d2=d2+holdoff_time;
%    else
%        d2=d2+1;
%    end
% end
%
% while d3<=(pulse_number+1)*pulse_period
```

```
%    if(received_3(d3))>0
%        received_3_holdoff(d3)=1;
%        d3=d3+holdoff_time;
%    else
%        d3=d3+1;
%    end
% end
%
% hist_1=zeros(1,tdc_total_lsb);
% hist_2=zeros(1,tdc_total_lsb);
% hist_3=zeros(1,tdc_total_lsb);
%
% for m1=1:pulse_number
%
% for n1= start_1(m1) : start_1(m1)+tdc_total_lsb
%
%    if received_1_holdoff(n1)>0 && n1-start_1(m1)>0
%        hist_1(n1-start_1(m1))=hist_1(n1-start_1(m1))+1;
%    end
%
% end
%
% for n2= start_2(m1) : start_2(m1)+tdc_total_lsb
%
%    if received_2_holdoff(n2)>0 && n1-start_2(m1)>0
%       hist_2(n2-start_2(m1))=hist_2(n2-start_2(m1))+1;
%    end
%
% end
%
% for n3= start_3(m1) : start_3(m1)+tdc_total_lsb
%
%    if received_3_holdoff(n3)>0 && n3-start_3(m1)>0
%        hist_3(n3-start_3(m1))=hist_3(n3-start_3(m1))+1;
%    end
%
% end
%
% end
%
%
%
%
% moving_average=ones(1,word_length);
```

```
%
% response_1=filter(moving_average,1,hist_1);
% response_2=filter(moving_average,1,hist_2);
% response_3=filter(moving_average,1,hist_3);
%
% [Ma1 In1]=max(response_1);
% [Ma2 In2]=max(response_2);
% [Ma3 In3]=max(response_3);
%
% %Correct peak check
% %[1st peak correct, 2nd peak correct, wrong peak that
might happen, correct
% %1st peak index, correct 2nd peak index
% [2*x1 2*x2 2*x3 In1 In2 In3]
%
%
%
%
% figure
% subplot(3,2,1)
% plot(hist_1);
% xlabel('Distance')
% ylabel('Photon Count')
% title('Unfiltered Reponse 1')
%
% subplot(3,2,2)
% plot(response_1);
% xlabel('Time (clock periods)')
% ylabel('Photon Count')
% title('Filtered Response 1')
% [max_1_noshifting MI_1] = max(response_1);
% max_noisefloor_1_noshifting = max(max(response_1(1:(MI_1-
50))),max((response_1(MI_1+50:8096))));
% signalnoise_1_noshifting = (max_1_noshifting /
max_noisefloor_1_noshifting);
%
%
% subplot(3,2,3)
% plot(hist_2);
% xlabel('Distance')
% ylabel('Photon Count')
% title('Unfiltered Response 2')
%
% subplot(3,2,4)
```

```
% plot(response_2);
% xlabel('Distance')
% ylabel('Photon Count')
% title('Filtered Response 2')
% [max_2_noshifting MI_2] = max(response_2);
% max_noisefloor_2_noshifting = max(max(response_2(1:(MI_2-
50))),max((response_2(MI_2+50:8096))));
% signalnoise_2_noshifting = (max_2_noshifting /
max_noisefloor_2_noshifting);
%
% subplot(3,2,5)
% plot(hist_3);
% xlabel('Distance')
% ylabel('Photon Count')
% title('Unfiltered Response 3')
%
% subplot(3,2,6)
% plot(response_3);
% xlabel('Distance')
% ylabel('Photon Count')
% title('Filtered Response 3')
% [max_3_noshifting MI_3] = max(response_3);
%
%
% max_noisefloor_3_noshifting = max(max(response_3(1:(MI_3-
50))),max((response_3(MI_3+50:8096))));
% signalnoise_3_noshifting = (max_3_noshifting /
max_noisefloor_3_noshifting);
%
% disp('#1 No Shifting');
% disp(signalnoise_1_noshifting);
%
% disp('#2 No Shifting');
% disp(signalnoise_2_noshifting);
%
% disp('#3 No Shifting');
% disp(signalnoise_3_noshifting);
%
disp('#1 Shifting');
disp(signalnoise_1_shifting);

%disp('#1 Shifting Unfiltered');
%disp(signalnoise_1_shifting_unfiltered);
%
```

```
% disp('#2 Shifting');
% disp(signalnoise_2_shifting);
%
% disp('#2 Shifting Unfiltered');
% disp(signalnoise_2_shifting_unfiltered);
%
% disp('#3 Shifting');
% disp(signalnoise_3_shifting);
%
% disp('#3 Shifting Unfiltered');
% disp(signalnoise_3_shifting_unfiltered);
```

PRBS Verilog Code

```
`timescale 1ns / 1ps

module laser_and_start(clk, laser, trigger, reset_n, period,
LFSR);
input clk;
output reg laser;
output reg trigger;
output reg [10:0] period;
output reg [7:0] LFSR;
input reset_n;

reg [10:0] counter;

//parameter LFSR_Offset_Shift = 0;



wire feedback = LFSR[7];

always @(posedge trigger or negedge reset_n)
begin
      if(~reset_n)
            begin
            LFSR<=8'b00000011;
              period<=11'd1744;
            end
        else
              begin
                LFSR[0] <= feedback;
                LFSR[1] <= LFSR[0];
                LFSR[2] <= LFSR[1] ^ feedback;
                LFSR[3] <= LFSR[2] ^ feedback;
```

```
                LFSR[4] <= LFSR[3] ^ feedback;
                LFSR[5] <= LFSR[4];
                LFSR[6] <= LFSR[5];
                LFSR[7] <= LFSR[6];
                period<= LFSR + 11'd1744;
                end

  //keep constantly cycling through LFSR outputs. When the laser
fires, the LFSR period for the next pulse will be locked in.
  //shift_temp <= feedback[0] + feedback[1]*2 + feedback[2]*4 +
feedback[3]*8 + feedback[4]*16 + feedback[5]*32 + feedback[6]*64
+ feedback[7]*128;

  //center the shift, so that it can go -128 and +127.
  //shift_temp <= shift_temp - 128;


    //if(counter>=(11'd1999 + LFSR_Offset_Shift))
      //1counter<=0;
      //LFSR_Offset_Shift <= shift_temp;
      //else
      //counter<=counter+1'b1;

end

always @(posedge clk or negedge reset_n)
begin
    if(~reset_n)
        begin
        laser<=1'b1;
        end
      else
        begin
            if(counter<=12'd49)
            laser<=1'b0;
            else
            laser<=1'b1;
        end
end

always @(posedge clk or negedge reset_n)
begin
    if(~reset_n)
        begin
           trigger<=1'b0;
             counter<=0;
        end
      else
```

```
        begin
                if(counter>=period)
                     begin
                     trigger<=1'b1;
                     counter<=0;
                     end
                else
                     begin
                     trigger<=1'b0;
                     counter<=counter+1'b1;
                     end
        end
end

endmodule
```

PRBS Verilog Testbench code

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////
////////////////
// Company:
// Engineer:
//
// Create Date:   16:10:41 04/06/2018
// Design Name:   laser_and_start
// Module Name:
C:/Users/pchang0628/Documents/ise_project/laser_test/laser_teseb
ench.v
// Project Name:  laser_test
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module:
laser_and_start
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////
////////////////
```

```
module laser_tesebench;

	// Inputs
	reg clk;
	reg reset_n;

	// Outputs
	wire laser;
	wire trigger;
	wire [10:0] period;
	wire [7:0] LFSR;

	// Instantiate the Unit Under Test (UUT)
	laser_and_start uut (
		.clk(clk),
		.laser(laser),
		.trigger(trigger),
		.reset_n(reset_n),
		.period(period),
		.LFSR(LFSR)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		reset_n = 0;

		// Wait 2.5 ns for global reset to finish
		#2.5;
	 reset_n = 1;
		// Add stimulus here
		forever begin

		#2.5 clk=!clk;

		end

	end

endmodule
```