OPTICAL COMMUNICATIONS TESTBED FOR THE EXPLOITATION OF LUMINESCENCE EMISSIONS OF SOLAR CELLS FOR OPTICAL FREQUENCY IDENTIFICATION (OFID)

by

Samuel L. Denton

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Engineering Technology

West Lafayette, Indiana

May 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Walter D. Leon-Salas, Chair
School of Engineering Technology

Dr. Suranjan Panigrahi
School of Engineering Technology

Dr. Abdul Salam
Department of Computer and Information Technology

Approved by:
Dr. Duane D. Dunlap
Professor and Chair, School of Engineering Technology Graduate Education Committee

To my parents,

John and Nancy Denton,

To my sisters,

Maya and Grace,

And to all my friends,

Without your conversations and support, none of my success would be possible.

ACKNOWLEDGMENTS

The author would like to thank Prof. Leon-Salas and fellow lab members in the tinyLab for their guidance and support throughout this project. The author would also like to thank the National Science Foundation for its support through grant ECCS-1809637.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BER	bit error rate
COB	chip on board
EL	electro-luminescence
FPGA	field programmable gate array
GaAs	Gallium Arsenide
IoT	Internet of Things
IR	Infrared
LED	light emitting diode
LiFi	Light-Fidelity
OFID	optical frequency identification
OOF	on-off keying
OW	optical wireless
NFC	near field communication
PL	photo-luminescence
PLC	powerline communication
PLM	powerline modulation
RF	radiofrequency
VLC	visual light communication
WiFi	Wireless-Fidelity

GLOSSARY

Electro-luminescence – “The phenomenon in which a material emits light when in the presence of an electrical charge or field.” (Leon-salas & Fan, 2018)

Internet of Things – “The premise of Internet of Things is to have smart sensors collaborate directly without human involvement.” (Al-Fugaha et al., 2015)

Light-Fideltiy – “A bidirectional wireless communications technology covering the infrared and visible light spectrum for data communications.” (Haas, 2018)

Optical frequency identification – “A parallel to RFID, OFID uses solar cells to optically receive both power and information from an interrogating device.” (Leon-salas & Fan, 2018)

Photo-luminescence – “The phenomenon in which a material emits light when in the presence of light.” (Leon-salas & Fan, 2018)

Powerline communication – The transmission of data over powerlines

Powerline modulation – Modulation is the process of mixing an information signal with a carrier signal, in this instance the carrier is the powerline.

Visual light communication – Optical transmission of data taking place using the visible light spectrum.

Radiofrequency Identification – The use of radiofrequency waves to receive power and information from an interrogating device. (Chawla, Vipul and Ha, 2007)

ABSTRACT

Denton, Samuel M.S., Purdue University, May 2020. Optical Communications Testbed for the Exploitation of Luminescence Emissions of Solar Cells for Optical Frequency Identification (OFID). Major Professor: Walter D. Leon-Salas

The purpose of this thesis was to investigate the possibility of Optical Frequency Identification (OFID) technology being used as a communication pathway for devices in LiFi systems that serve to open alternative transmission paths for Internet-of-Things infrastructure. LiFi or light-fidelity, plays off the concept of wireless-fidelity, commonly known as WiFi, and follows the trend of moving to higher frequencies within the electromagnetic spectrum. LiFi lies within the visual light and infrared wavelength range, which can be referred to as the nanometer wave range. The developed optical communication testbed is a proof of concept showing that OFID technology, enabled by Gallium Arsenide solar cell emission, can communicate with Visual Light Communication (VLC) systems. The scope of the work entails the development of a testbed for a custom optical communications testbed for OFID linked to VLC communication by sending transmissions via powerline modulation. An optical receiver circuit was developed and tested, and integration and testing for powerline communication and LED luminaire were successful. Manchester encoded data was sent at 4800 bit rate optically from an infrared light source, received by the developed receivers and was decoded. Information was successfully transmitted over powerline from computer terminal to LED luminaire output at 2400, 3600, 4800, 7200, and 9600 bit rate. Integration of these communication links did not occur due to Purdue University closure of campus related activities from COVID-19.

CHAPTER 1. INTRODUCTION

The introduction and widespread adoption of Internet-of-Things (IoT) technology has created a vision of seamless interaction between multitudes of devices. To support this vision of connected vehicles, buildings, and other devices, many systems are being converted to enable IoT operations. To be an IoT device, an object must be connected with the internet (Al-Fuqaha et al., 2015). An IoT device consists of sensors collecting data and interacting with the environment or machine to machine over a network (Cisco Systems, 2016). The traditional transmission route for data in IoT schemes has taken place through wired or radio-based communications.

Pathways for radio transmission within the IoT environment include radio frequency identification (RFID), Wireless Fidelity (WiFi), Bluetooth, ZigBee, and Near Field Communication (NFC) (Al-Fuqaha et al., 2015). It is hard to narrow down the current number of IoT devices in use but McKinsey estimates 127 new devices connect to the internet each second (Patel et al, 2017). Cisco Systems has projected that 500 billion devices are expected to have IoT capability by 2030 (Cisco Systems, 2016). The dramatic increase in devices using these standards poses a problem for the unlicensed radio bands that are used. The size of the entire RF spectrum is only 0.3 THz (Haas, 2018). As unlicensed radio bands become more crowded, transmissions speeds will slow and interference will increase. The future bandwidth demand of wireless devices is projected to be 6 THz by 2040 (Haas, 2018).

An alternative transmission path has been in development based off the ability to transmit information through light, first called visible light communication (VLC) and then extended into LiFi. A play on the concept of WiFi, LiFi is a wireless communications technology covering the infrared (IR) and visible light spectrum for data communications. The size of the infrared and visible light spectrum is 780 THz (Haas, 2018). The data transmission rates depend on the modulation and lighting technology used. A segment within the optical communication ecosystem includes the idea of optical frequency identification (OFID). OFID is named as such due to the parallel ideas within RFID technology, that devices can receive both power and information from an interrogating device (Leon-salas & Fan, 2019). Within OFID, solar cells are employed for both conventional energy harvesting and the transmission and reception of information encoded optically (Leon-salas & Fan, 2019).

The testbed setup built for this thesis creates a platform that can be used as an optical communications testbed for OFID devices. The developed platform operates with high efficiency Gallium Arsenide (GaAs) solar cells of which the electroluminescence and photoluminescence properties can both be modulated. The modulated solar cell luminescence creates a communications signal used as an uplink within the testbed. In the system's downlink, the conventional LED light is used to modulate the electroluminescence of the OFID receiver. Communications from the luminaire are controlled by microcontroller and powerline modulation module linked to a computer creating a VLC system.

1.1 Statement of Problem

The topic of this thesis was the development of an optical communications test bed for OFID devices. The test bed is designed to operate with OFID technology conceptually described and developed by Prof. WD Leon-Salas and Xiaozhe Fan (Leon-salas & Fan, 2018, 2019). OFID technology is enabled using high efficiency solar cells which uses electro-luminescence and photo-luminescence from solar cells for wireless communication. The photo-luminescence property of the solar cell can be modulated to produce an output signal that can be used for communications purposes.

The purpose of this thesis was to investigate the possibility of OFID technology being used as a communication pathway for devices in LiFi systems that serve to open alternative transmission paths for IoT infrastructure. The developed optical communication testbed is a proof-of-concept showing that OFID technology can communicate with VLC systems. The scope of this work entails the development of a custom optical communications testbed.

1.2 Significance of the Problem

The number of deployed IoT devices is projected to be 500 billion by 2030 (Cisco Systems, 2016). A gap in necessary bandwidth for the operation of wireless devices is projected in the near future. With a crowded wireless spectrum increased interference and slower transmissions will occur. The RF spectrum has a size of 0.3 THz and the bandwidth demand of wireless devices is projected to be 6 THz in the next 20 years (Haas, 2018). The gap in available bandwidth can be supplemented with a new transmission path over IR and visible light. The available bandwidth over IR and visual light is projected to be 780 THz (Haas, 2018). OFID technology uses GaAs solar cells which can transmit data on the IR spectrum and receive information over the visible light spectrum. Establishing OFID devices as able to work within VLC systems provides a proof of concept for an application of OFID technology and aids in the development of systems and architecture used within the IR and visible light spectrum.

1.3 Scope of the Study

The scope of this study is the development and test of hardware and software related to an optical communications testbed for OFID technology. The testbed consists of a transmitter section to send encoded light communications from an LED luminaire to the OFID device and a receiver to collect the output signal from an OFID device to decode transmitted signals. LED luminaire communications are sent via powerline communication establishing the system to use VLC.

Outside of the scope of this study is the development of an OFID device to transmit data.

1.4 Purpose of the Study

The goals of this thesis are:

- Design of an optical communications testbed that transmits information able to be received by an OFID device and to receive information from an OFID device using powerline communication.
- Validation of the testbed regarding the transmission and reception of optical data. Validation consisting of both the build and test of the testbed.

The test of both the transmitter and receiver to establish that an uplink and downlink of the optical communication testbed work appropriately. First, each functional block of the design for the transmitter and receiver will be tested to determine that it functions as designed. Next the testbed will be constructed and each functional block will be tested to ensure functionality. Finally, whole system validation will take place by demonstrating the operation of the full testbed.

1.5 Assumptions, Limitations, & Delimitations

1.5.1 Assumptions

Assumptions for this thesis include:

- Optical communications taking place in the system will be line of sight.
- Due to the scarcity of GaAs solar cells, high-power IR LEDs will be used to simulate luminescence from GaAs solar cells.

1.5.2 Limitations

Limitations of this thesis include:

- A fully operational OFID device will not be developed. Luminescent emissions from GaAs solar cells will be simulated with LEDs.
- VLC transmission is enabled from the use of off the shelf LED, luminaires, LED drivers, and PLC module.
- OFID data will be encoded via Manchester coding

1.5.3 Delimitations

The delimitations of this thesis include:

- Communications distance is limited to up 4 meters within a closed room environment and the only light source being from the system.
- A low data bit rate will be used, not exceeding 20 kbps.
- Silicon solar cells will not be used in this thesis.

1.6 Summary

The rapid proliferation of IoT devices puts a strain on the finite amount of radiofrequency bandwidth that enables these devices. Alternative transmission paths on the electromagnetic spectrum make use of visible and infrared light. OFID transmission uses the photo-luminescence property of GaAs solar cell which transmit data using infrared light. The optical communication testbed developed for this thesis creates a system which receives OFID transmission and implements VLC transmission with a LED luminaire. The scope of this thesis is the development and test of hardware and software related to an optical communications testbed for OFID technology. The purpose of this study is to design an optical communications testbed that transmits information able to be received by an OFID device and to receive information from an OFID device using powerline communication. The purpose of the study is to also validate the design of the testbed regarding the transmission and reception of optical data and powerline communication links. Assumptions for this thesis are that communication are line of sight and that high efficiency GaAs solar cells will be used. Limitations are that an OFID device will not be developed in the study, VLC will occur using off the shelf components, and OFID data will use Manchester encoding. Delimitations for this thesis are communication distance will not exceed 4 meters, data bitrate will not exceed 20 kbps, and silicon solar cells will not be used.

CHAPTER 2. REVIEW OF LITERATURE

A review of literature was conducted to investigate the current state of visible light communications and optical frequency identification design and technology. To supplement this information background research on powerline communication, wireless communication, and optical communication was performed. This chapter serves to give a brief overview to the works and concepts relevant to the research question.

2.1 Powerline Communication

Wired communication is the act of transmitting data over a wire-based technology, by extension of this concept powerline communication (PLC) is the transmission of data over powerlines. There is a long history of utility companies using the power grid for control, maintenance, and charging for this commodity. Overtime the application of communication over power grid has become a realizable and efficient networking loop. Electrical distribution networks constitute a universal wiring system but at time of implementation were not designed for communication purposes. Electrical equipment causes erratic levels of impedance and attenuation from switching (Cypress Semiconductor, 2011). This and other time-variant interference places constraints on the transmission capability with restrictions on bandwidth, power limits, and high noise levels. Electrical powerlines are classified at three separate levels: high (>100kV), medium (1-100kV), and low(<1kV) (Cypress Semiconductor, 2011). A general trend is the lower the voltage level of the communication network the more hostile the transmission path.

Looking at PLC in terms of bands, two options are present: narrowband PLC and broadband PLC. Narrowband PLC utilizes lower frequencies and data rates from 3-500 kHz and up to hundreds of kbps respectively. Narrowband PLC is commonly applied for use in smart grids and other energy related applications. Broadband PLC utilizes higher frequencies, 1.8-250 MHz, and high data rates, up to hundreds of Mbps (Cypress Semiconductor, 2011). Broadband PLC applications have found widespread acceptance for internet distribution and other home networking solutions due to the high data rates and ease of implementation.

2.2 Wireless Communication

Wireless communication is the act of transmitting data with technology that does not utilize an electrical conductor. The most commonly recognized and widely used instances of wireless communication uses the radio frequency section of the electromagnetic spectrum. Less common is optical communications which creates wireless communication networks on the infrared, visual, and ultraviolet light section of the electromagnetic spectrum.

2.2.1 Radio Frequency Communication

As noted above, radio frequency (RF) refers to a section of the electromagnetic spectrum. Radio waves are electromagnetic waves ranging from 3 kHz to 300 GHz (Mouser Electronics, 2019). RF is a naturally occurring phenomena whose causes include solar flares, lighting, and stars aging. Humankind uses technology to artificially create radio waves to oscillate at desired frequencies for communications. RF signals are described by either frequency or wavelength. The relationship between frequency and wavelength is inverse, shown Equation 1, in relationship to the speed of light, *C*. Further in **Table 1** the frequency bands and their wavelengths are shown.

$$C = f \times \lambda$$

Equation 1. Speed of light equation

Table 1. Frequency Band Designations (National Instruments, 2019)

Frequency	Wavelength	Band	Description
30-300 Hz	$10^4 – 10^3$ km	ELF	Extremely low frequency
300-3000 Hz	$10^3 – 10^2$ km	VF	Voice frequency
3-30 kHz	100 – 10 km	VLF	Very low frequency
30-300 kHz	10 – 1 km	LF	Low frequency
0.3-3 MHz	1 – 0.1 km	MF	Medium frequency
3-30 MHz	100 – 10 m	HF	High frequency
30-300 MHz	10 – 1m	VHF	Very high frequency
300-3000 MHz	100 – 10 cm	UHF	Ultra-high frequency
3-30 GHz	10 – 1 cm	SHF	Superhigh frequency
30-300 GHz	10 – 1 mm	EHF	Extremely high frequency

Examples of RF communication in action include television and radio broadcasts, radar systems, mobile communications, and remote controlled vehicles (Mouser Electronics, 2019).

Due to a fixed amount of radiofrequency spectrum, usage is highly regulated and segments of frequency are allocated for licensed and unlicensed use, **Table 2**, shows this distribution.

Table 2. Table of allocated frequencies and designated use (National Instruments, 2019)

Frequencies in kHz	Allocated purposes
490 – 510	Distress (telegraph)
510 – 535	Government
535 – 1605	AM radio
1605 – 1750	Land/mobile public safety
1800 - 2000	Amateur radio
Frequencies in MHz	Allocated purposes
26.96 – 27.23, 462.525 – 467.475	Citizen band radios
30.56 – 32, 33 – 34, 35 – 38, 39 – 40, 40.02 – 40.98, 75.2 – 76, 150.05 – 156.2475, 157.1875 – 161.575, 162.0125 – 173.4, 220 – 222, 421 – 430, 451 – 454, 456 – 459, 460 – 512, 746 – 824, 851 – 869, 896 – 901, 935 – 940	Private mobile radio (taxis, trucks, buses, railroads)
74.8 – 75.2, 108 – 137, 328.6 – 335.4, 960 – 1215, 1427 – 1525, 220 – 2290, 2310 – 2320, 2345 - 2390	Aviation (communication and radar)
162.0125 – 173.2	Vehicle recovery (LoJack)
50 – 54, 144 – 148, 216 – 220, 222 – 225, 420 – 450, 902 – 928, 1240 – 1300, 2300 – 2305, 2390 – 2450	Amateur radio
72 – 73, 75.2 – 76, 218 – 219	Radio control (personal)
54 – 72, 76 – 88, 174 – 216, 470 - 608	Television broadcasting VHF and UHF
88 – 99, 100 – 108	FM radio broadcasting
824 – 849	Cellular telephones
1850 – 1990	Personal communications
1910 – 1930, 2390 – 2400	Personal comm. (unlicensed)
1215 – 1240, 1350 – 1400, 1559 – 1610	Global Positioning Systems (GPS)
Frequencies in GHz	Allocated Purposes
0.216 – 0.220, 0.235 – 0.267, 0.4061 – 0.45, 0.902 – 0.928, 0.960 – 1.215, 1.215 – 2.229, 2.320 – 2.345, 2.360 – 2.390, 27 – 3.1, 3.1 – 3.7, 5.0 – 5.47, 5.6 – 5.925, 8.5 – 10, 10.0 – 10.45, 10.5 – 10.55 13.25 – 13.75, 14 – 14.2, 15.4 – 16.6, 17.2 – 17.7, 24.05 – 24.45, 33.4 – 36, 45 – 46.9, 59 – 64, 66 – 71, 76 – 77, 92 – 100	Radar, all types
2.390 – 2.400	LANs (unlicensed)
2.40 – 2.4835	Microwave ovens
45.5 – 46.9, 76 – 77, 95 – 100, 134 – 142	Vehicle, anti-collision, navigation
10.5 – 10.55, 24.05 – 24.25	Police speed radar
0.902 – 0.928, 24 – 2.5, 5.85 – 5.925	Radio frequency identification

Table 2 (cont). Table of allocated frequencies and designated use (National Instruments, 2019)

3.7 – 4.2, 11.7 – 12.2, 14.2 – 14.5, 17.7 – 18.8, 27.5 – 29.1, 29.25 – 30, 40.5 – 41.5, 49.2 – 50.2	Geostationary satellites with fixed earth receivers

In the introduction of the thesis, two common applications of RF signal, WiFi and RFID are referenced for sake of comparison and will be further described here. WiFi, or Wireless Fidelity, is a wireless network architecture based on the serial standards first laid out by IEEE 802.11 (Kaushik, 2012). The frequency bands designated for WiFi are 2.4 GHz and 5 GHz. With the original standard for WiFi the given data rate was 1 Mbps, now more current iterations of the standard have a data rate going up to 54 Mbps (Kaushik, 2012). The operating range of WiFi varies with the hardware implementation, commonly the range is 50 feet for indoor operation and 1500 feet for outdoor operation. WiFi is a local area networking (LAN) technology designed to connect multiple devices and be used inside buildings or other small implementation areas (Kaushik, 2012).

RFID systems consist of two subsystems: a reader, or interrogator, and a tag, or transponder (Chawla, Vipul and Ha, 2007). Communication between the reader and tag occurs through a technique called load modulation where variations in current flowing through the tag are detected by the reader. (Chawla, Vipul and Ha, 2007) There are three types of RFID systems: passive, semi-passive and active. Passive RFID tags receive power from the reader and are read through inductive coupling. Semi-passive RFID tags communicate by inductive coupling like passive RFID but a battery is included in the device allowing for sensors, time tracking, and other features. Active RFIDs use battery power to send energy into the reader instead of powering and then reflecting back energy like the passive RFID. Figure 3 shows the frequencies used for RFID transmission. Applications for RFID technology include keycards, toll collection, and product tracking.

Both WiFi and RFID face issues inherent to the nature and widespread success of RF communication. With the large-scale implementation of these RF products and only finite bands of a unlicensed bandwidth available for use, congestion will become an issue for these technologies causing slower transmission speed and interference (Leon-salas & Fan, 2019).

2.2.2 Optical Wireless Communication

Optical wireless (OW) communications modulate the intensity of light with the light being the carrier signal, link elements shown **Figure 2**. Optical signal is converted to electrical signal by photodiodes or conversely electrical signal into optical signal by LEDs. Many conventional OW communication systems are implemented to work within the near infrared range close to 850 nm and 1550 nm due to existing lighting sources (Langer & Grubor, 2007). Data rates for interior communication linkages range up to 100 Mbps while exterior communication linkages range in the Gbps (Langer & Grubor, 2007). OW transmission has high free-space loss due to unguided propagation, meaning radiating power is lost or not captured by the receiver (Langer & Grubor, 2007). Multipath propagation causes inter-symbol-interference and becomes a key factor with speeds above 10 Mbps (Langer & Grubor, 2007). The primary source of noise in OW systems is ambient light creating shot noise. With artificial light sources, harmonics up to 1 MHz can be introduced into the signal (Langer & Grubor, 2007). Both optical filtering and electrical filtering can be used to mitigate noise in an optical communications system.

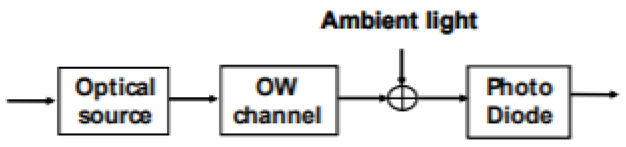


Figure 2. Elements in an OW communications link (Langer & Grubor, 2007)

A more recent form of wireless optical communication is Visible Light Communication (VLC) using LEDs producing wavelengths from 380-700 nm and have a bandwidth around 20 MHz (Langer & Grubor, 2007). In VLC scenarios, PLC can be used to feed signal into conventional LED lighting infrastructure leading to transmissions as shown in **Figure 3** (Langer & Grubor, 2007). VLC communication uses a photodiode as a receiver using the concept of direct detection, where light is used as point-to-point connection similar to a wire (Haas, Yin, Wang, & Chen, 2016). VLC faces many similar noises sources as detailed above.

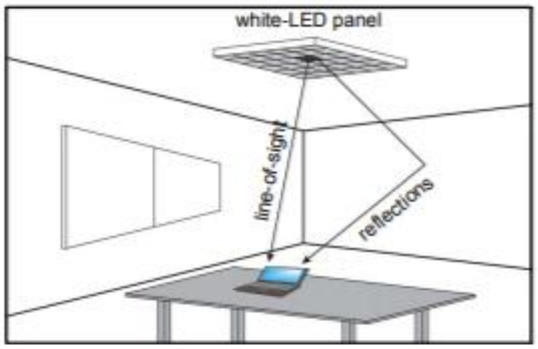


Figure 3. Visible Light Communication Scenario (Langer & Grubor, 2007)

Light-Fidelity (LiFi), an extension of VLC technology and concepts, uses the infrared and visible light spectrum to produce high speed, secure, and bi-directional network communications (Haas, 2018). Within LiFi, speeds of over 3 Gbps have been demonstrate using a single LED (Haas et al., 2016). VLC and LiFi have aspects in common such as use of intensity modulation, LED transmitters, and photodiode receivers. The difference between VLC and LiFi is the technique of communication, VLC was developed to be point to point while LiFi is being developed to be multipoint (Haas et al., 2016). Multipoint communication enables full user mobility within a cell. Modulation techniques used within LiFi are similar to those used within RF communication with necessary modifications added. A simple modulation scheme for LiFi is on-off keying (OOK) providing a balanced tradeoff between performance and implementation complexity (Haas et al., 2016).

Wireless optical communication is a complementary technology to conventional radio based wireless communications. Wireless optical communication has distinct characteristics from radio including: large unregulated available bandwidth, no electromagnetic interference with existing systems, and signal confinement within a room (Langer & Grubor, 2007). This creates applications where optical-based communications can be the desired choice over radio-based communication. One such application is in electromagnetically sensitive environments such as hospitals or airplanes due to the minimal electromagnetic interference. Another application due to the signal confinement would be for military use, or financial communications as information transfer would not leave the room.

2.3 Optical Frequency Identification

Optical Frequency Identification is an analogous technology to RFID following the idea that tags can receive information and power from a reader. The difference being the transmission path for the power and information in the two technologies, RFID from RF transmission and OFID from light sources. Separate from RFID which can powered passively at a close range or actively from a battery, OFID tags use high efficiency GaAs solar cells as transceivers and can be powered from ambient light. Also corresponding to RFID, OFID can have passive and active tags. Passive OFID uses ambient light as the power source and the reader receives luminescent emissions of the solar cell. Active OFID uses light from the reader to power the device and the reader receives the luminescent emissions of the solar cell.

GaAs solar cells can be used to transmit information by modulating the photo-luminescence and electro-luminescence emissions of the solar cell (Leon-salas & Fan, 2019). Photo-luminescence can be modulated by switching the solar cell between open circuit and short circuit, while electro-luminescence can be modulated by applying time varying voltage to the solar cell (Leon-salas & Fan, 2019)(Leon-salas & Fan, 2018). Solar cells have previously been implemented in schemes to receive optically encoded information, but GaAs solar cells also possess the ability to transmit information due to strong luminescence emission around 870 nm in the near infrared region.

2.4 Manchester Encoding

Manchester encoding is a data modulation technique used for binary data transfer based on analog, RF, optical, high-speed digital, or long-distance-digital signals (Keim, 2016). Manchester code is a classical code where ‘0’ is encoded as ‘01’ and ‘1’ is encoded as ‘10’ (Cailean, Cagneau, Chassagne, Dimian, & Popa, 2014). This establishes the central idea of Manchester encoding, data can be represented by voltage transitions instead of voltage levels (Keim, 2016).

While digital communication is largely advantageous for communication over analog signal, issues occur related to synchronization and the need for DC coupling. Advantages of Manchester code are DC balance, easy clock and data recovery, and decent bit error rate (BER) performance (Cailean et al., 2014). Manchester encoding solves these issues by doing two things.

First, the signal never remains at logic high or logic low for an extended period of time eliminating the coupling issue. Second the data signal is converted into a data plus synchronization signal (Keim, 2016). Data plus synchronization is useful because in many situations complexity and inefficiency is increased if two transmitters and receivers are needed to make a complex data link (Keim, 2016). In Manchester encoding data transitions line up with the clock edge. A main disadvantage of Manchester encoding is that twice as much bandwidth is used to transmit the data because a Manchester code introduces a transition in the middle of each bit to encode one logic state. If signal frequency is a constrained factor in a design, this creates a maximum data rate (Keim, 2016). Manchester encoding is used for optical data in this instance to remove error by AC coupling and to limit data synchronization related issues.

CHAPTER 3. RESEARCH METHODOLOGY

This chapter details the design of the optical communications testbed for OFID including the hardware and software descriptions for the receiver, transmitter, and powerline communication. The operation and physical set up of the platform is also described.

3.1 Optical Communication Testbed Functional Components

The testbed developed as a part of this thesis creates a VLC environment so that the communication capabilities of OFID technology may be explored. The transmitter section uses a microcontroller, powerline modulation board, LED driver, and chip on board (COB) LED luminaire. The receiver circuit uses a photodiode, a transimpedance amplifier (TIA), several bandpass gain stages, low-pass filter, comparator, and field programmable gate array (FPGA). The network created from the transmitter and receiver, powerline communications modules, and a PC allow for a transmission to be sent in the testbed and then received at the PC or to be sent from the PC and be received by a device below the testbed. A model of the testbed system is shown in **Figure 4**. A block diagram of the unit panel with receiver, transmitter, PLC module, and microcontroller with necessary peripherals is shown **Figure 5**.

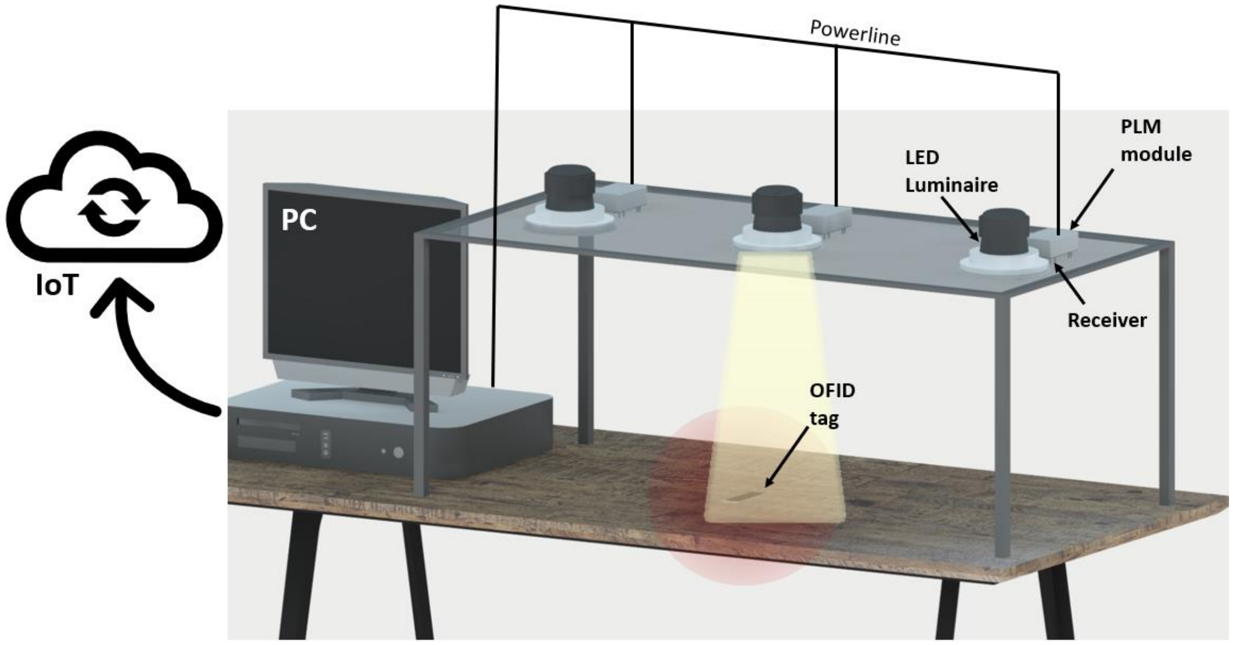


Figure 4. Model of optical communication testbed

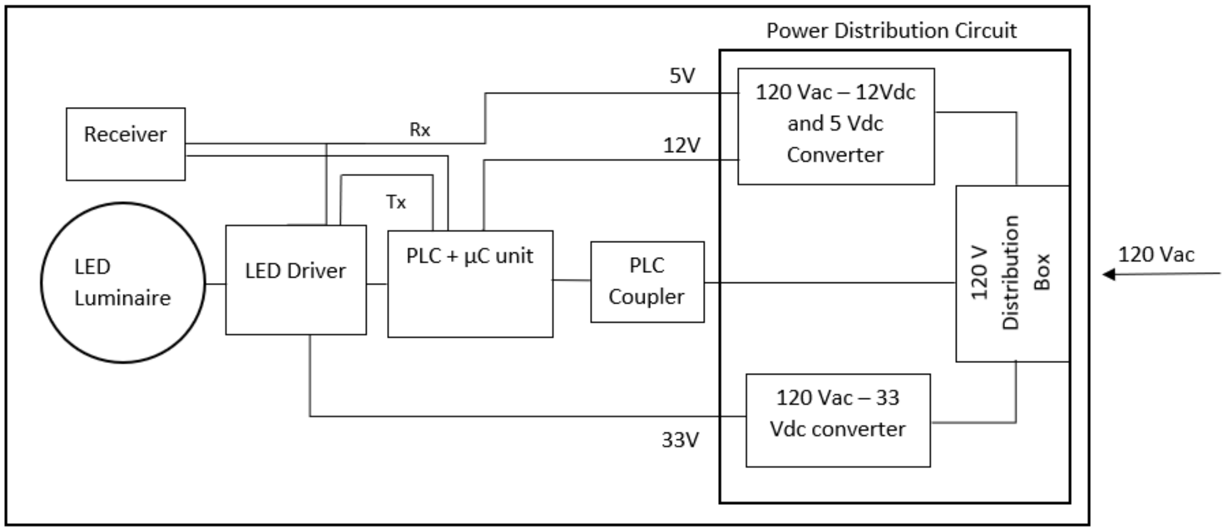


Figure 5. Optical communication panel layout

3.1.1 Receiver

The receiver for this testbed is designed to receive low power optical signals from OFID technology. Broken into functional blocks the receiver consists of a photodiode, a transimpedance amplifier, four bandpass filter gain stages, an eighth order lowpass filter, a comparator, and an FPGA. **Figure 6** shows the optical receiver circuit broken into functional blocks. In the implementation of the receiver, the number of gain stages are configurable. This arrangement enables experimentation with different gain and bandwidth settings.

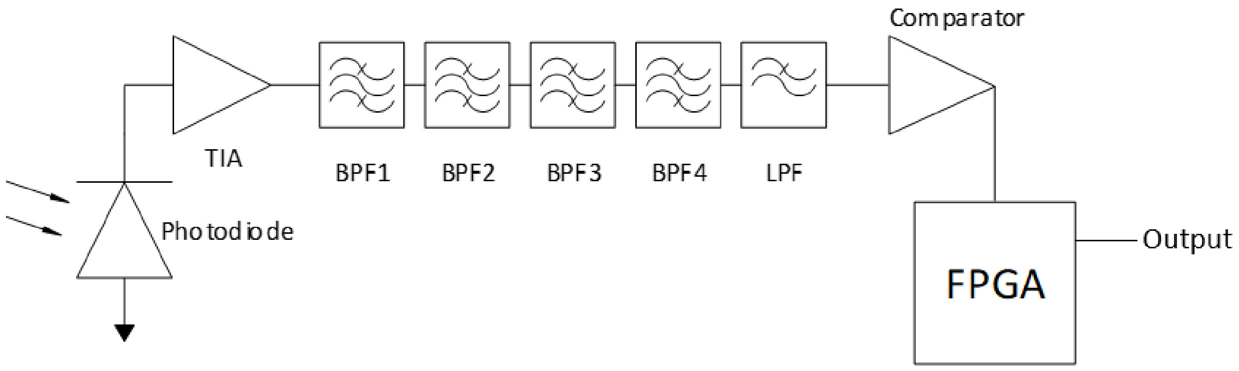


Figure 6. Optical receiver functional block diagram

3.1.1.1. Hardware Description

Photodiode and TIA

The first functional block of the system is the photodiode and transimpedance amplifier. A photodiode is used as a transducer to convert light to current. The effect of light on a photodiode is that an electron-hole pairs are generated when in reverse bias allowing current to flow (Sinclair, 2001). In this incidence the light hitting the photodiode is ambient light and the infrared light emitted from the solar cell.

The photodiode chosen for this block was the Vishay Semiconductor BPW34 for its high speed and radiant sensitivity. BPW34 responds to near infrared and visual light from wavelengths 430-1100 nm. With the signal as a current measurement, it needs to be converted to a voltage for further signal conditioning and processing. The conversion from current to voltage uses a transimpedance amplifier. The circuit to implement this is shown below in **Figure 7**. The current from the photodiode is driven over a feedback resistor creating a voltage output. Reverse photo-current on the BPW34 datasheet is listed for a typical value of 50 μA. The signal applied to the photodiode will cause a change to the current level. These levels of current are very small, so a high resistance needs to be used to create a reasonable level of voltage output. A resistor value was chosen to be 1MΩ so that voltage range of interest will take place on the millivolt level. If this is found to be experimentally untrue the resistor size can be adjusted appropriately. Another recommendation for transimpedance amplifiers is to include a balance capacitor in parallel with the feedback resistor, not shown in **Figure 7**. The balance capacitor provides stabilization to the TIA output. At this point in time, it is believed to be unnecessary for this project’s implementation of the circuit. Out of caution an empty footprint has been left for this in the PCB layout.

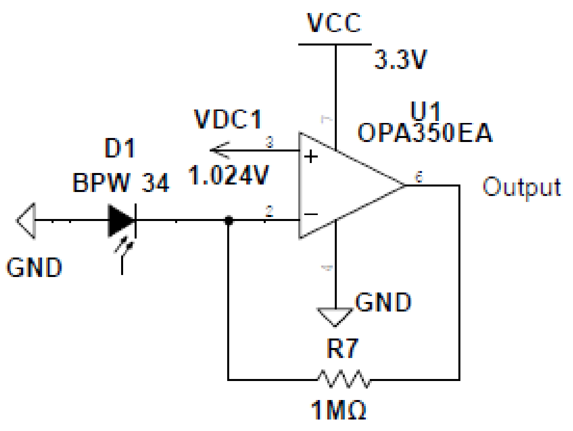


Figure 7. Photodiode and transimpedance circuit

Band Pass Filter Gain Stages

A filter is used to remove unwanted noise from a signal. Gain is used to increase the level of a signal. Gain is applied in filters to increase the level of the desired signal while reducing or removing the unwanted signal components. In this implementation, a bandpass filter is used to remove DC signal and associated noise at low frequencies and to provide the signal immunity from higher frequencies. The corner frequency for the high pass filter is placed close to 100 Hz to remove the DC signal and associated noise and allowing higher frequency signals to pass. Using commonly available resistor and capacitor values a filter with a corner frequency at 106 Hz was designed for implementation, calculations shown **Equation 8**. Similarly, the corner frequency for the lowpass filter was designed with to standard component values be close to 10kHz. The calculated lowpass corner frequency is calculated to be 12.9 kHz shown **Equation 9**. The gain relationship in the passband is determined by the relationship between the resistors in the low and high pass filters. Looking at **Figure 11,** it is seen that an inverting amplifier configuration is used. The calculated gain is shown in **Equation 10**.

$$f_{hp} = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 1.5\ \text{k}\Omega \times 100\ \text{nF}} = 106.1\ \text{Hz}$$

Equation 8. High pass filter design calculation

$$f_{lp} = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 56\ \text{k}\Omega \times 2.2\ \text{nF}} = 12.2\ \ \text{kHz}$$

Equation 9. Low pass filter design calculation

$$G = \frac{R_2}{R_1} = \frac{56\ \text{k}\Omega}{1.5\ \text{k}\Omega} = 37$$

Equation 10. Passband inverting gain calculation

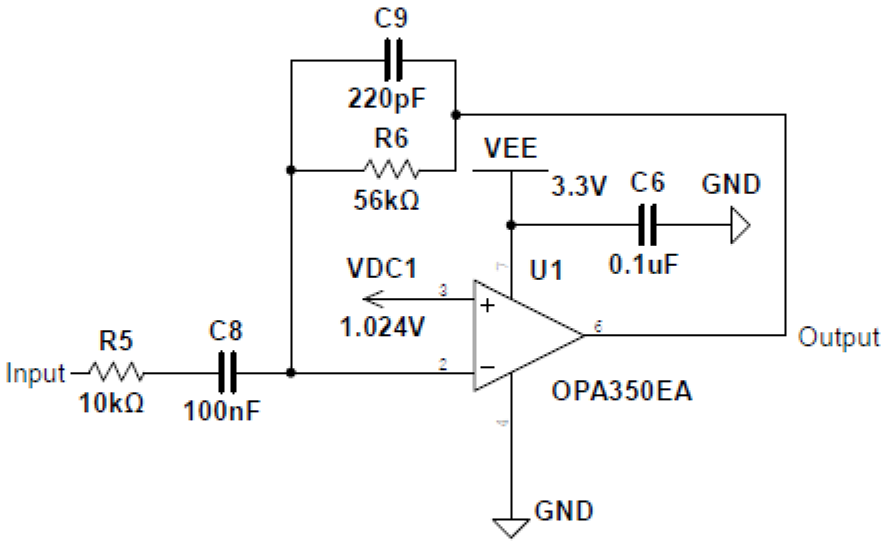


Figure 11. Bandpass filter gain circuit

Low-pass filter

In the actual implementation of bandpass filters, the actual roll off rate of the upper corner frequency is not as ideal as designed, to remedy this issue a MAX291 eighth-order lowpass filter was added to provide steeper attenuation to the high frequencies. The MAX291 IC uses a clock, LTC6992, to select the corner frequency of the lowpass filter. Clock speed is determined by DIVCODE programming or the resistor network used with V+, DIV, and GND, shown in **Figure 12**. The datasheet of the LTC6992 contains a table to show resistor networks for desired frequencies. For this implementation, a frequency of 1 MHz was chosen and resistors must be an open and short. The potentiometer attached to DIVMOD controls the duty cycle of the modulation. The circuit for the MAX291 implementation is shown, **Figure 12**.

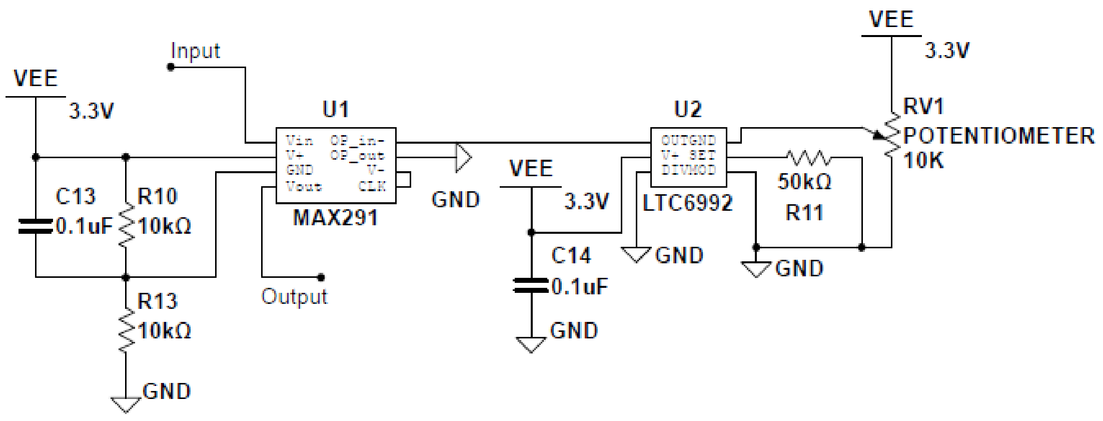


Figure 12. Eighth order lowpass filter circuit

Comparator

The next block in the receiver is the comparator. This section converts the analog signal into a digital signal. The comparator chosen to be used in this section is the LT1716 for its low power consumption and successful use in other lab projects. The threshold level of the comparator can be set by the potentiometer or by a data slicer depending on the jumper postion. The circuit design is shown **Figure 13**.

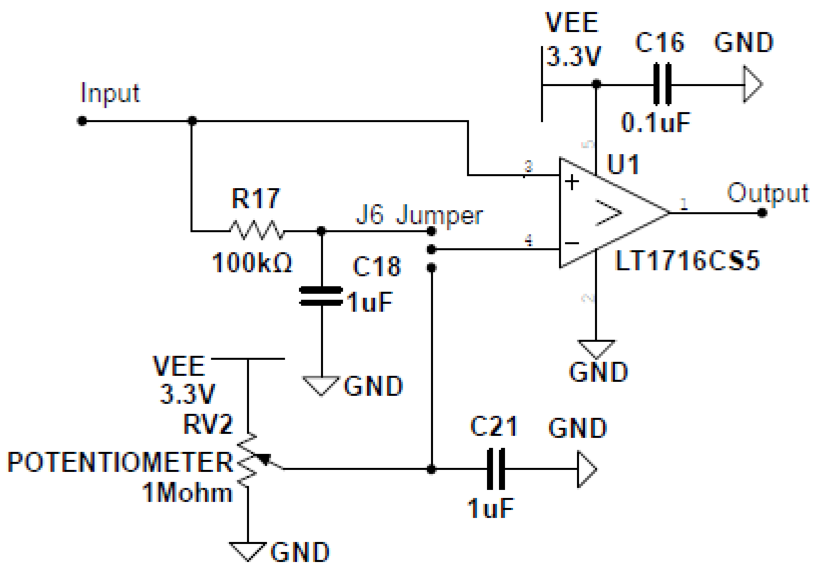


Figure 13. Comparator Circuit

FPGA

An FPGA and peripherals were added to the board for serial communication and Manchester encoding of the optical signal. The FPGA chosen was the ICE5LP4K-SG48ITR, for its low power operation and good feedback from successful implementation by other members of the lab. The ICE5LP4K-SG48ITR also feature 4 DSP blocks which provides flexibility in the design to add digital filtering as well as encoding and decoding. At current state the logic for the FPGA implementation has not been designed. The functional block diagram and board layout are shown, **Figure 15** and **Figure 19**.

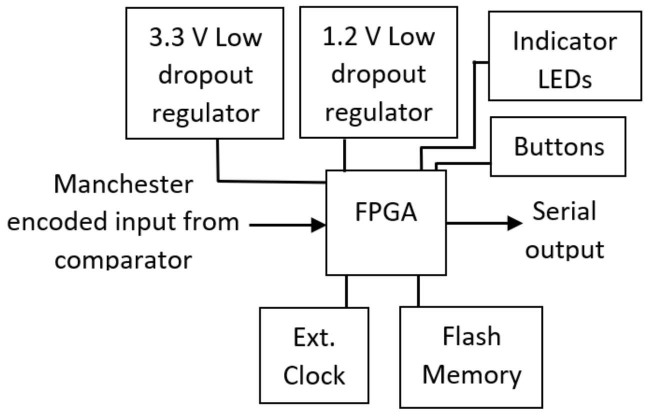

3.3 V Low dropout regulator
1.2 V Low dropout regulator
Indicator LEDs
Buttons
Manchester encoded input from comparator
FPGA
Serial output
Ext. Clock
Flash Memory

Figure 14. FPGA block diagram

3.1.2 Transmitter

The transmitter for this project utilizes transmit data from the powerline communication module and microcontroller to control the pulse modulation input for an LED driver which in turn drives the LED luminaire. The transmission signal then is emitted from the luminaire as an optical transmission. The LED luminaire chosen for this solution is a Bridgelux Vero 13, BXRC-40E2000-D-73, chip on board (COB) LED in a housing used to convert traditional incandescent downlight into an LED downlight, housing shown in **Figure 15**. The housing from a Juno Lighting downlight conversion unit was used, the model number for this is Juno 4RLDG2-927-9-WWH This model of the Vero 13 has a nominal correlated color temperature (CCT) of 4000 K, color rendering index (CRI) of 90, pulsed drive current of 500 mA, forward voltage of 31.8, and wattage of 15.9 W. CCT is a way to describe the color spectrum output of a light by correlating the output color spectrum to that of a black body emitter at the designated temperature, in this case 4000 K. The color spectrum for the Vero series of light is shown in **Figure 16**. The 4000 K option was chosen due to the highest amount of spectral power distribution across the widest band of wavelengths. The CRI is a measurement index the accuracy of the color output of the light to render color for human eyes, commonly for interior lighting 85 to 90 is considered good quality. There is no apparent link between CRI and solar cell performance, but this a specification which matters in commercial lighting where VLC applications may be used. The forward voltage is the amount of voltage that the COB LED will need to transmit light. From the forward voltage the voltage drive levels of the LED driver are determined. When the forward voltage is reached, the saturation current for the LED is reached, for pulsed applications this is listed as 500 mA.

Figure 15. LED downlight conversion unit (Acuity Brands, 2019)

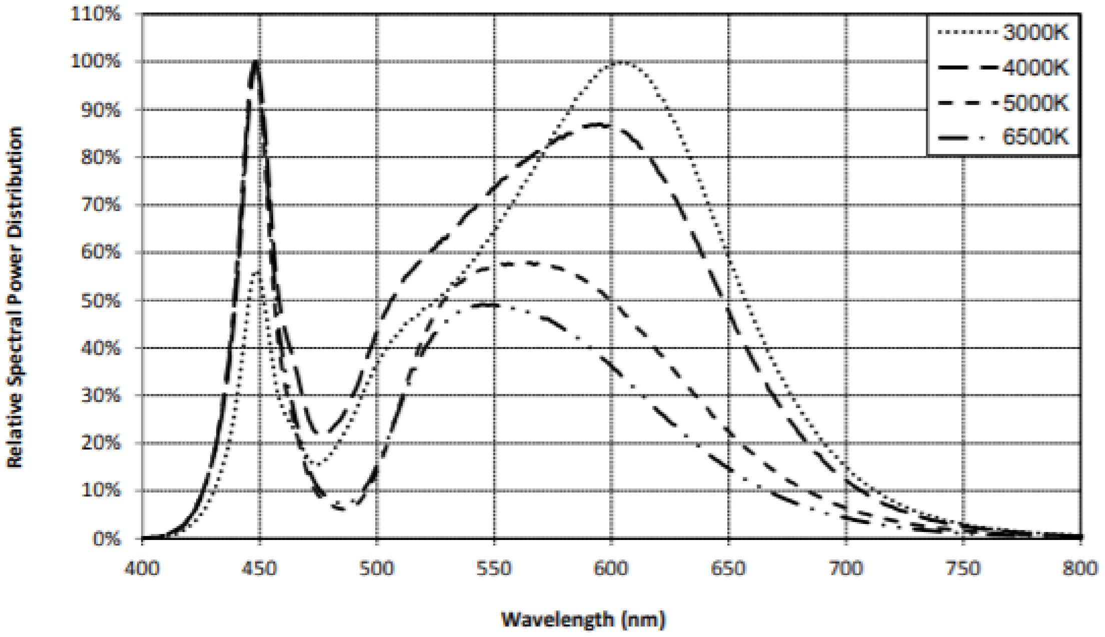


Figure 16. Spectral power distribution of Vero13 series LED modules (Bridgelux, 2018)

The LED driver chosen for this application is a LM3409HV PFET controller. This device was chosen because the evaluation board works with the required voltage level and allows for external pulse width modulation (PWM) dimming. External PWM produces a square wave pulsed light output, which is necessary for sending the desired transmission.

After experimentation, it was found that using external PWM dimming caused light noise instability, or flickering, when data was sent. In this instance there are two possible causes of perceptible flicker. Human eyes can perceive changes in light levels when the light flickers at a speed less than 100-120 Hz. The sent data is variable so it can change in length or value and transmission speed in depending on the test case. This results in a case where either a test message or a test bit can cause a flicker at less than 120 Hz and be visible to the human eye due to the frequency of light modulation. Since frequency of light modulation is constrained by other factors in the system, an option to resolve the issue is controlling the amplitude of the signal modulation to reduce visible flicker. In the original design a serial signal from the microcontroller (0-3.3V) is directly connected to the led driver.

A level-shifter circuit was designed and connected in series in between the microcontroller and the led driver to create a variable and user-controlled amplitude. **Figure 17** shows the circuit implemented for the level shifter. Two TL081 op amps were used. The first op amp was implemented as a buffer in case of any unknown loading effects that could occur between the microcontroller and the shifting stage. The second op amp stage uses a potentiometer as an adder on the circuit and the possibility to add gain remains from R2 and R4. Initially the resistor configuration is set to unity gain and can be modified if determined necessary. The potentiometer was set to provide 1.30 V as the voltage offset. This resulted in no visible flicker from the light output. The voltage can be set higher, but the lowest possible value was selected to maximize the amplitude of the transmitted signal. Due to the complexity of human vision, outside factors affect the sensitivity and perception of the change of flux from noise. The voltage offset was set qualitatively, and the voltage level may need to be changed due to the perception of the user

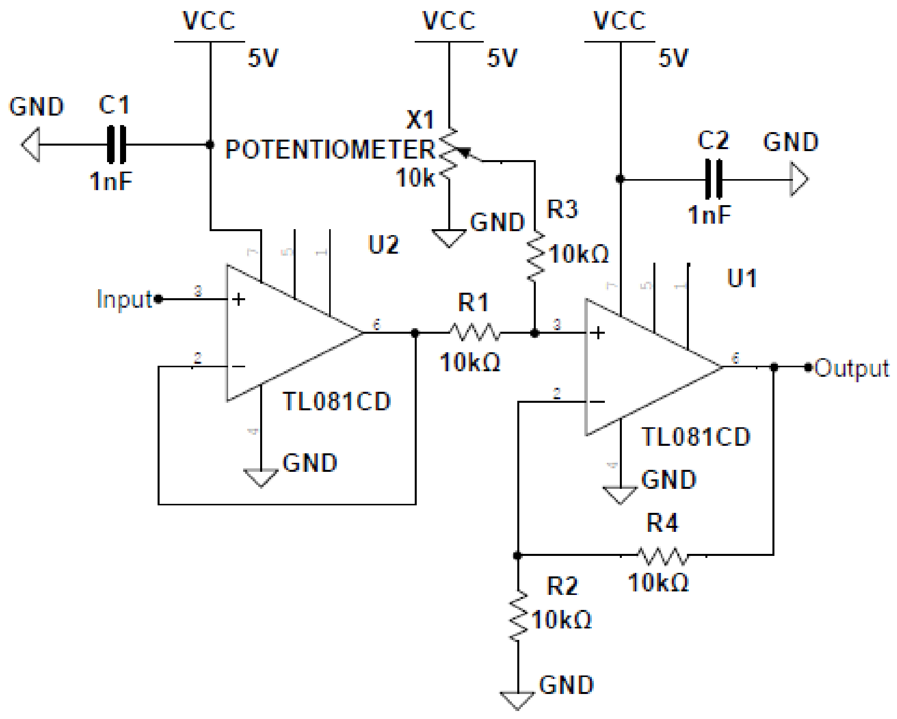


Figure 17. Level-shifter circuit

3.1.3 Powerline Communication Network

The powerline communication network for this system was built using evaluation modules from STMicroelectronics (STM). STM developed a ST7580 powerline communication expansion board for use with the STM32 Nucleo microcontroller products. The ST7580 is a chip developed to be used for powerline networking system on chip. The ST7580 expansion paired with AC coupling modules allow for communication over AC powerlines, expansion board is shown **Figure 18** and AC coupler **Figure 19**. The ST7580 modem allows for communication up to 28.8 kbps (STMicroelectronics, 2017b). The AC coupler is designed by the manufacturer to be in accordance with relevant band regulations for Cenelec B to D.

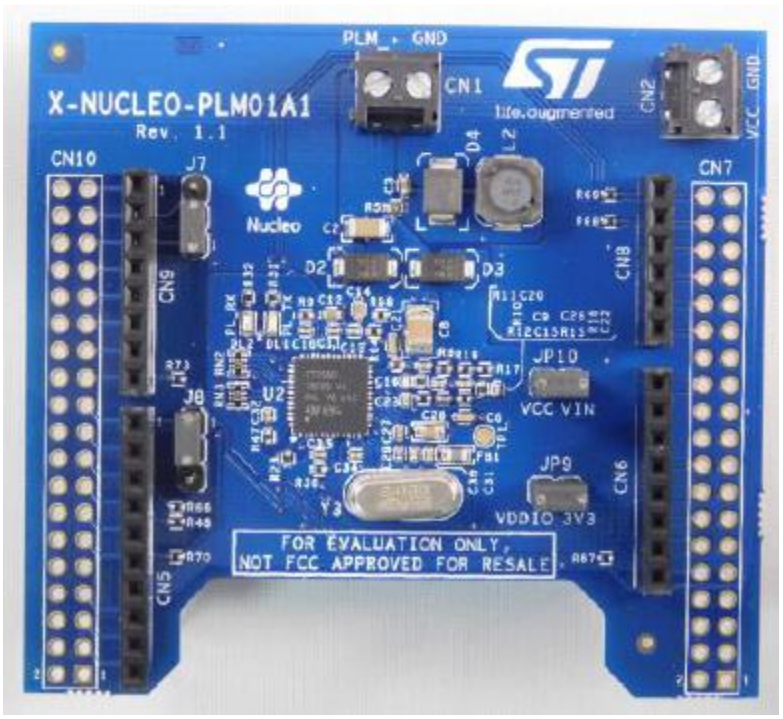


Figure 18. ST7580 powerline communication expansion board (STMicroelectronics, 2017b)

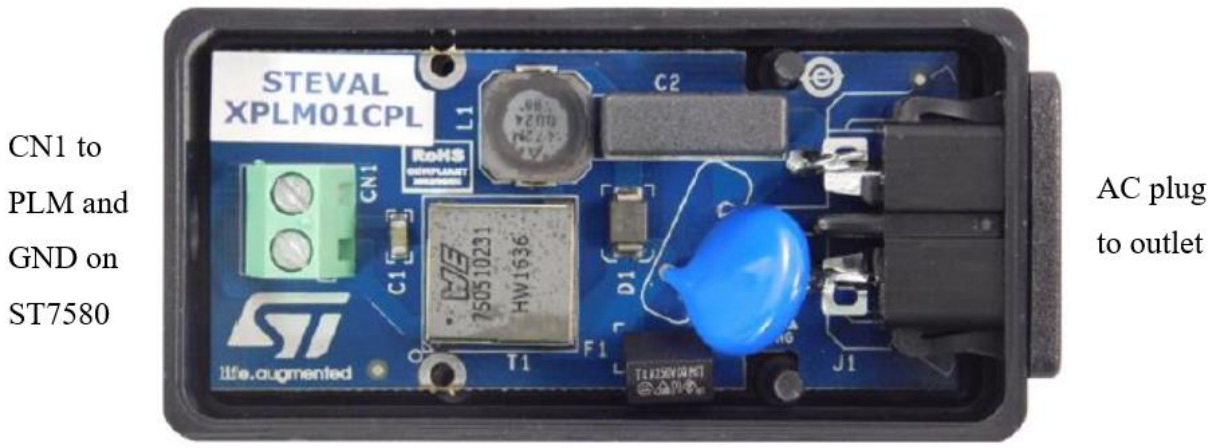


Figure 19. STM AC coupler (STMicroelectronics, 2017b)

A library and sample program for the expansion board is provided in Appendix C. The sample program has been successfully implemented with two modules. The master and slave interaction in the sample program is shown **Figure 20**. The operation of the sample program provides an output as shown in **Figure 21** when monitored on a serial terminal. To send information within the testbed the sample program was modified transmit data over the payload independently and further to transmit user input data.one module is used to drive the PWM input for the LED driver. Final setup for the optical communication testbed needs to have the PLC modules setup to perform as serial relays. This allows for a serial port monitor on a PC to send a signal, be transmitted by PLC to the panel in the testbed, received by the next module, and sent as a PWM signal to the driver.

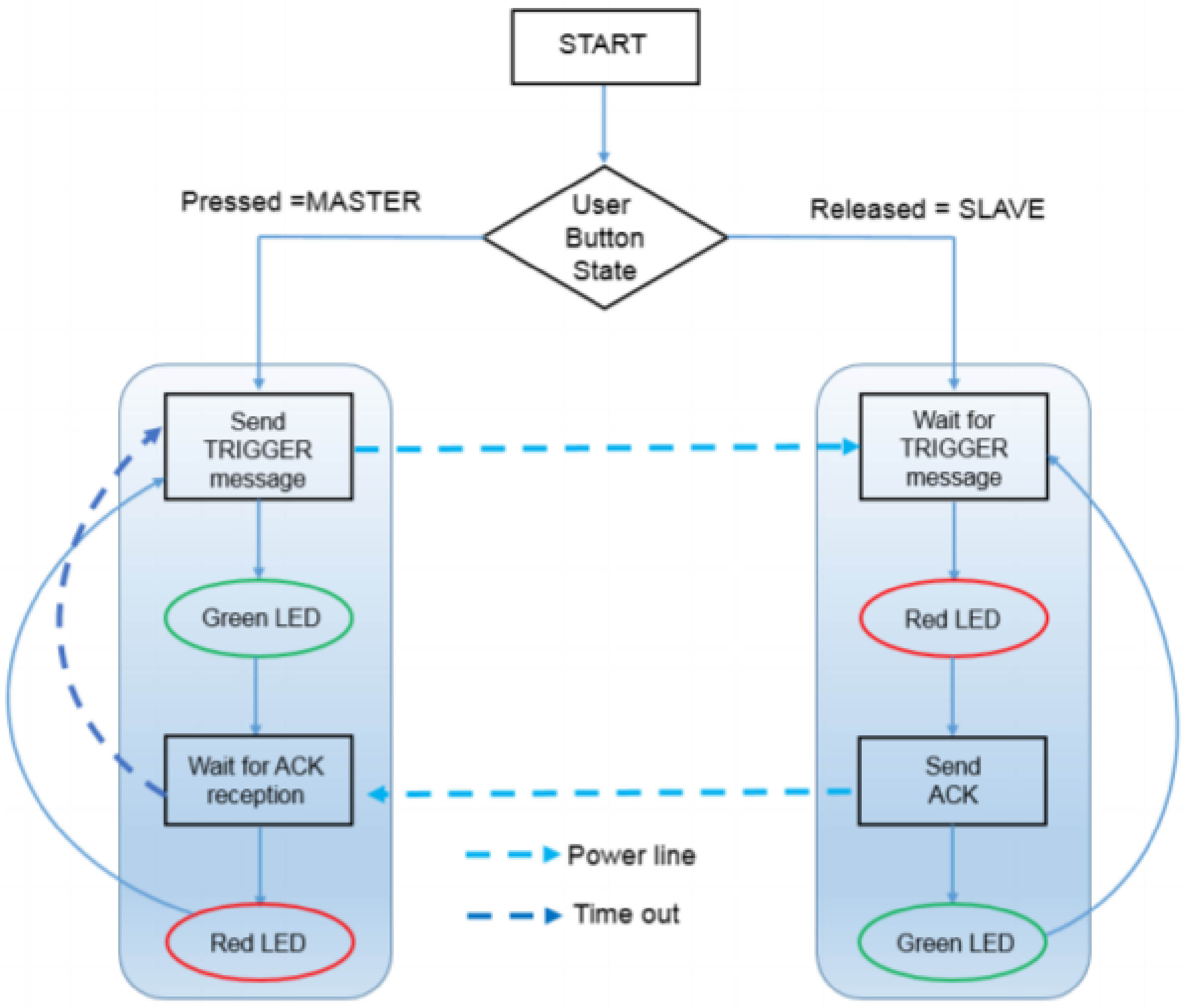
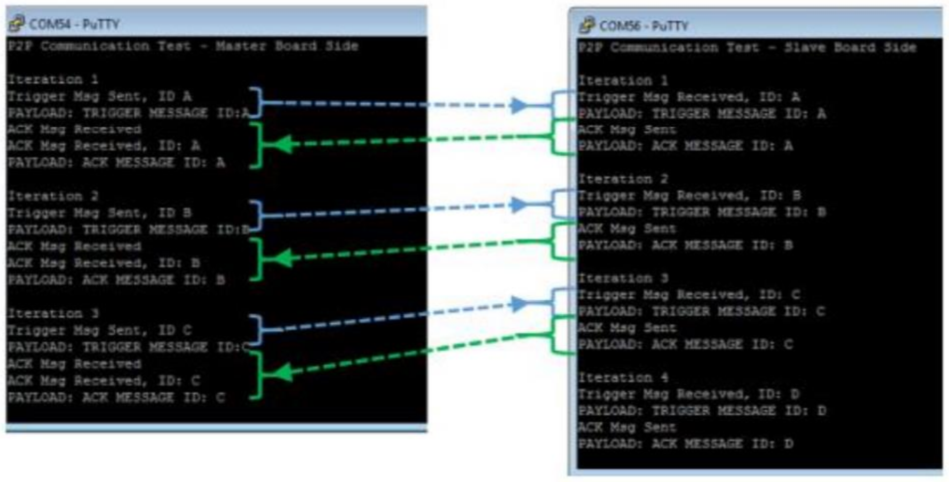


Figure 20. STM32 PLM Sample program flow (STMicroelectronics, 2017a)



```
P2P Communication Test - Master Board Side

Iteration 1
Trigger Msg Sent, ID A
PAYLOAD: TRIGGER MESSAGE ID:A
ACK Msg Received
ACK Msg Received, ID: A
PAYLOAD: ACK MESSAGE ID: A

Iteration 2
Trigger Msg Sent, ID B
PAYLOAD: TRIGGER MESSAGE ID:B
ACK Msg Received
ACK Msg Received, ID: B
PAYLOAD: ACK MESSAGE ID: B

Iteration 3
Trigger Msg Sent, ID C
PAYLOAD: TRIGGER MESSAGE ID:C
ACK Msg Received
ACK Msg Received, ID: C
PAYLOAD: ACK MESSAGE ID: C
```

```
P2P Communication Test - Slave Board Side

Iteration 1
Trigger Msg Received, ID: A
PAYLOAD: TRIGGER MESSAGE ID: A
ACK Msg Sent
PAYLOAD: ACK MESSAGE ID: A

Iteration 2
Trigger Msg Received, ID: B
PAYLOAD: TRIGGER MESSAGE ID: B
ACK Msg Sent
PAYLOAD: ACK MESSAGE ID: B

Iteration 3
Trigger Msg Received, ID: C
PAYLOAD: TRIGGER MESSAGE ID: C
ACK Msg Sent
PAYLOAD: ACK MESSAGE ID: C

Iteration 4
Trigger Msg Received, ID: D
PAYLOAD: TRIGGER MESSAGE ID: D
ACK Msg Sent
PAYLOAD: ACK MESSAGE ID: D
```

Figure 21. STM32 PLM program debug output (STMicroelectronics, 2017a)

3.1.4 Power Distribution Board

To supply power to the system and to safely connect to 120 Vac powerlines, a power distribution circuit was built using AC-DC convertor modules. The power requirements are 5 Vdc, 12 Vdc, 32 Vdc, and 120 Vac. To meet the requirements two modules were used, the Delta AA60S3600A to provide 36 Vdc and the Delta AA04D0512A for 5 and 12 Vdc. Fusing was implemented to protect the system. The functional block layout is shown **Figure 22**.

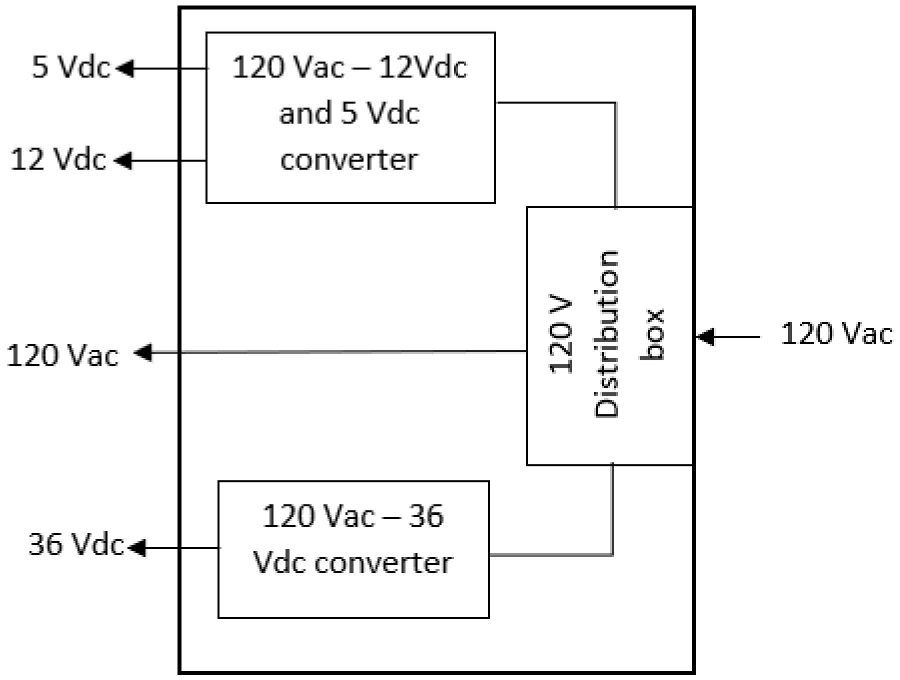

5 Vdc
12 Vdc
120 Vac – 12Vdc and 5 Vdc converter
120 Vac
120 V Distribution box
120 Vac
36 Vdc
120 Vac – 36 Vdc converter

Figure 22. Power distribution board

3.2 Summary of Operation

The receiver converts an optical signal to electrical signal. The electrical signal is then conditioned and converted to a digital signal to send to the PLC module. From the PLC module the received signal is relayed to the PLC module at the computer. A serial monitor at the computer then presents the transmission to the user. From the serial monitor, signal can be sent to the PLC module over powerline. The transmission is sent to the LED driver, which then sends a PWM signal to the luminaire to be receiver by a device underneath the testbed.

CHAPTER 4. RESULTS

This chapter details the test plan and test results of the optical communication testbed for OFID. The sections of the optical communication testbed tested are the optical receiver circuit, powerline communication, power distribution circuit, Manchester communication test, and serial communication test.

4.1 Test Plan and Results

4.1.1 Optical Receiver Circuit

To test the optical receiver circuit the functionality of each block within the circuit was tested, **Figure 23**. For the transimpedance amplifier, bandpass filters, and lowpass filter frequency response was measured. To measure the frequency response, a Keysight33210 A function/arbitrary waveform generator and Siglent SDS 1104X-E oscilloscope were used. The test process is to output a signal from the function generator using logarithmic steps across the frequency spectrum, at each step a csv file from the oscilloscope was saved, and after the data for all steps was collected a transfer function script was used in MATLAB to determine the frequency response, transfer function script located in **Appendix B**. The comparator was evaluated by verifying the functionality with a sinewave and voltage reference placed on the inputs of the circuit.

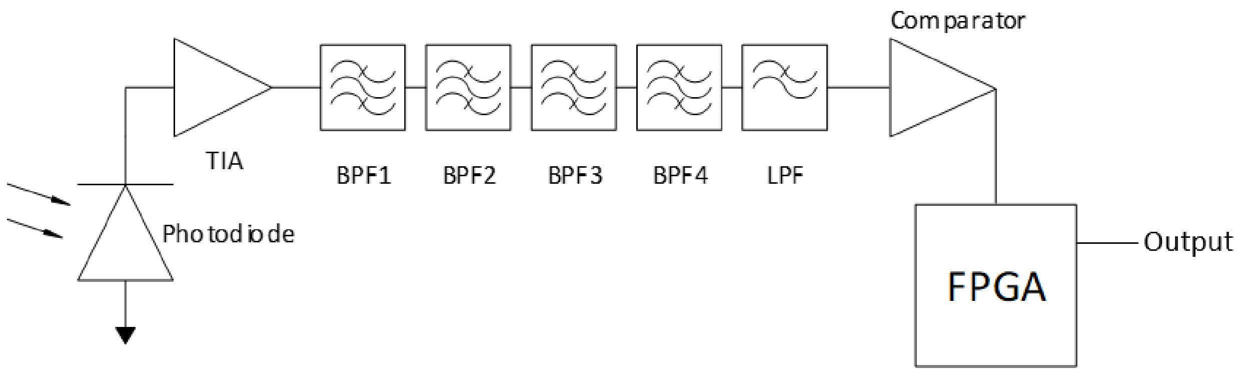


Figure 23. Optical receiver functional block diagram

Photodiode and Transimpedance Amplifier

The frequency response of the photodiode and transimpedance amplifier was tested. **Figure 24** shows the functional blocks of the test setup used. To perform the test a light source was created to emulate the output of the OFID tag. The light source was created using a function generator to set the desired frequency, an ALD115pal MOSFET to drive the LED, and SFH 4232A infrared LED to emulate the solar cell output, the circuit diagram of the IR light source is shown **Figure 25** and the test setup is shown **Figure 26**. The frequency response of the photodiode and transimpedance amplifier was calculated using MATLAB, tf_estimation.m script shown in **Appendix B**, and the resulting frequency response is shown **Figure 27.**

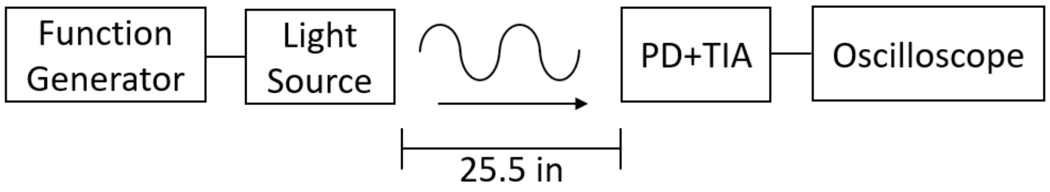


Figure 24. Photodiode and transimpedance amplifier frequency response functional test setup

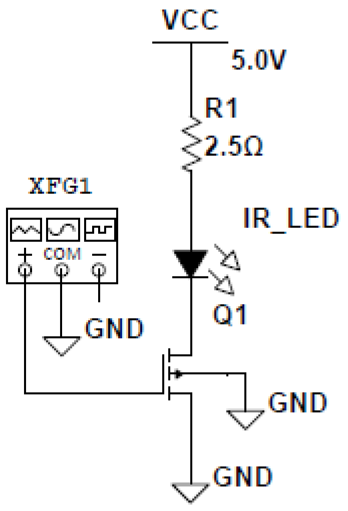


Figure 25. IR light source circuit

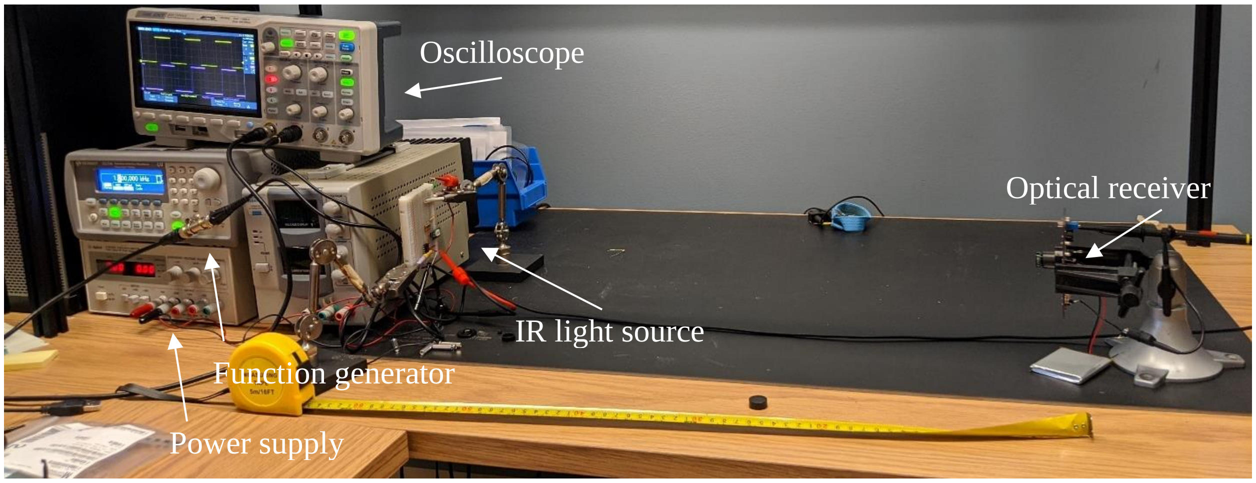


Figure 26. Photodiode and transimpedance amplifier test setup

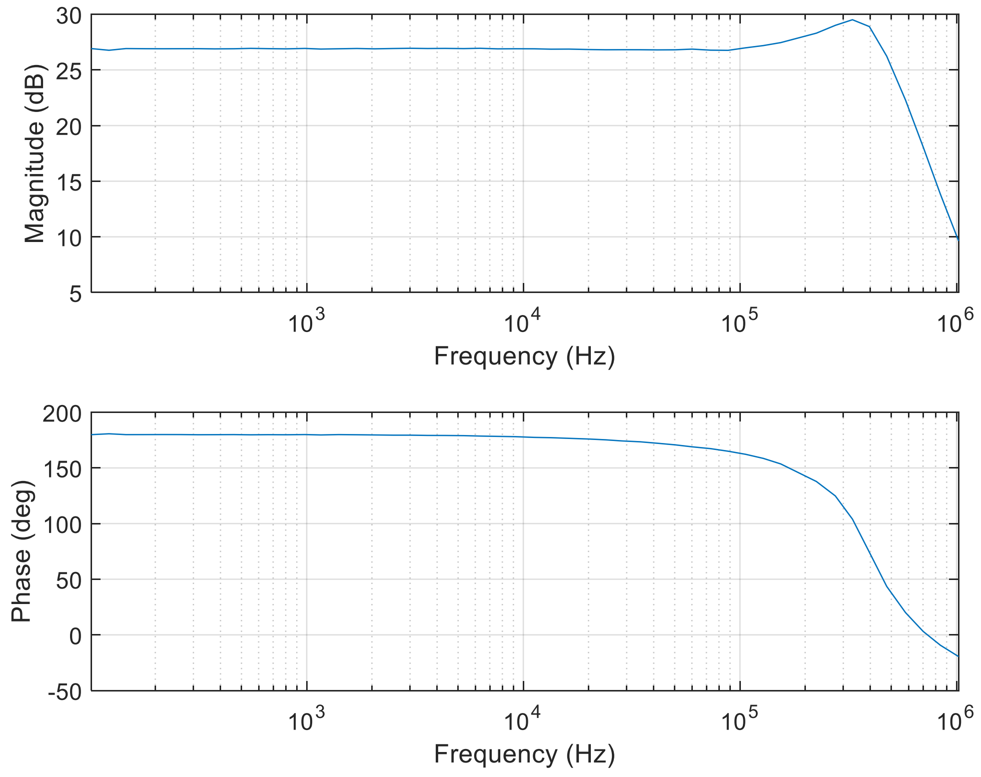


Figure 27. Frequency response of photodiode and transimpedance amplifier

Bandpass Filter

The frequency response of bandpass filters 1, 2, 3, and 4 were measured. **Figure 28** shows the functional blocks of the test setup used. The frequency response of the bandpass filters was calculated using MATLAB, tf_estimation.m script shown in **Appendix B**. The calculated frequency response for bandpass filter 2 is shown **Figure 29**. The results for the filters 1, 3, and 4 are in **Appendix B**.

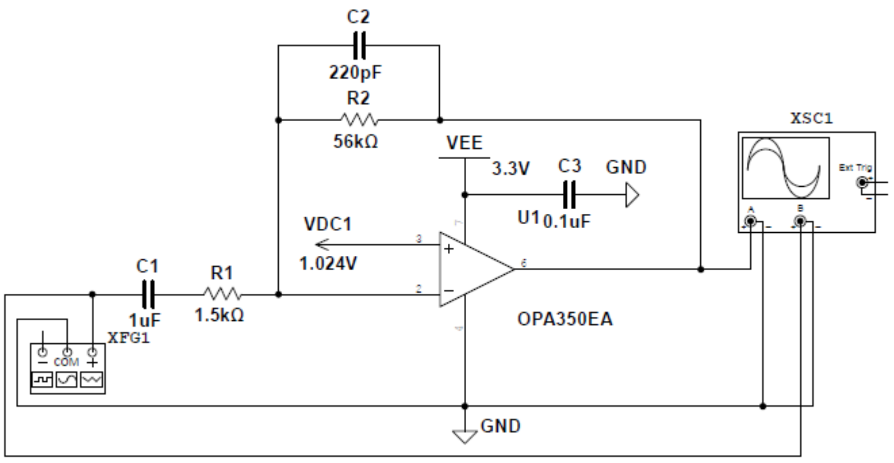


Figure 28. Bandpass filter frequency response test setup

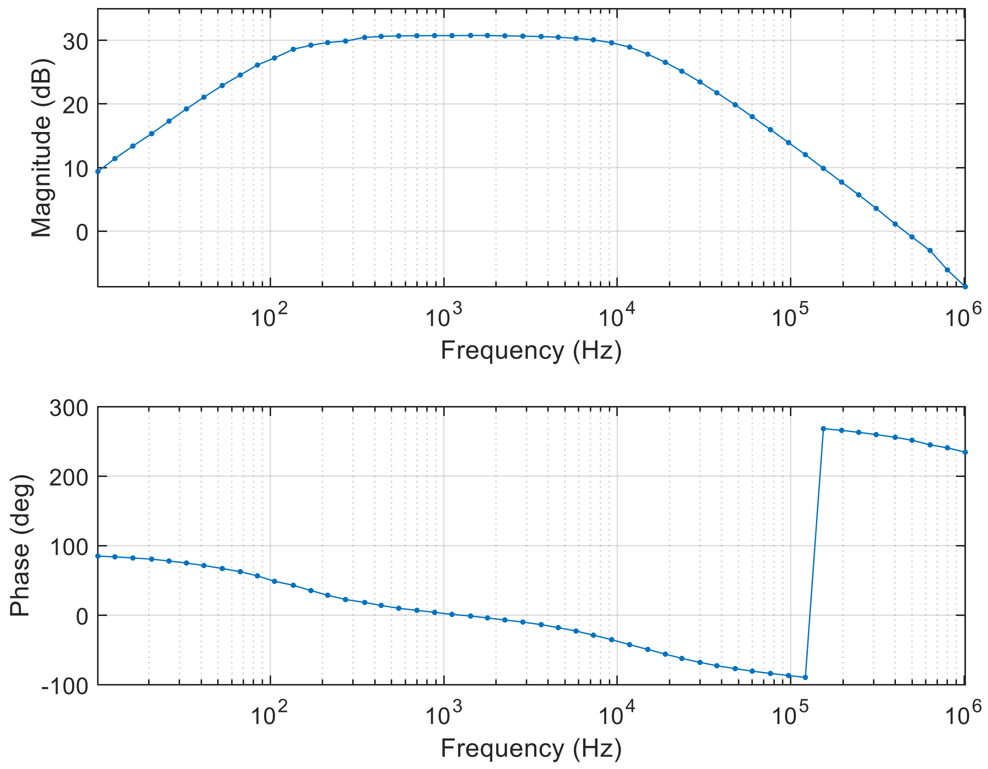


Figure 29. Bandpass filter 2 frequency response

Eighth Order Lowpass Filter

The frequency response of the eighth order lowpass filter was tested. **Figure 30** shows the functional blocks of the test setup used. The frequency response of the lowpass filter was calculated using MATLAB, tf_estimation.m script shown in **Appendix B**. The calculated frequency response for lowpass filter is shown **Figure 31**.

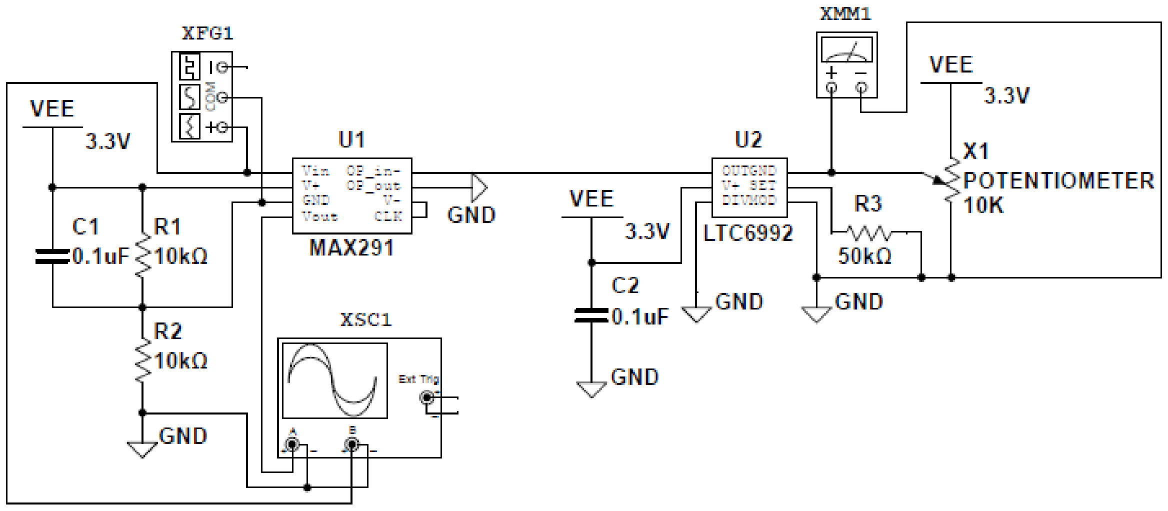


Figure 30. Lowpass filter test setup

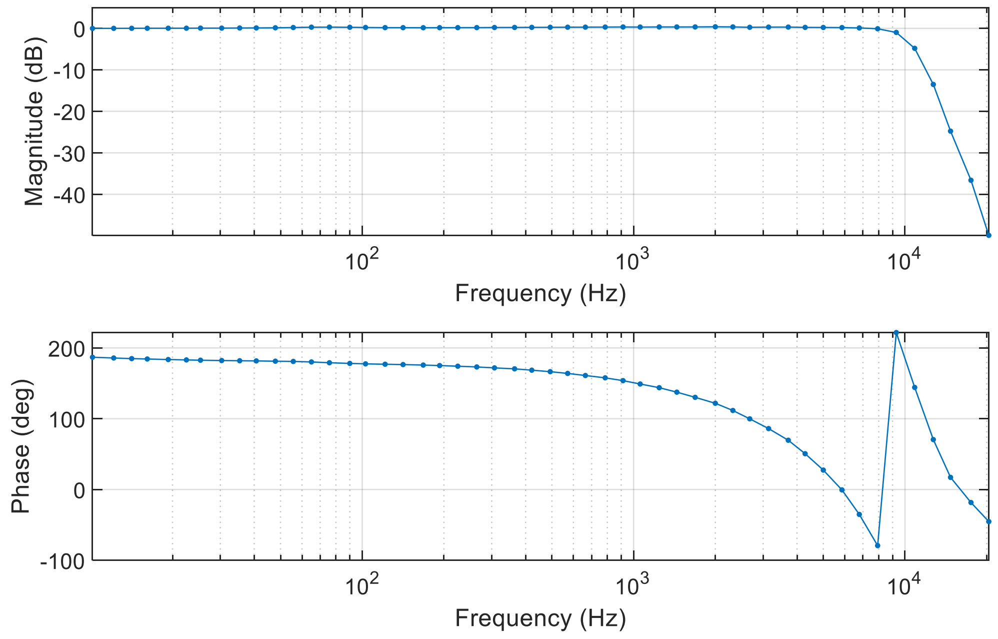


Figure 31. Lowpass filter frequency response

Comparator

The operation of the comparator was tested using Keysight33210A function generator, Siglent SDS 1104X-E oscilloscope, and Fluke 107 multimeter. The test setup for the comparator is shown **Figure 32.** The potentiometer on the input to the comparator was adjusted to various voltages between 0 – 3.3 V and the output of the circuit was observed. **Figure 33** shows the operation of the comparator with channel 1 being an input sinewave, channel 3 is the voltage from the potentiometer, and channel 2 is the output of the comparator.

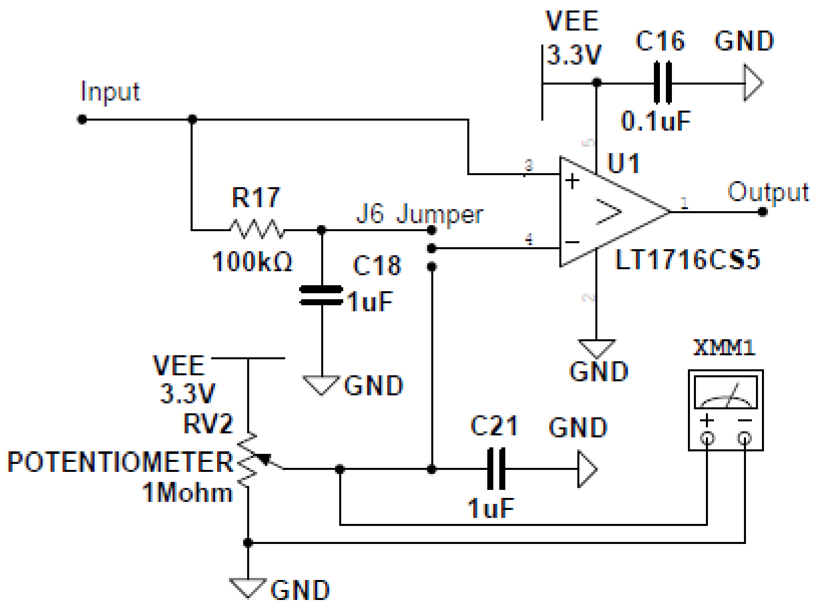


Figure 32. Comparator test setup

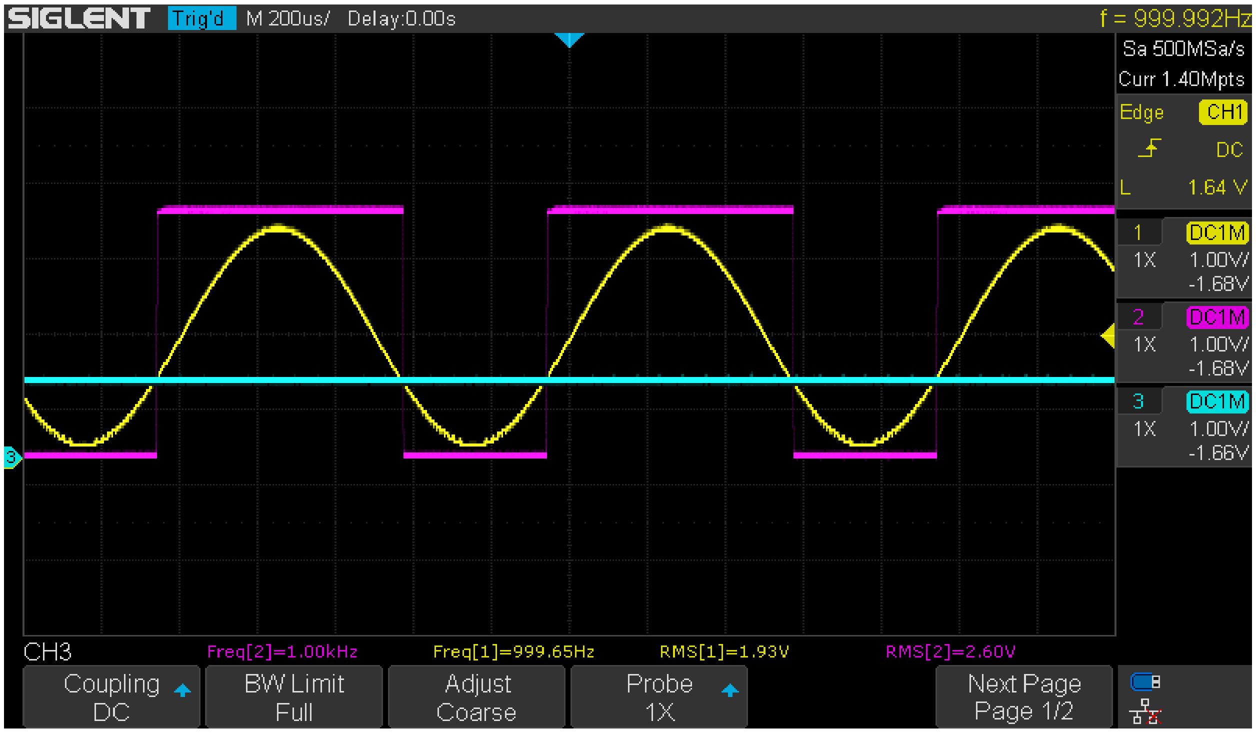


Figure 33. Demonstration of comparator circuit

4.1.2 System Integration Noise Investigation

After testing each block of the optical receiver circuit, the blocks were connected to test the functionality, **Figure 34** shows the block diagram of the test set up and **Figure 35** shows the test setup. The system noise and troubleshooting are detailed in **Appendix B**. Noise was found from several sources within the circuit. The lowpass filter and clock were removed as the trace for the clock passed too close to several components and infected the circuit with clock noise. A source of noise was also determined to be ambient light from the lab. **Figure 36** shows the frequency spectrum before the comparator in the optical receiver circuit when no ambient light or signal is being received by the optical receiver. **Figure 37** shows utilizes the same configuration and setup as the system in **Figure 36**, after noise issues have resolved. **Figure 38** shows a 1kHz signal sent from the IR light source with the LED transmitter being used for ambient light. In **Figures 36**, **37**, and **38**, the oscilloscope was used to measure traces from the light source (trace 1, yellow), the comparator output (trace 2, purple), and the output and frequency spectrum of bandpass filter 3 (trace 4, green and FFT, white).

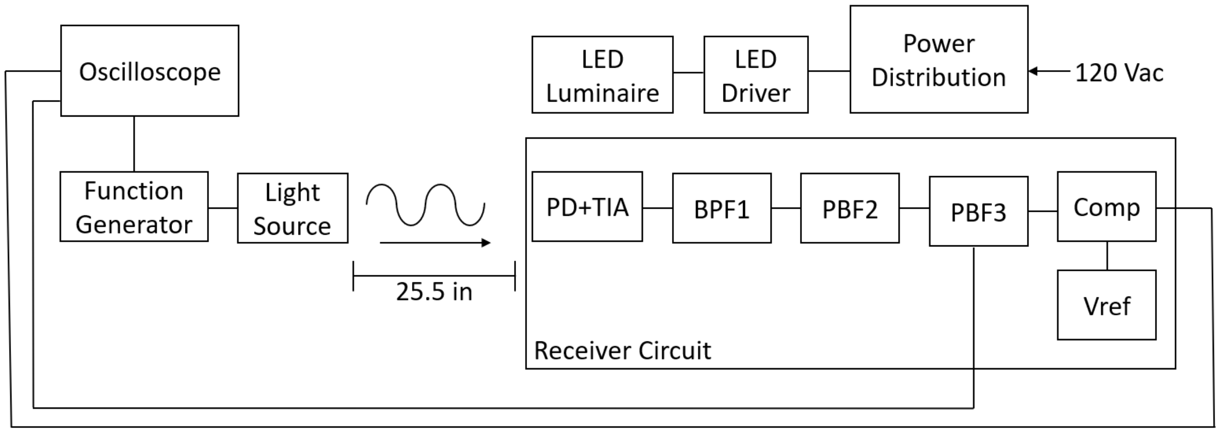


Figure 34. Functional block diagram of noise test setup

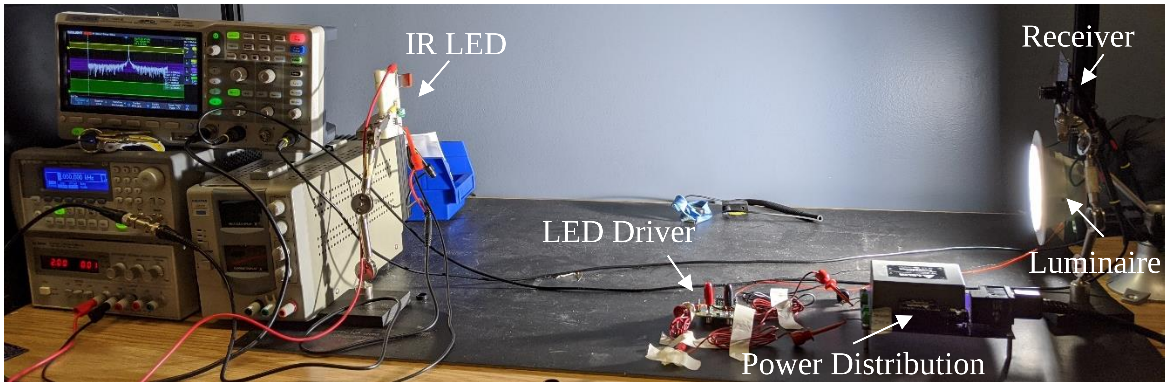


Figure 35. Setup for system noise test

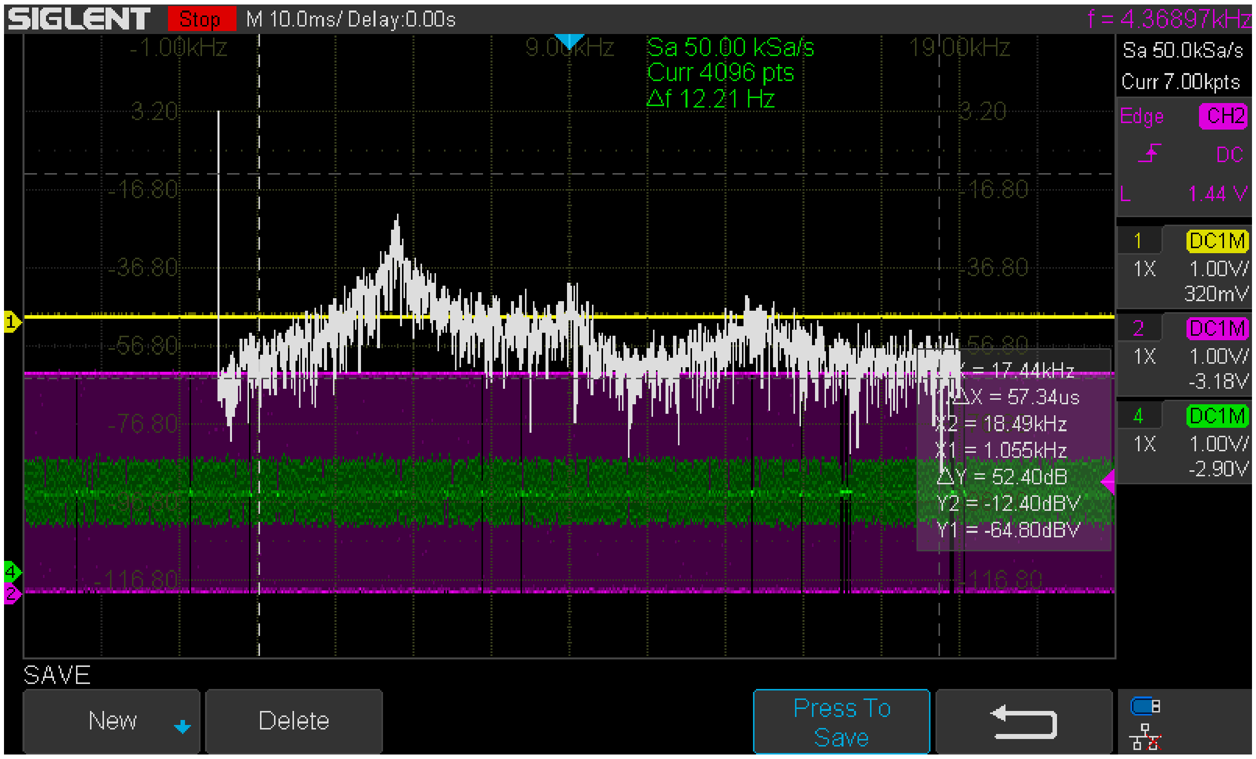


Figure 36. Frequency spectrum of optical receiver with no signal (pre-noise investigation)

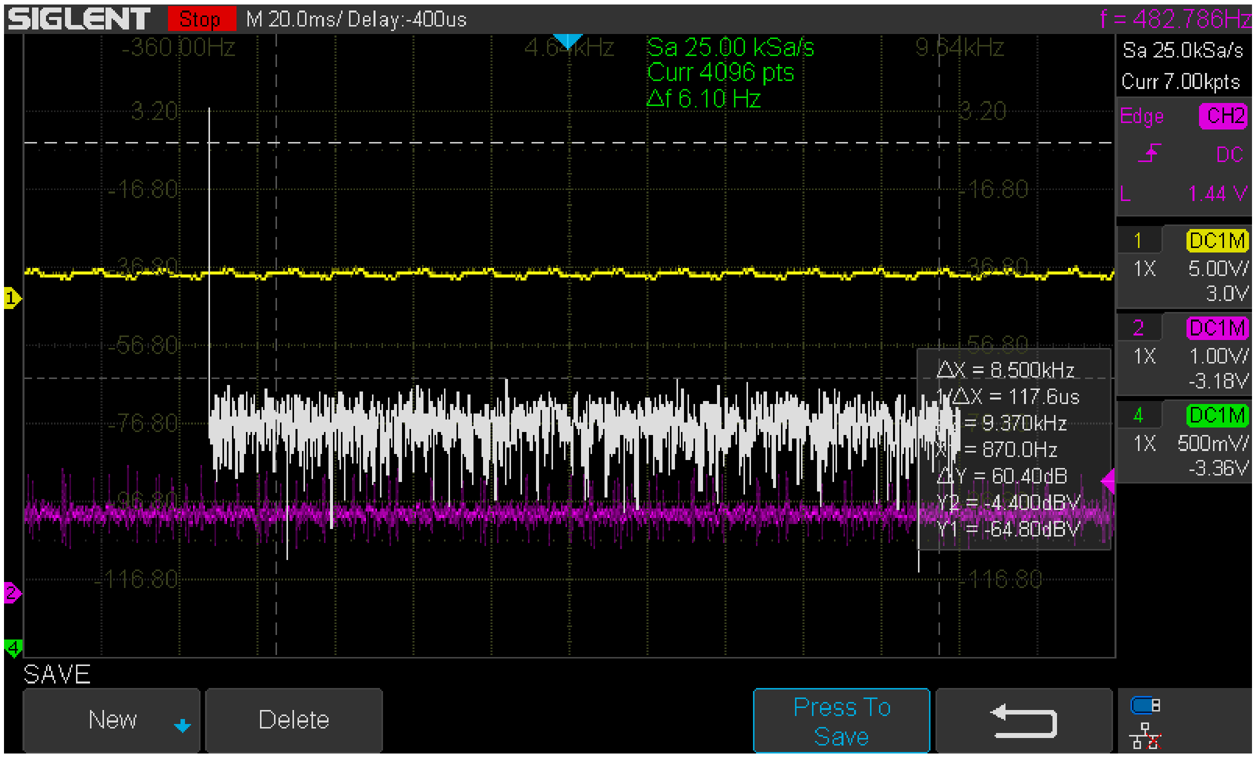


Figure 37. Frequency spectrum of optical receiver with no signal (post-noise investigation)

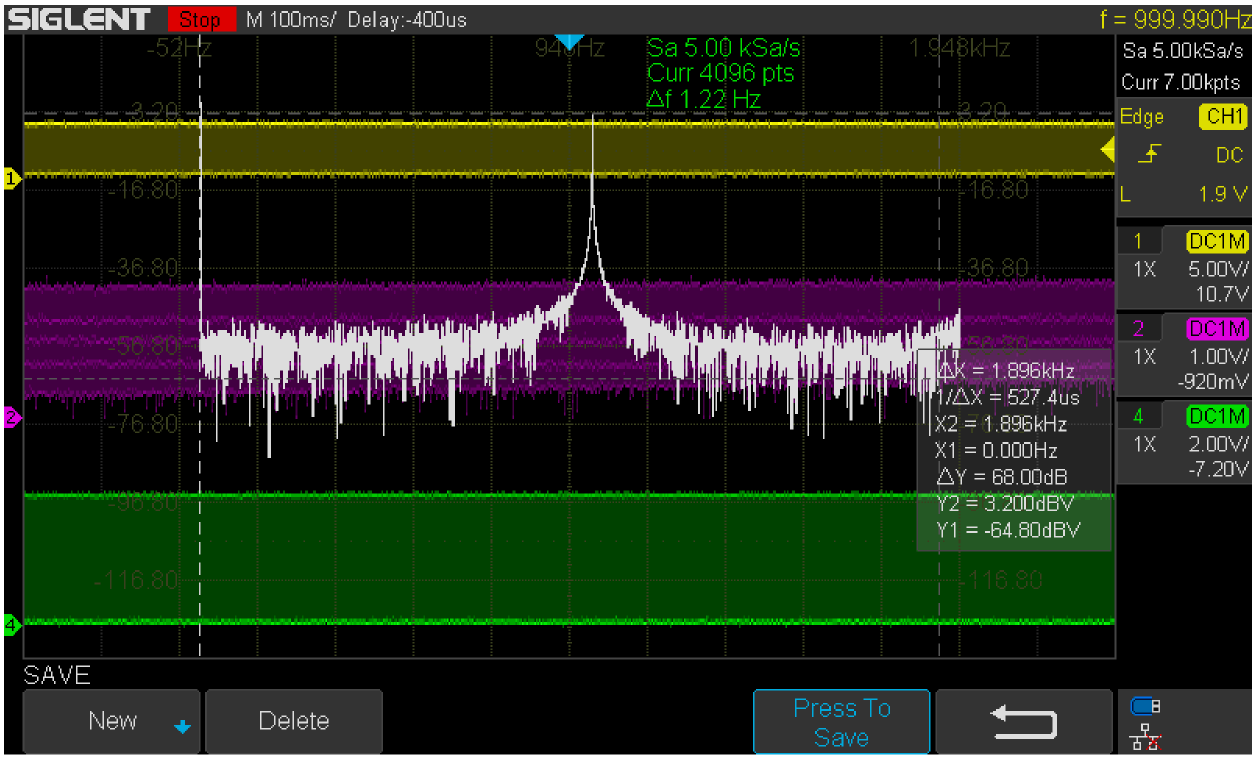


Figure 38.System noise with IR light source at 1kHz and ambient LED light

4.1.3 Powerline Modulation Communication Test

A test was performed to validate the operation of the powerline communication board. The system test setup is shown **Figure 39**. Information is successfully transmitted over powerline following the communication process detailed in **3.1.3 Powerline Communication Network**. **Figure 40** shows the messages monitored on serial terminals on the computer with data being sent between the two powerline communication units successfully. Code used for this test is located in **Appendix C** and is main.c and St7580_appli.c for the powerline communication test.

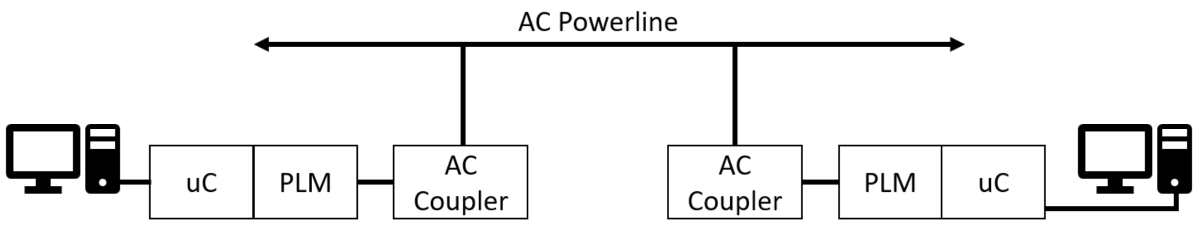
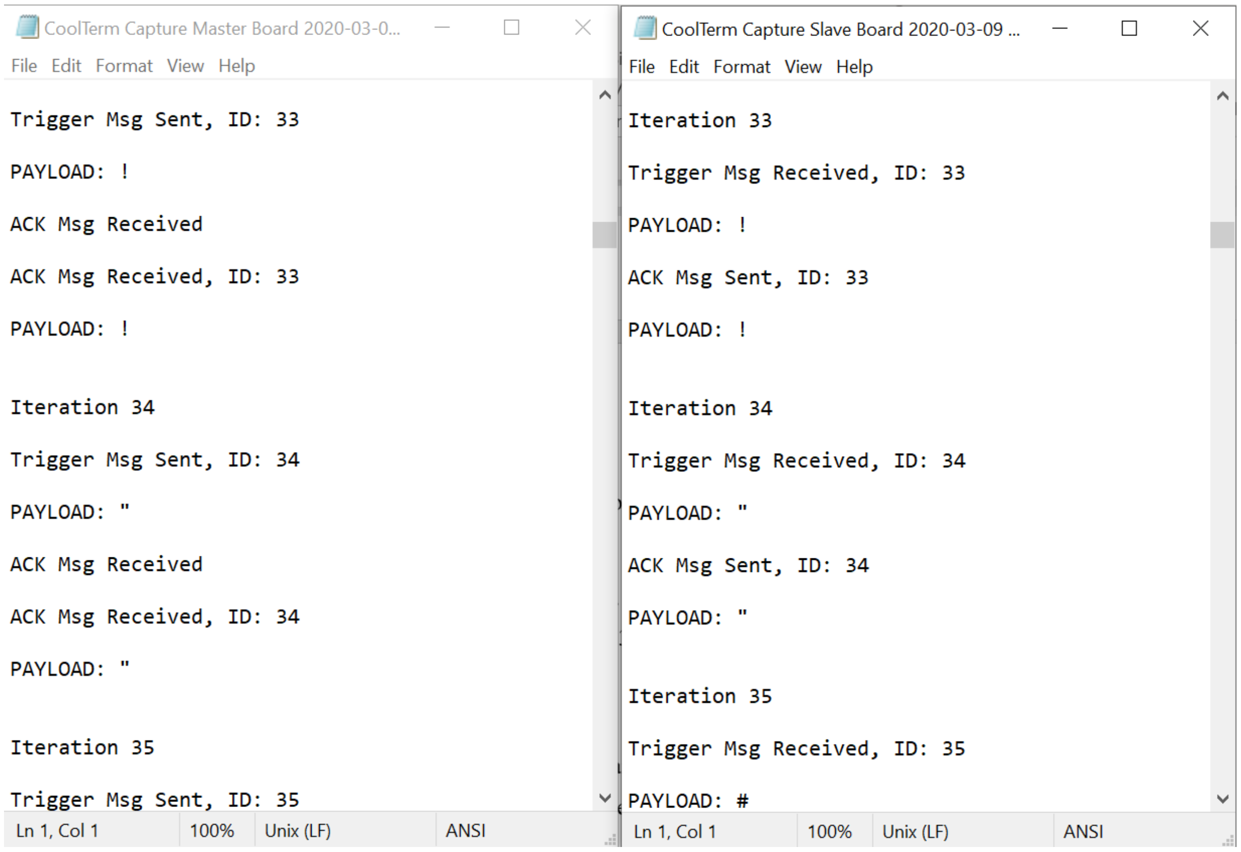


Figure 39. Powerline communication test setup


CoolTerm Capture Master Board 2020-03-0...
File Edit Format View Help
Ln 1, Col 1
100%
Unix (LF)
ANSI

```
Trigger Msg Sent, ID: 33

PAYLOAD: !

ACK Msg Received

ACK Msg Received, ID: 33

PAYLOAD: !


Iteration 34

Trigger Msg Sent, ID: 34

PAYLOAD: "

ACK Msg Received

ACK Msg Received, ID: 34

PAYLOAD: "


Iteration 35

Trigger Msg Sent, ID: 35
```

CoolTerm Capture Slave Board 2020-03-09 ...
File Edit Format View Help
Ln 1, Col 1
100%
Unix (LF)
ANSI

```
Iteration 33

Trigger Msg Received, ID: 33

PAYLOAD: !

ACK Msg Sent, ID: 33

PAYLOAD: !


Iteration 34

Trigger Msg Received, ID: 34

PAYLOAD: "

ACK Msg Sent, ID: 34

PAYLOAD: "


Iteration 35

Trigger Msg Received, ID: 35

PAYLOAD: #
```

Figure 40. Powerline communication data

4.1.4 Power Distribution Board

The power distribution board, shown in **Figure 28**, was tested using the Fluke 107 multimeter to measure the outputs on the circuit. The measured outputs are listed in **Table 3**.

Table 3. Power distribution board performance

	Expected (V)	Measured (V)
Vout_5	5	6.1
Vout_12	12	12.03
Vout_36	36	36.16

As shown in **Table 3**, the voltage outputs on the circuit are what is expected except that the 5V output on the 5V/12V dual power converter is 6V. This is not ideal, but the higher voltage should not change the operation of the circuits connected to this voltage. It is uncertain whether this is a manufacturer defect or that the manufacturer datasheet is inaccurate. After system integration it became apparent that there were also minor issues with grounding on the power distribution board. Switching from independent power supplies to the board added a small visible ripple on all boards connected to the power distribution board.

Within the lab environment there are no issues powering the board on. After this board was brought home for testing due to COVID-19 related lab closure, it was not possible to turn the board on without tripping the circuit breaker. Since further testing is not possible, the source of this issue is not clear. The board was brought to two separate houses and powered on resulting on the circuit breaker being tripped.

4.1.5 Manchester Communication Test

A test was used to evaluate the transmission of Manchester encoded data from an infrared light source to the receiver circuit, passing from the photodiode through to the comparator, full communication link shown in **Figure 41**, test setup shown in **Figure 42**. To transmit a Manchester encoded signal, an arbitrary waveform was generated in MATLAB, uploaded to a Koolertron JDS6600 Signal Generator, transmitted and then received by the optical receiver circuit, and decoded by a Picoscope 2204A using Picoscope 6 software.

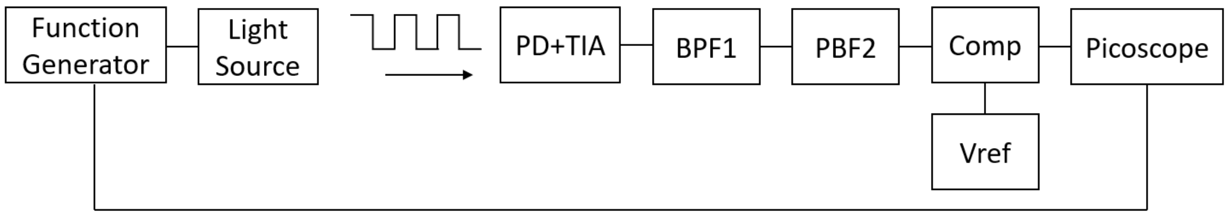


Figure 41. Manchester encoded data communication link

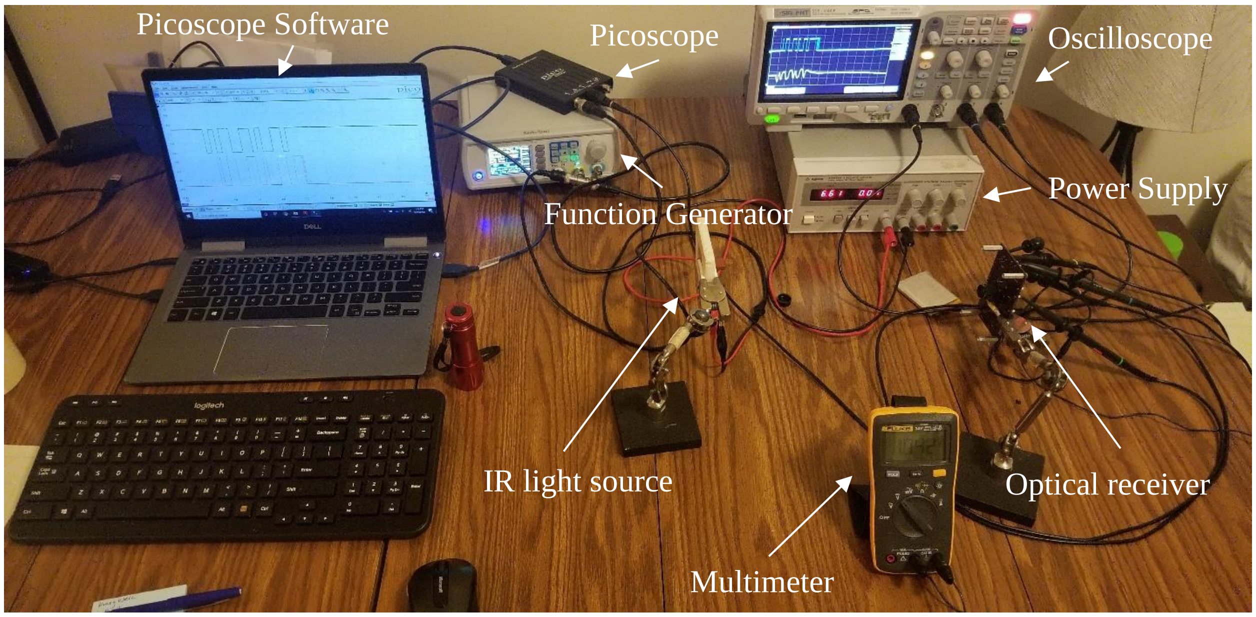


Figure 42. Manchester encoded data communication test setup

The MATLAB script generate_manchester_waveform.m was used to generate the Manchester encoded data and the scripts output is in **Appendix B**. To upload the CSV file to the signal generator the JDS6600 software was utilized. The arbitrary waveform function outputs the data at a speed and amplitude set by the frequency and amplitude generator display. To calculate the appropriate speed for the signal generator the relation between encoded and non-encoded bits was used. In example “1010’ becomes “01100110” when encoded, an example calculation is shown **Equation 43**. **Table 4** shows the output frequency for each bit rate tested.

$$Bandwidth\ of\ 1\ Manchester\ encoded\ bits = 2\ \times Bandwidth\ of\ 1\ bit$$

$$1\ Manchester\ encoded\ bit = 8\ samples$$

$$1\ bit\ =\ 4\ samples$$

$$Total\ bits\ = \frac{Samples}{4} = \frac{2048}{4} = 512\ bits$$

$$bit\ rate\ = \frac{bit}{second} = bit \times f$$

$$bit\ rate = 512\ bits \times f$$

Equation 43. Bit rate calculations

Table 4. Function generator setting for bit rate

Bit rate	Function Generator Fout (Hz)
9600	18.75
7200	14.0625
4800	9.375
3600	7.03125
2400	4.6875

IR LED light source, **Figure 25**, is used to send the messages with the function generator set to output Manchester encoded data through the arbitrary waveform function. Both the input and output signal are monitored by the Picoscope to make use of its Manchester decoding function. An example of what this measurement looks like is shown **Figure 44**. **Table 5** shows the results of decoding messages sent at 4800 bits per second, data for other transmission speeds is located in **Appendix B**. **Table 6**, provides a summary of the bit error of transmission and decoding of the data.

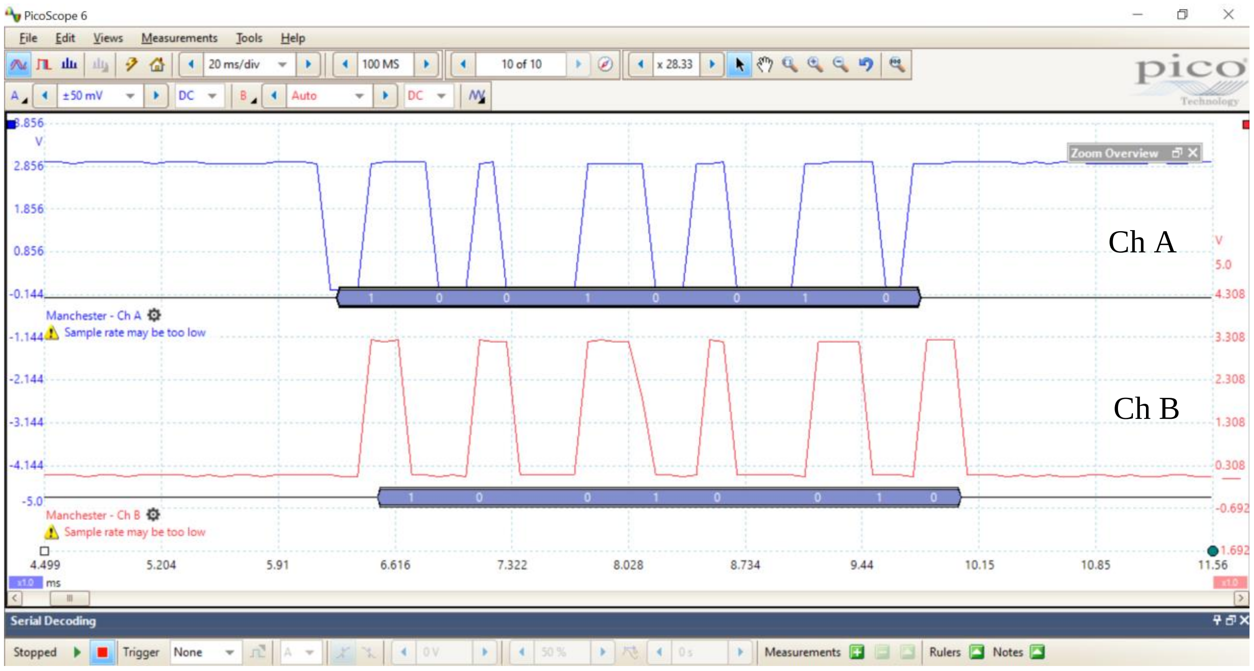


Figure 44.Picoscope decoding Manchester data at 4800 bps, Ch A (blue) input and Ch B (red) output

Table 5. Manchester encoded and decoded data packets at 4800 bps

	Input			Output			Bit Error	
Packet	Bin	Dec	ASCII	Bin	Dec	ASCII	Transmitted	Decoded
1	10010010	73	I	10010010	73	I	0/8	0/8
2	11010010	75	K	11010010 1	75 1	K SOH	0/8	1/9
3	10110010	77	M	10010010	73	I	0/8	1/8
4	11110010	79	O	11110010	79	O	0/8	0/8
5	10000010	65	A	10000010	65	A	0/8	0/8
6	11000010	67	C	11000010	67	C	0/8	0/8
7	10100010	69	E	10100010 1	69 1	E SOH	0/8	1/9
8	11100010	71	G	11100010	71	G	0/8	0/8
9	10010010	73	I	10010010	73	I	0/8	0/8
10	11010010	75	K	11010010	75	K	0/8	0/8
11	10110010	77	M	10010010	73	I	0/8	1/8
12	11110010	79	O	11110010 1	79 1	O SOH	0/8	1/9
13	10000010	65	A	10000010	65	A	0/8	0/8
14	11000010	67	C	11000010	67	C	0/8	0/8
					Average		0%	4%

Table 6. Bit error rate of optical receiver at different speeds

	Average Bit Error	
Speed	Transmission	Decoding
2400	0%	31%
3600	0%	40%
4800	0%	4%
7200	0%	36%
9600	0%	43%

Within **Table 6**, the bit errors were broken into two sections the error from transmission and the error in decoding. Error in transmission was determined by observing the sent and received packet data and counting the number of ‘01’ and ‘10’ transitions to see if any data was lost over transmission. Decoding bit error rate was determine by examining the amount of bit errors that occurred within decoding process from the Picoscope software. The average was calculated based off the number of packets sent within the test at each bit rate. In example, the percent in **Table 6** for 4800 bit rate uses the packets listed in **Table 5**. **Figure 45** shows an example of the phenomenon when all the packet data is transmitted but not properly decoded. Channel A is the input from the function generator and Channel B is the output at the comparator of the optical receiver circuit, the logic is inverted due to the relationship between the light output of the IR led and the received signal at the photodiode. Within the received packet the decoding is in error due to a change in the pulse width. The pulse transition is configurable and set by the potentiometer input on the comparator. The set point is determined by the average voltage of the signal which is 1.04 V. **Table 7** shows measured pulse width and the difference between the input to the system which is successfully decoded and the output of the system which follows the bit error rate shown in **Table 6**.

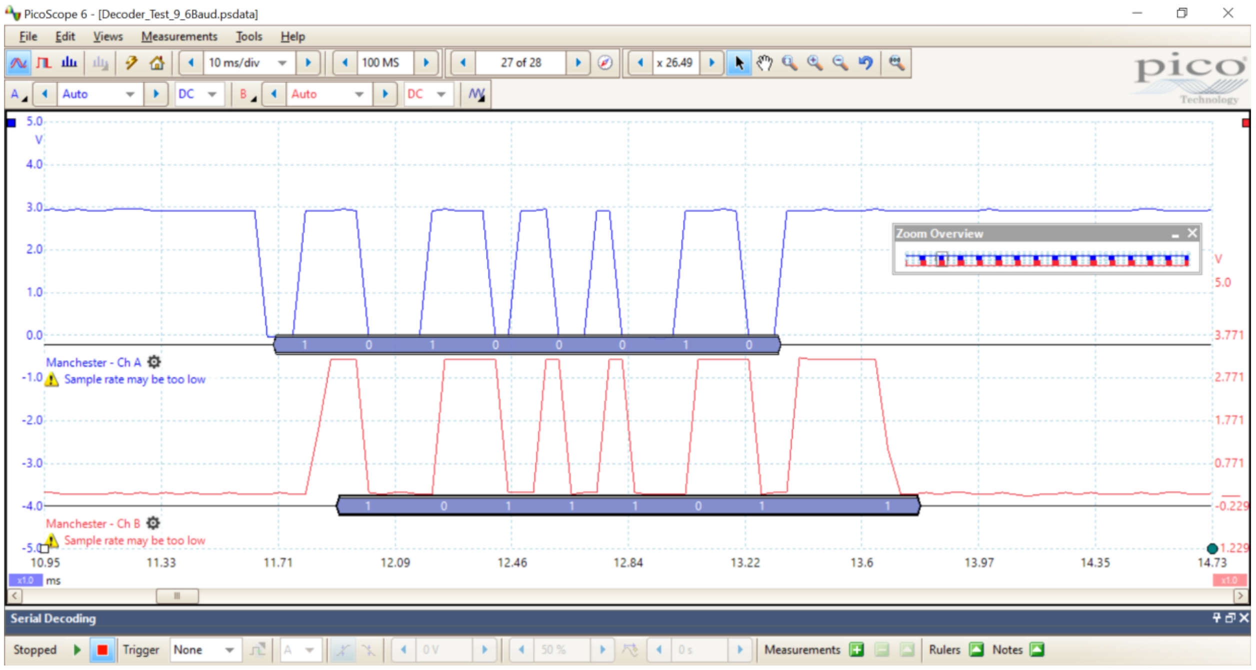

PicoScope 6 - [Decoder_Test_9_6Baud.psdata]
File Edit Views Measurements Tools Help
10 ms/div
100 MS
27 of 28
x 26.49
Auto DC Auto DC
Manchester - Ch A
Sample rate may be too low
Manchester - Ch B
Sample rate may be too low
Zoom Overview
Serial Decoding
Stopped Trigger None Measurements Rulers Notes

Figure 45. Packet data at 9600 bit rate

Table 7. Pulse widths at different bit rates

	Input Pulse Width (us)				Output pulse width (us)			
Bit Rate	101	010	0110	1001	101	010	0110	1001
9600	91.62	123.1	214.6	211.4	76	140	196	224
7200	124.6	171.8	292.1	240.5	104	180	252	296
4800	241.7	253.5	412.7	424.5	190	290	300	420
3600	250.2	323.8	493.1	574	310	310	350	560
2400	326.1	505.9	832	809.5	360	440	400	1280

From the results of **Table 7** and our ability to change the voltage setpoint on the comparator, it appears that further options to tune the results are present. This approach was unsuccessful and in **Figure 46** this can be seen. Increasing or decreasing the voltage setpoint decreases the pulse width causing more possibility for error from the decoding algorithm in the Picoscope software as it is decodes in relation to the speed for which it is set.

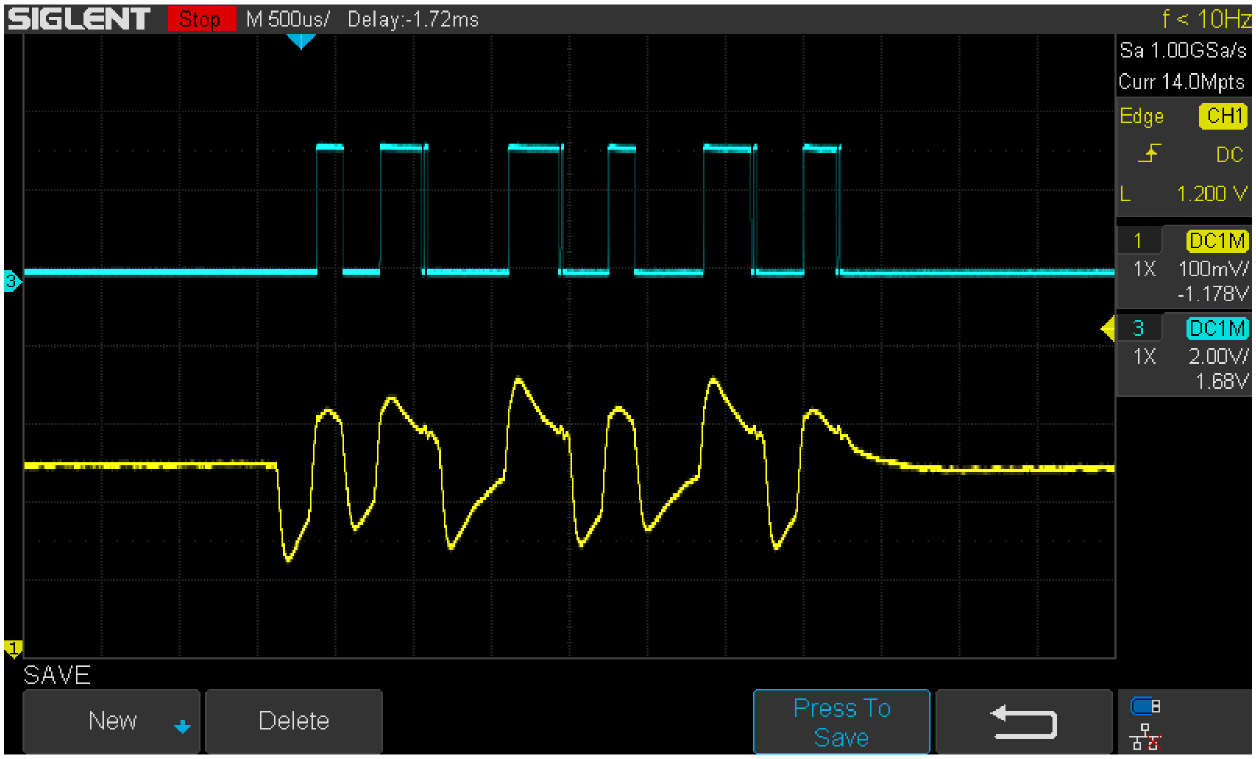


Figure 46. Manchester decode test comparator input (yellow) and output (blue)

4.1.6 Powerline Serial Communication Test

A test was used to demonstrate that powerline serial communication could occur at different speeds, a diagram of the communication link is shown **Figure 47**. CoolTerm serial port application was used to send a message from the computer to the microcontroller and powerline modem. Modem to modem communication occurs through the powerline and microcontroller relays the data to the LED driver to transmit the light encoded message. The test setup is shown **Figure 48**. The summarized results of the test are shown **Table 8**, because CoolTerm is used the carriage return (CR) and line feed (LF) are shown in the input data but are not in the data sent over the powerline. The raw results are located in **Appendix B** under Serial Decoding Test.

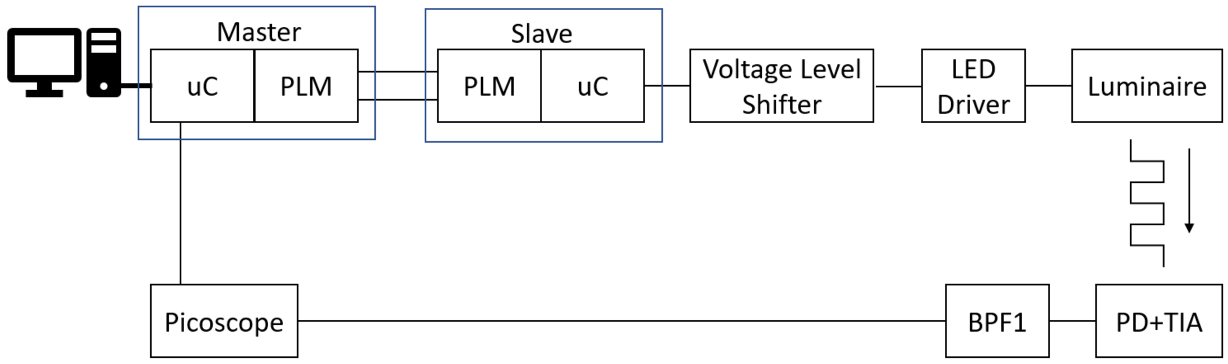


Figure 47. Functional blocks within serial communication link

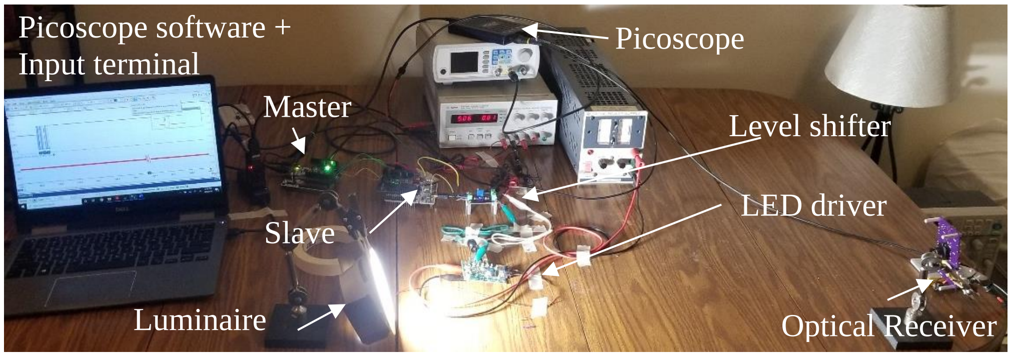


Figure 48. Serial communication test setup

Table 8. Serial decoding test at different bit rates

		Input				Output				
Bit Rate	Packet	Bin	Hex	Dec	ASCII	Bin	Hex	Dec	ASCII	Bit Error
2400	1	10101100	35	53	5	10101100	35	53	5	0/8
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
3600	1	10101100	35	53	5	10101100	35	53	5	0/8
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
4800	1	10101100	35	53	5	10101100	35	53	5	0/8
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
7200	1	10101100	35	53	5	10101100	35	53	5	0/8
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
9600	1	10101100	35	53	5	10101100	35	53	5	0/8
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					

CHAPTER 5. SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Conclusions and recommendations about the optical communication testbed and the development of the optical receiver board, Manchester communication link, and powerline communication link are discussed in this chapter.

5.1 Conclusion

This section details the conclusions drawn from the development and test of the optical communication testbed, this includes the optical receiver board, power distribution board, Manchester communication link, and powerline communication link. The transmission of Manchester encoded optical data was successfully achieved from an IR light source to the optical receiver at 4800 bit rate with 4% errors in decoding present in all the data packets recorded. The transmission of serial encoded data from a computer to a device within the optical communication testbed was successfully achieved at bit rates of 2400, 3600, 4800, 7200, and 9600 with no errors present. The transmission of decoded optical information from the receiver to powerline modem via FPGA was not achieved due to Purdue University closure of on campus activities due to COVID-19. The working components of an optical communication testbed were all realized except for the FPGA hardware to link the microcontroller and the optical receiver.

5.1.1 Optical Receiver Board

The optical receiver board developed in this thesis is able to receive Manchester encoded data optically. Each functional block within the optical receiver was able to be independently verified for operation. The FPGA in the optical receiver was not operational to decode Manchester data into serial data and transmit to the microcontroller.

Noise testing was conducted on the optical receiver. System noise was caused by poor routing of a clock signal causing noise to infect the circuit. Removing the clock and the filter it set removed this noise from the circuit. Noise from oscillation also occurred on the circuit. The gain on the first gain stage was reduced to 4.8 dB to create a preamplification stage and this resulted in oscillation being removed. The remaining noise in the circuit was reduced by placing a 100 nF balance capacitor in parallel with the photodiode in the transimpedance amplifier. **Figure 49** shows the final tested version of the optical receiver circuit.

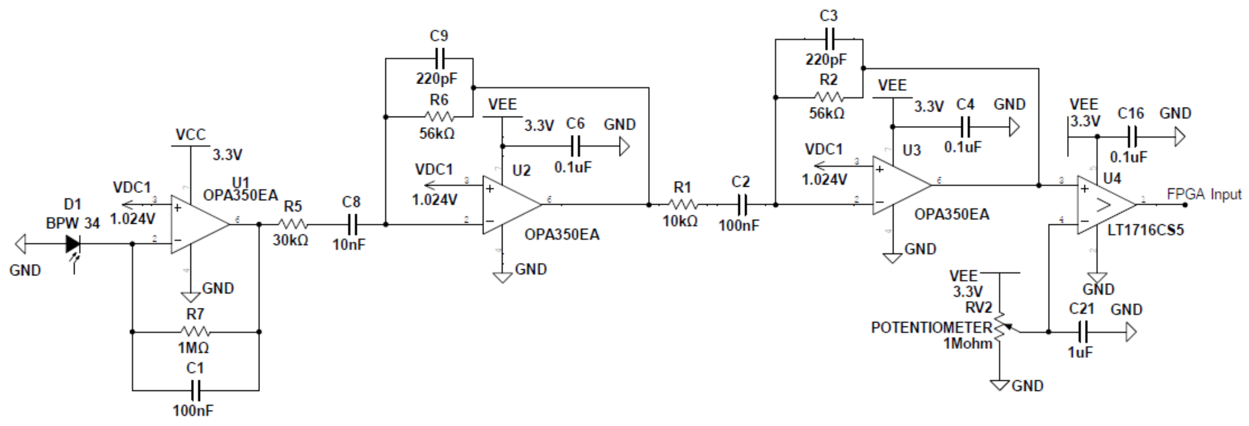


Figure 49. Final tested optical receiver

5.1.2 Optical Communication Testbed

Manchester communication tests were conducted on the optical receiver. All data was successfully transmitted from the light source to the output of the optical receiver. At 4800 bit rate the decoding of all the Manchester encoded data was successful with an error rate of 4%. The other bit rates tested, 2400, 3600, 7200, and 9600, had varying levels of success with percent error ranging from 31% to 43%.

Powerline communication tests were conducted on the optical communication testbed at bit rates of 2400, 3600, 4800, 7200, and 9600. The powerline communication tests successfully transmitted a byte over a powerline at all speeds tested with no bit errors. The transfer of the serial data to the FPGA in the optical receiver was not achieved.

5.3 Recommendations

The development of FPGA hardware and system integration was significantly more complex than anticipated. More time allocation to this process and discussion with lab members on this subject is necessary for successful implementation.

The Manchester encoded data faced issues with decoding. It is recommended that the bandpass filters implemented within the circuit use a Bessel response to minimize the group delay on the signal. The optical receiver also faced issue where the gain was reduced on a filter to reduce noise from oscillations. It is recommended to use a transimpedance amplifier circuit configuration with gain to act as a preamplifier in the circuit. On the optical receiver board the gain required in the circuit was unknown, due to this four gains stages were developed with the functionality to added and remove gain if necessary. It is recommended for the distance designed in the optical communication testbed to remove the third and fourth gain stage from the circuit. Another recommendation for gain within the optical receiver is to implement an automatic gain control circuit so that no matter the location or signal strength received the gain added will always be bring the signal to a predetermined level.

Powerline modulation transmission successfully occurred using the STM32 environment. The documentation on this system is lacking causing significant time and understanding to make the system less of a blackbox. It is recommended to evaluate whether the communication handling used within the systems API's is necessary. If it is deemed unnecessary, further research into other options is needed. An option that could be an attractive route to pursue is the use DSP development boards and AC coupling.

REFERENCES

Acuity Brands. (2019). *LED RETROFIT BAFFLE TRIM DOWNLIGHT RETROFIT FOR STANDARD 4 ”*.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, *17*(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Bridgelux. (2018). *Bridgelux ® Gen 7 Vero ® 13 Array Introduction*.

Cailean, A. M., Cagneau, B., Chassagne, L., Dimian, M., & Popa, V. (2014). Miller code usage in visible light communications under the PHY I layer of the IEEE 802.15.7 standard. *IEEE International Conference on Communications*, 1–4. https://doi.org/10.1109/ICComm.2014.6866699

Chawla, Vipul and Ha, D. S. (2007). *An Overview of Passive RFID*. *IEEE Applications & Practice*.

Cisco Systems. (2016). *At-a-Glance Internet of Things*. Retrieved from www.cisco.com/go/iot.

Cypress Semiconductor. (2011). What is Power Line Communication? | EE Times. Retrieved November 11, 2019, from https://www.eetimes.com/document.asp?doc_id=1279014#

Haas, H. (2018). Reviews in Physics LiFi is a paradigm-shifting 5G technology. *Reviews in Physics*, *3*(October 2017), 26–31. https://doi.org/10.1016/j.revip.2017.10.001

Haas, H., Yin, L., Wang, Y., & Chen, C. (2016). What is LiFi? *Journal of Lightwave Technology*, *34*(6), 1533–1544. https://doi.org/10.1109/JLT.2015.2510021

Kaushik, S. (2012). *An overview of Technical aspect for WiFi Networks Technology*. Retrieved from www.ijecse.org

Keim, R. (2016). Manchester Encoding: What Is It, and Why Use It? - Technical Articles. Retrieved November 11, 2019, from https://www.allaboutcircuits.com/technical-articles/manchester-encoding-what-is-it-and-why-use-it/

Langer, K., & Grubor, J. (2007). Recent Developments in Optical Wireless Communications. *IEEE International Conference on Transparent Optical Networks*, 146–151.

Leon-salas, W. D., & Fan, X. (2018). Live Demonstration : Modulating Luminescence Emissions of Solar Cells for Sensing and Identification. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1. https://doi.org/10.1109/ISCAS.2018.8351753

Leon-salas, W. D., & Fan, X. (2019). Exploiting Luminescence Emissions of Solar Cells for Optical Frequency Identification (OFID). *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. https://doi.org/10.1109/ISCAS.2018.8351139

Mouser Electronics. (2019). RF Wireless Technology | Mouser. Retrieved November 10, 2019, from https://www.mouser.co.uk/applications/rf-wireless-technology/

Patel, Mark, Shangkuan, Jason, and Thomas, C. (2017). What's new with the Internet of Things? | McKinsey. Retrieved November 6, 2019, from https://www.mckinsey.com/industries/semiconductors/our-insights/whats-new-with-the-internet-of-things#

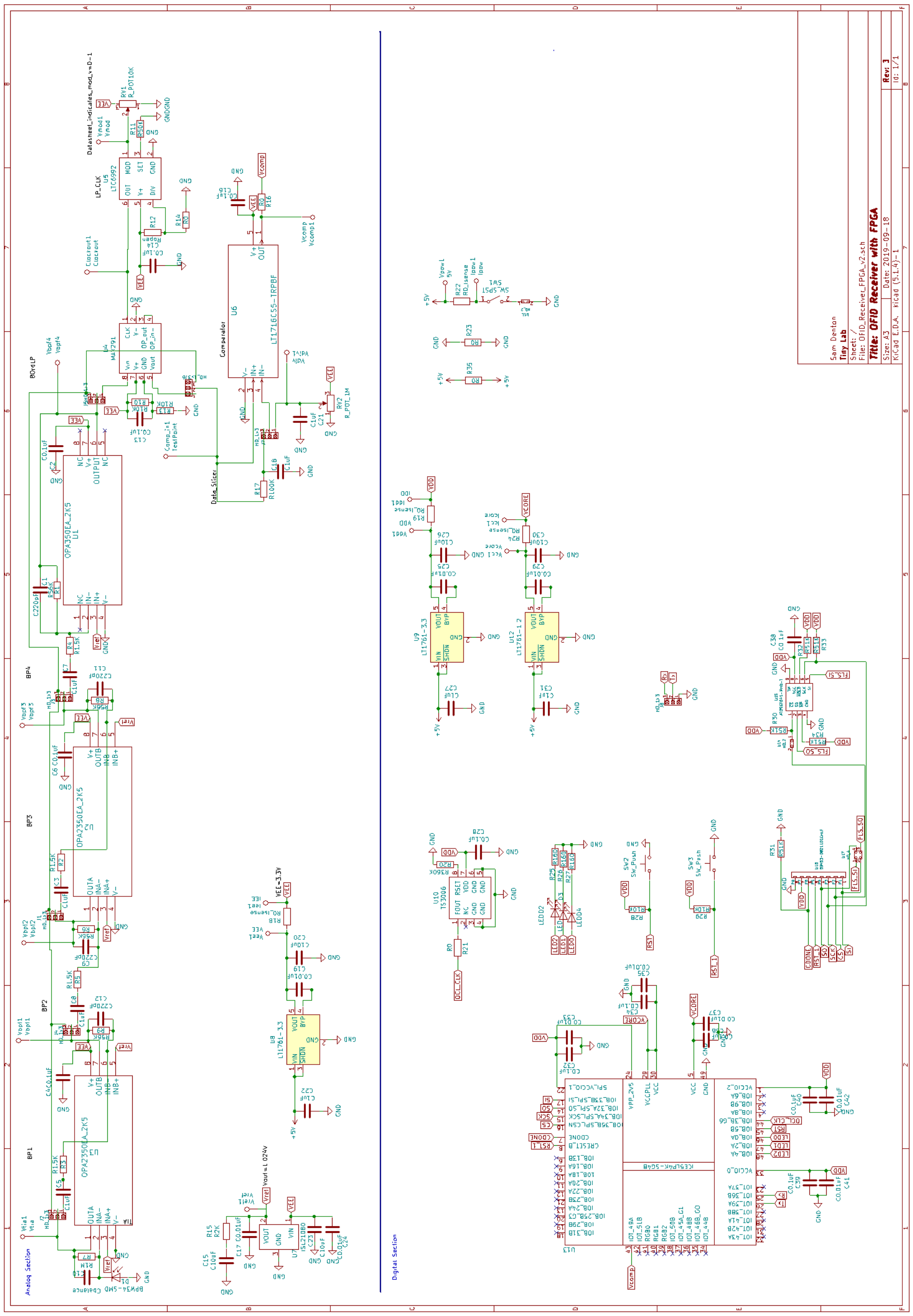
Sinclair, I. (2001). Transducing components. In *Passive Components for Circuit Design*. https://doi.org/10.1016/b978-075064933-9/50008-x

STMicroelectronics. (2017a). *UM2189 Getting started with the X-CUBE-PLM1 power line communication software expansion for STM32Cube*.

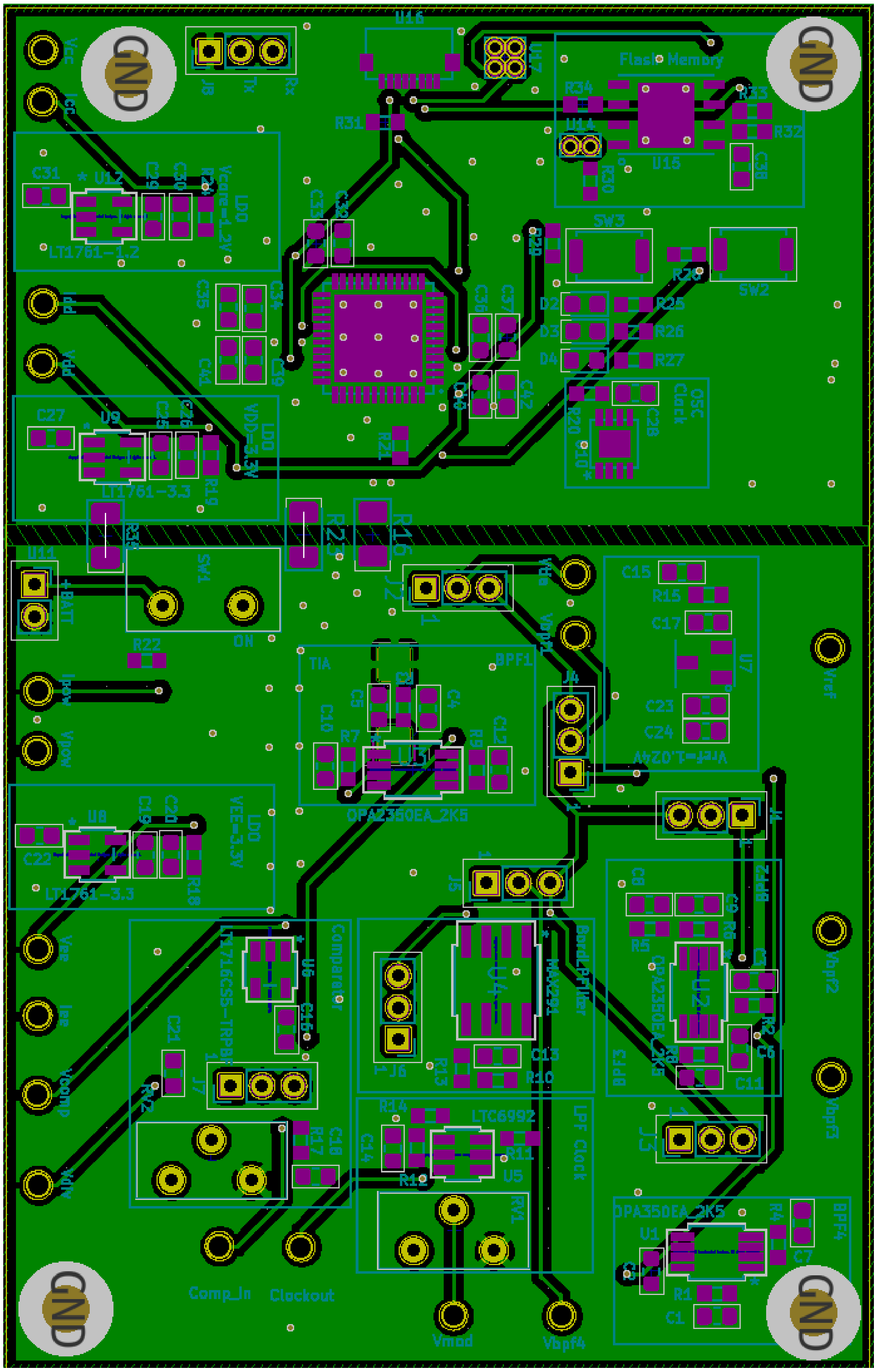
STMicroelectronics. (2017b). X-NUCLEO-PLM01A1, (June), 1–6.

APPENDIX A. PCB LAYERS OF RECEIVER

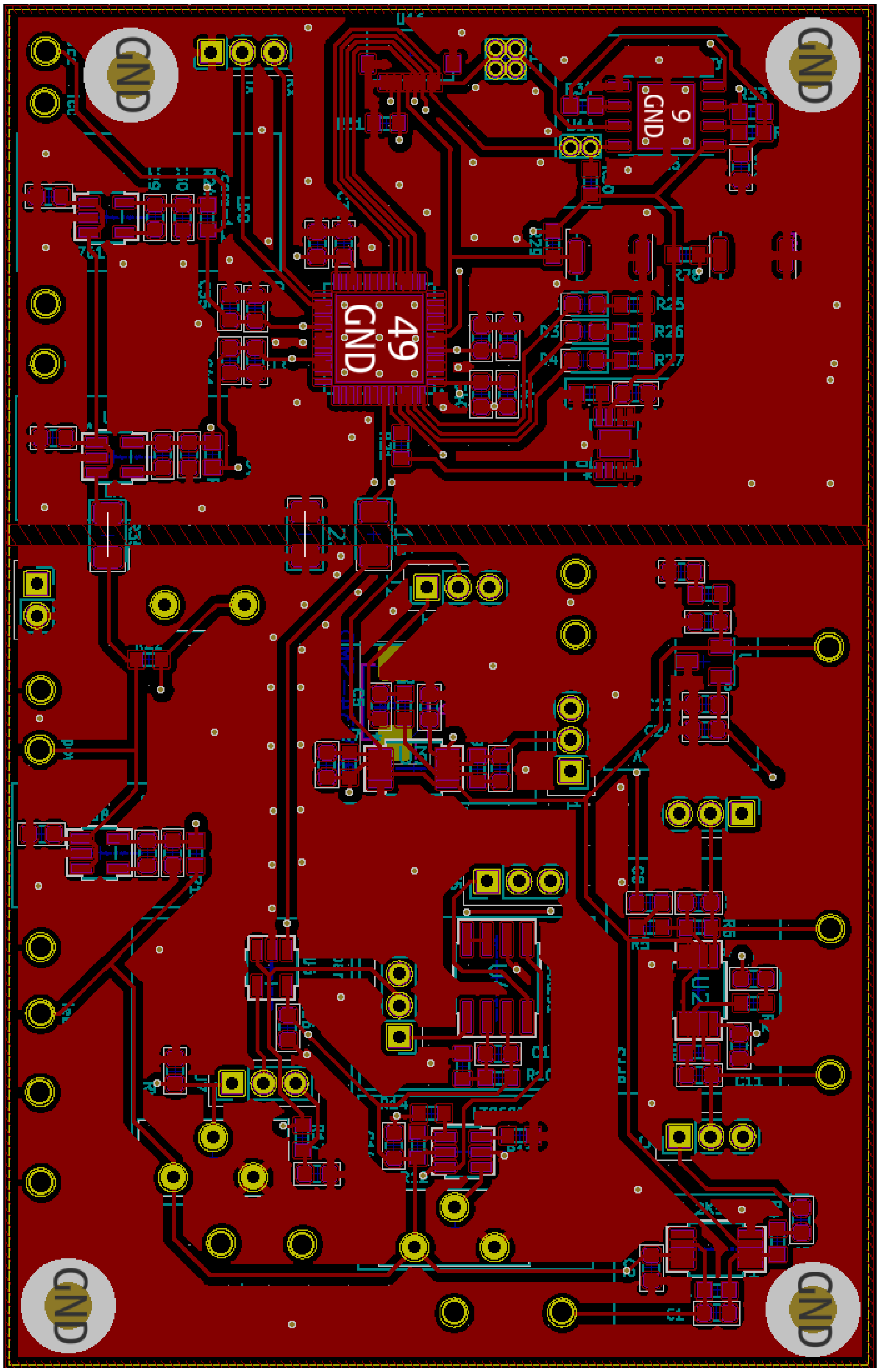
Schematic Capture of Receiver


Sam Denton
Tiny Lab
Sheet: /
File: OFID_Receiver_FPGA_v2.sch
Title: OFID Receiver with FPGA
Size: A3 Date: 2019-09-18
KiCad E.D.A. kicad (5.1.4)-1
Rev: 1
Id: 1/1
Analog Section
Digital Section
Comparator
LT1716CS5-TRPBF
OPA350EA_2K5
OPA2350EA_2K5
LTC6992
MAX7401
LT1761-3.3
LT1761-1.2
ISL21080
iCE5LP4K-SG48
TS3006
BPW34-SMD Cdiodance
Data_Slice
Datasheet_indicates_mod_v=0-1

Bottom copper view of Receiver



Top copper view of Receiver



APPENDIX B. RECEIVER MEASUREMENTS

Power Measurements

Board Power

Battery	Theoretical	Actual
Output Voltage	3.7 V	3.81 V

Digital Power

Vdd LDO	Theoretical	Actual
Input Voltage	3.7 V	3.81 V
Output Voltage	3.3 V	3.28 V

Vcore LDO	Theoretical	Actual
Input Voltage	3.7 V	3.81 V
Output Voltage	1.2 V	1.20 V

Analog Power

Vee LDO	Theoretical	Actual
Input Voltage	3.7 V	3.81 V
Output Voltage	3.3 V	3.28 V

Vref	Theoretical	Actual
Input Voltage	3.3V	3.28 V
Output Voltage	1.024 V	1.021 V

Frequency Response Measurement

Matlab Script: tf_estimation.m

The following is an example of the transfer function estimation script which is used to determine the frequency response of a circuit from .csv files recorded using an oscilloscope.

```
%--------------------------------------------------------------------------
% manual transfer function estimation from a list of .csv files recorded
% with an oscilloscope. Each .csv file contains input and output
% (sinusoidal) waveforms. Input is on channel 2 and output is on channel 1

clear all;

datadir   = 'T:\dentons\Measurements\TF estimation\tia_11212019\';

scope_file_list = [1:1:50];
num_of_files = length(scope_file_list);

A = zeros(1, num_of_files);
P = zeros(1, num_of_files);
f = zeros(1, num_of_files);

for m=1:num_of_files
    %----------------------------------------------------------------------
    % read data from file
    filename = strcat('scope_', num2str(scope_file_list(m)), '.csv');
    [data, text, full] = xlsread(strcat(datadir,filename));
    t  = data(5:2000, 1);
    y  = data(5:2000, 3);
    x  = data(5:2000, 2);
    Ts = t(2) - t(1);
    fs = 1/Ts;

    X       = fft(x);
    Y       = fft(y);
    Xmag    = abs(X);
    Xphase  = angle(X);
    Ymag    = abs(Y);
    Yphase  = angle(Y);

    max_ind = find(Xmag == max(Xmag(2:end)));
    n       = max_ind(1);
    N       = length(X);

    A(m) = Ymag(n)/Xmag(n);
    P(m) = Yphase(n) - Xphase(n);
    f(m) = n*fs/N;

    if P(m) > (pi()/2)
        P(m) = P(m) - pi();
    else P(m) < (-pi()/2)
        P(m) = P(m) + pi();
    end
end
%-------------------------------------------------------------------------
% plot transfer function

figure;
subplot(2,1,1)
semilogx(f, 20*log10(abs(A)));
xlim([0 inf])
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Frequency Response of Transimpedance Amplifier')
grid on;

subplot(2,1,2)
semilogx(f,P*180/pi);
xlim([0 inf])
xlabel('Frequency (Hz)');
ylabel('Phase (deg)');
grid on;
```

Photodiode and Transimpedance Amplifer

Date:	11/20/2019
Time:	11:00PM EST

Instrument:	Siglent SDS 1104X-E
Cal date:	

Settings:

Memdepth:	7k
Filetype:	Matlab (.dat)

Instrument: Keysight33210A Function/Arbtitrary Waveform Generator

Cal date:

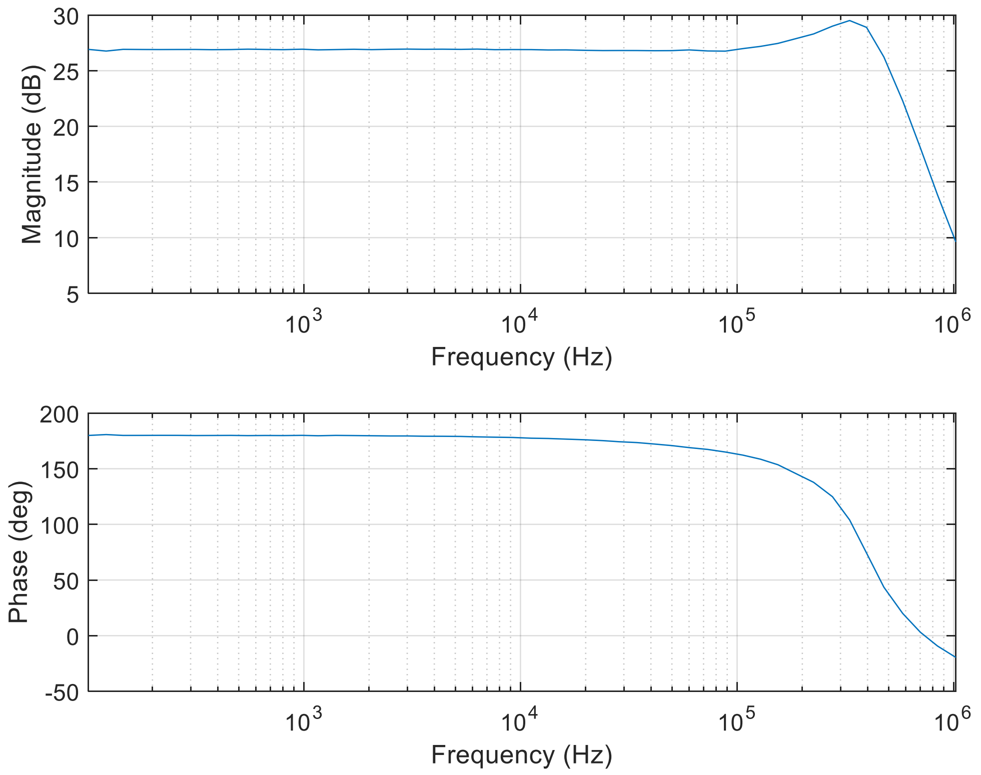
Settings:

Start Freq:	10 Hz
Stop Freq:	20 kHz
Steps:	50

Log steps determined by logspace(1, 4.3, 50) (matlab function)

Transimpedance Amplifier (TIA)

Op Amp:	TI OPA2350
Photodiode:	Vishay BW34
Lightsource:	Keysight33210A Function/Arbtitrary Waveform Generator
	SFH 4232A IR LED
	ALD1115pal MOSFET



Frequency Response of Transimpedance Amplifier

Bandpass Filter 1

Date:	4/20/2020
Time:	4:45PM EST

Instrument:	Siglent SDS 1104X-E
Cal date:	

Settings:

Memdepth:	7k
Filetype:	Matlab (.dat)

Instrument: Keysight33210A Function/Arbtitrary Waveform Generator

Cal date:

Settings:

Start Freq:	10Hz
Stop Freq:	1MHz
Steps:	50

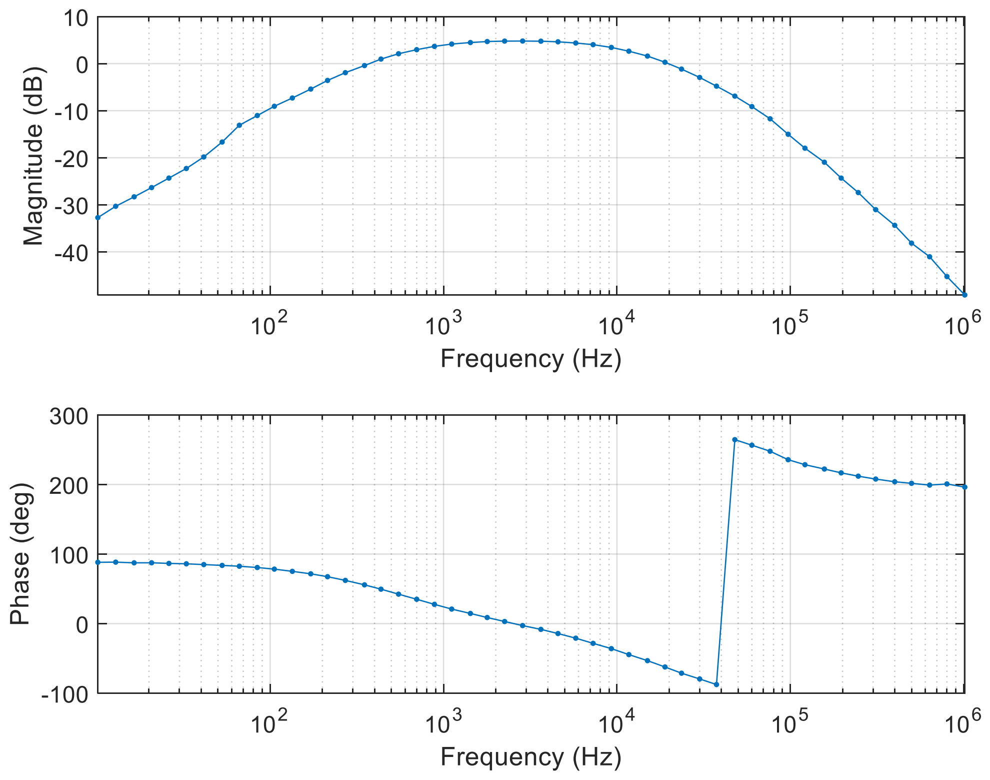
Log steps determined by logspace(1, 6, 50) (matlab function)

Bandpass Filter Gain Stage 1 (BPF1)

Op Amp: TI OPA2350

fc1: 106 Hz

fc2: 12.92 kHz



Frequency Response of Bandpass Gain Stage 1

Bandpass Filter 2

Date:	4/20/2020
Time:	5:30PM EST

Instrument:	Siglent SDS 1104X-E
Cal date:	

Settings:

Memdepth:	7k
Filetype:	Matlab (.dat)

Instrument: Keysight33210A Function/Arbtitrary Waveform Generator

Cal date:

Settings:

Start Freq:	10Hz
Stop Freq:	1MHz
Steps:	50

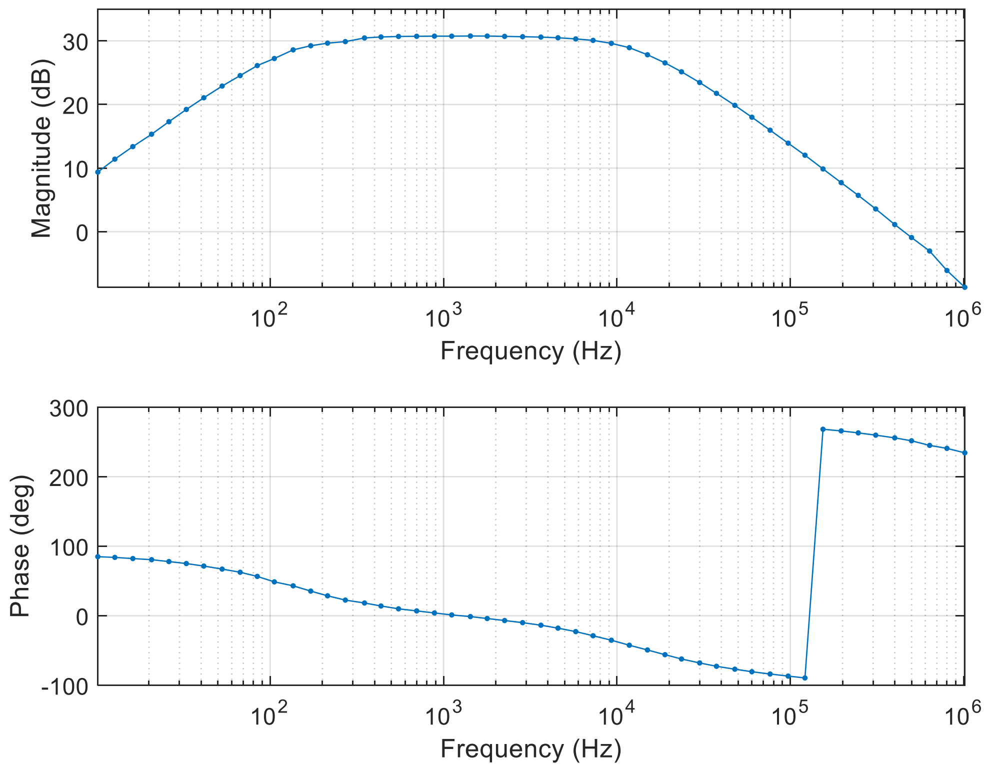
Log steps determined by logspace(1, 4.3, 50) (matlab function)

Bandpass Filter Gain Stage 2 (BPF2)

Op Amp: TI OPA2350

fc1: 106 Hz

fc2: 12.91 kHz



Frequency Response of Bandpass Gain Stage 2

Bandpass Filter 3

Date:	4/20/2020
Time:	6:00PM EST

Instrument:	Siglent SDS 1104X-E
Cal date:	

Settings:

Memdepth:	7k
Filetype:	Matlab (.dat)

Instrument: Keysight33210A Function/Arbtitrary Waveform Generator

Cal date:

Settings:

Start Freq:	10 Hz
Stop Freq:	20 kHz
Steps:	50

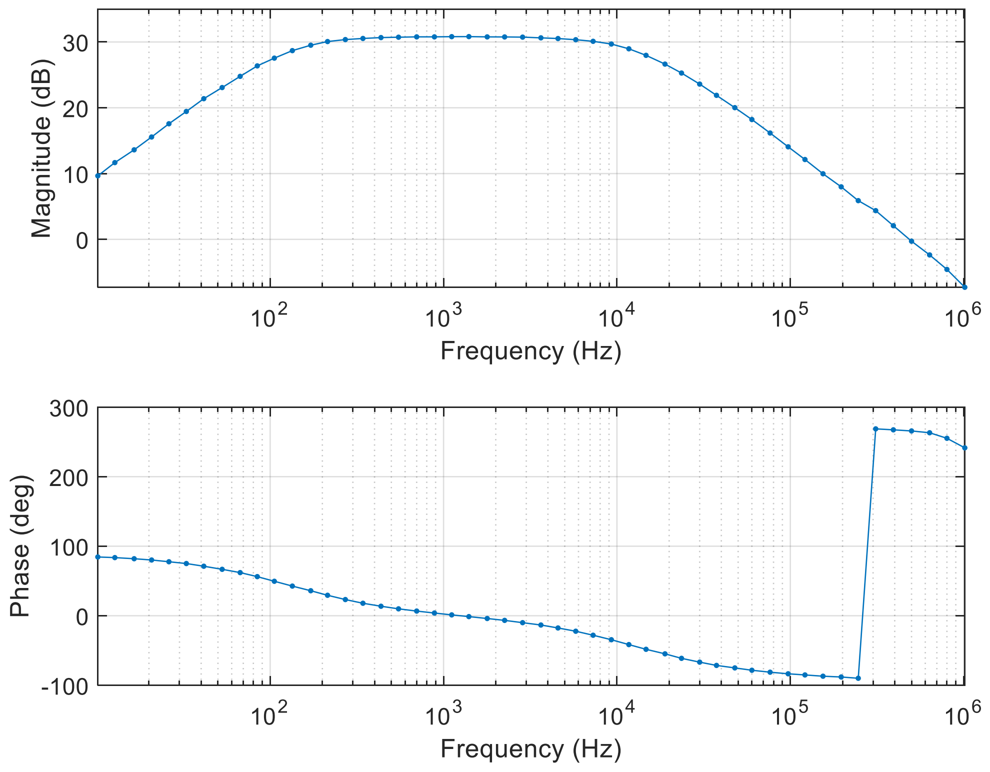
Log steps determined by logspace(1, 4.3, 50) (matlab function)

Bandpass Filter Gain Stage 3 (BPF3)

Op Amp: TI OPA2350

fc1: 106 Hz

fc2: 12.92 kHz



Frequency Response of Bandpass Gain Stage 3

Lowpass Filter

Date:	11/15/2019
Time:	2:20PM EST

Instrument:	Siglent SDS 1104X-E
Cal date:	

Settings:

Memdepth:	7k
Filetype:	Matlab (.dat)

Instrument: Keysight33210A Function/Arbtitrary Waveform Generator

Cal date:

Settings:

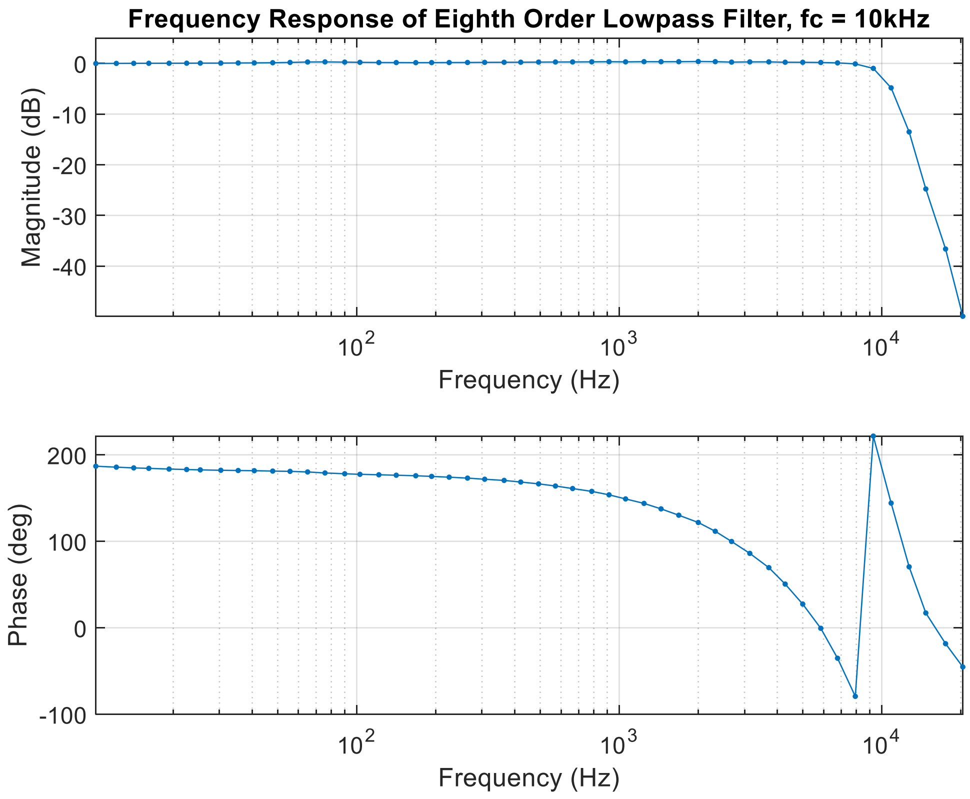
Start Freq:	10 Hz
Stop Freq:	20 kHz
Steps:	50

Log steps determined by logspace(1, 4.3, 50) (matlab function)

Eighth Order Lowpass Filter (LPF)

IC: MAX291

fc: 10 kHz

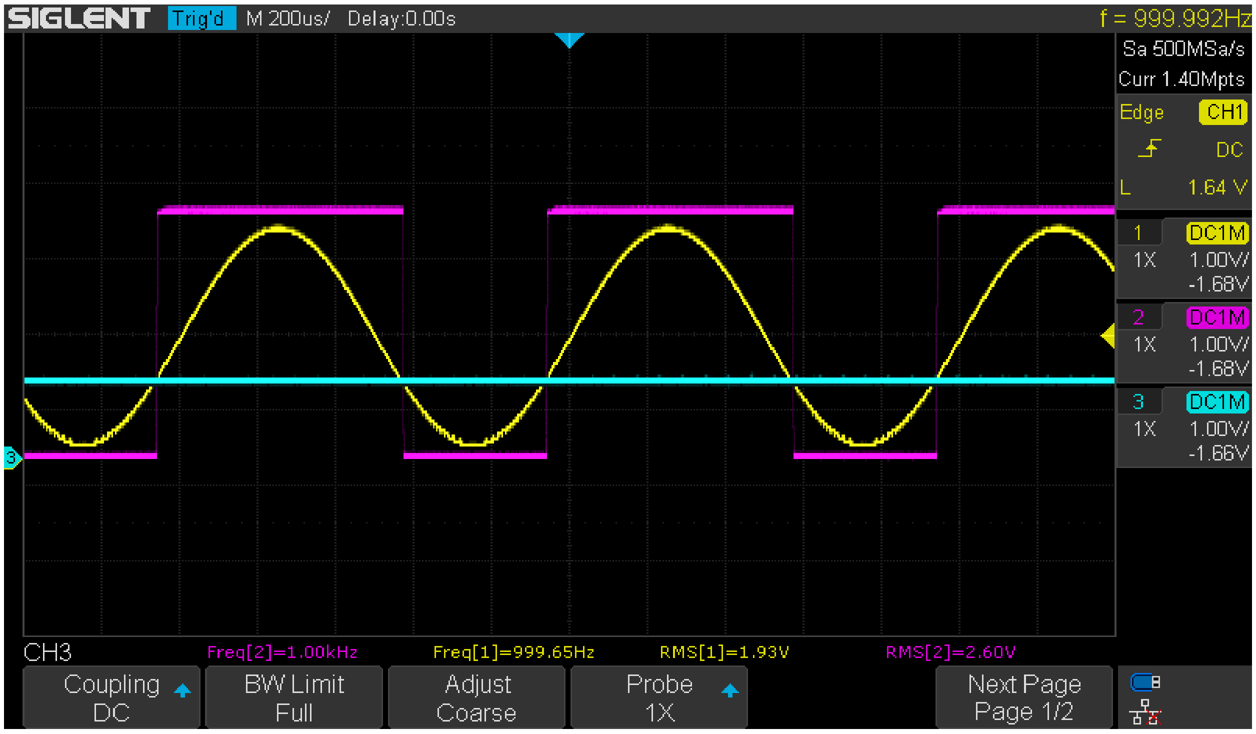

Frequency Response of Eighth Order Lowpass Filter, fc = 10kHz
Magnitude (dB)
0
-10
-20
-30
-40
10^2
10^3
10^4
Frequency (Hz)
Phase (deg)
200
100
0
-100
10^2
10^3
10^4
Frequency (Hz)

Comparator

Channel 1: Input

Channel 2: Output

Channel 3: Comparator Setpoint



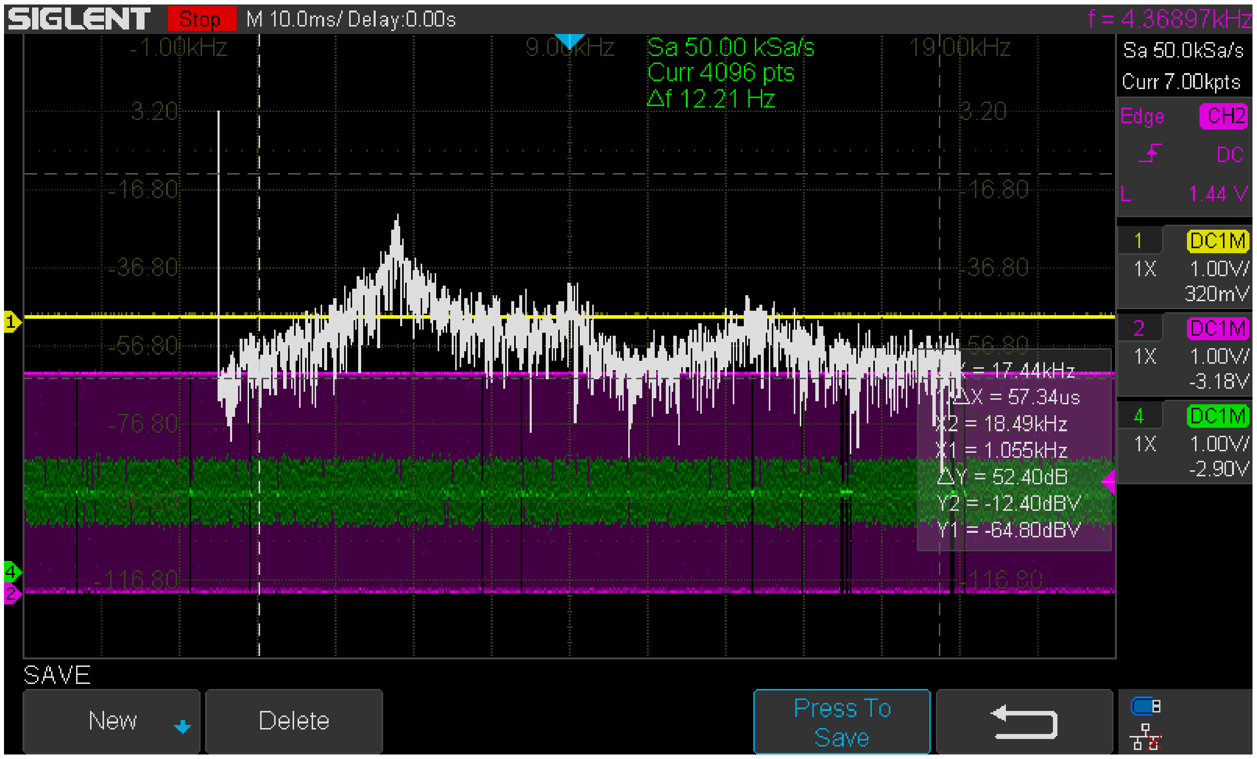
Optical Receiver Noise Investigation

Trace 1 (yellow): Light source signal

Trace 2 (purple): Comparator output

Trace 4 (green): BPF 3

FFT (white): from BPF3



M 10.0ms/ Delay:0.00s

f = 4.36897kHz

Sa 50.00 kSa/s
Curr 4096 pts
Δf 12.21 Hz

Sa 50.0kSa/s
Curr 7.00kpts

Edge CH2
DC
L 1.44 V

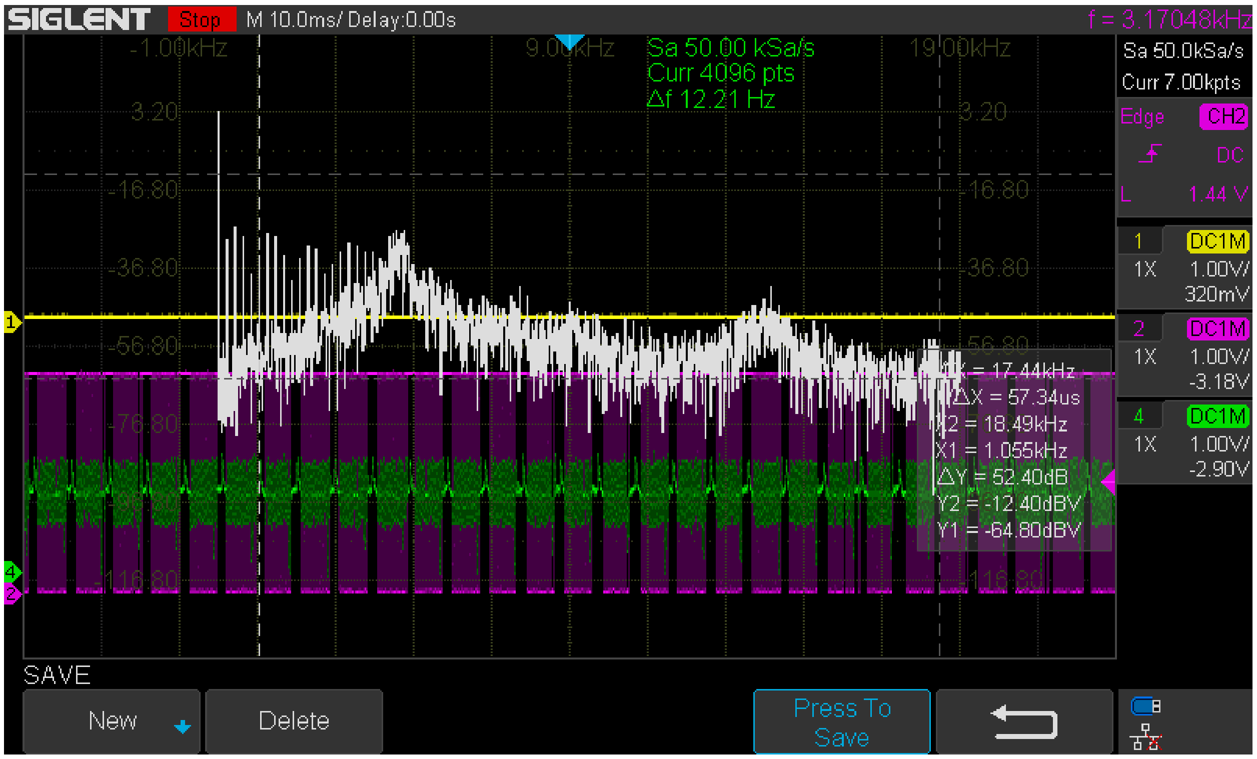
1 DC1M 1X 1.00V/ 320mV

2 DC1M 1X 1.00V/ -3.18V

4 DC1M 1X 1.00V/ -2.90V

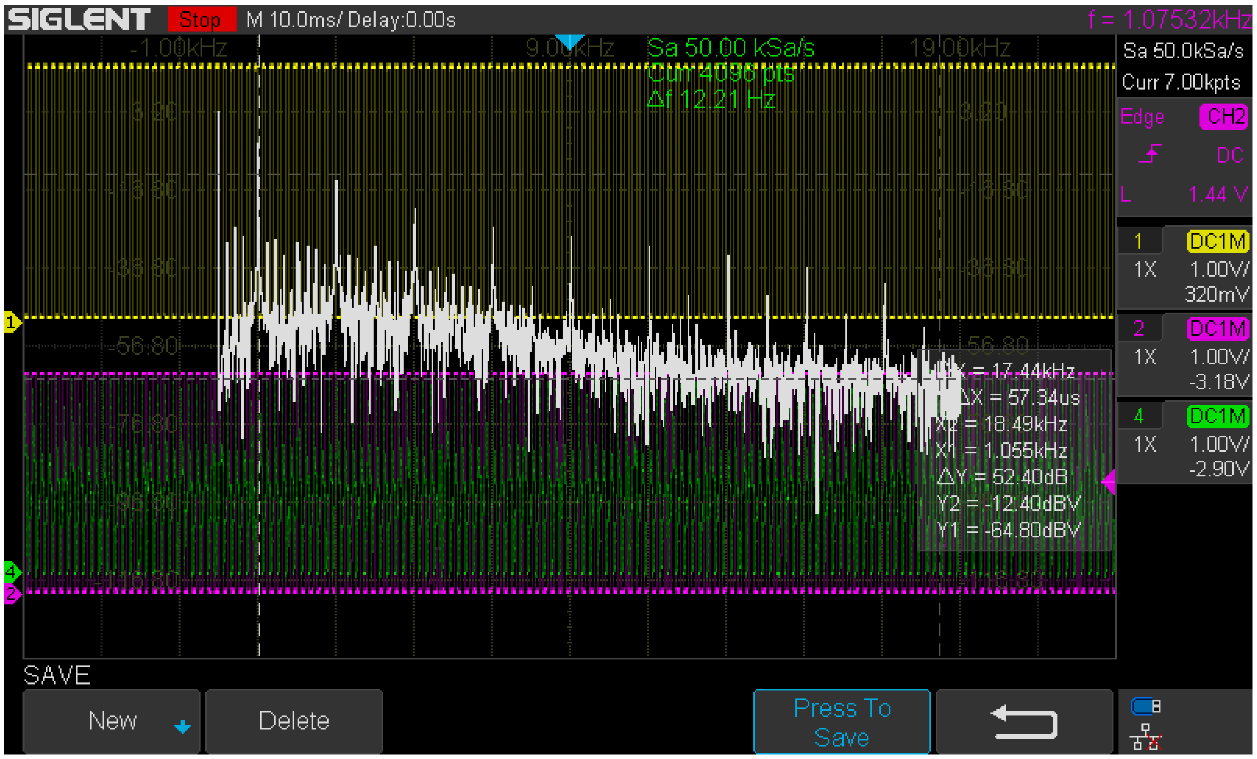
X = 17.44kHz
ΔX = 57.34us
X2 = 18.49kHz
X1 = 1.055kHz
ΔY = 52.40dB
Y2 = -12.40dBV
Y1 = -64.80dBV

No ambient light, no light source signal



M 10.0ms/ Delay:0.00s
f = 3.17048kHz
Sa 50.00 kSa/s
Curr 4096 pts
Δf 12.21 Hz
Sa 50.0kSa/s
Curr 7.00kpts
Edge CH2
DC
L 1.44 V
1 DC1M 1X 1.00V/ 320mV
2 DC1M 1X 1.00V/ -3.18V
4 DC1M 1X 1.00V/ -2.90V
ΔX = 17.44kHz
1/ΔX = 57.34us
X2 = 18.49kHz
X1 = 1.055kHz
ΔY = 52.40dB
Y2 = -12.40dBV
Y1 = -64.80dBV

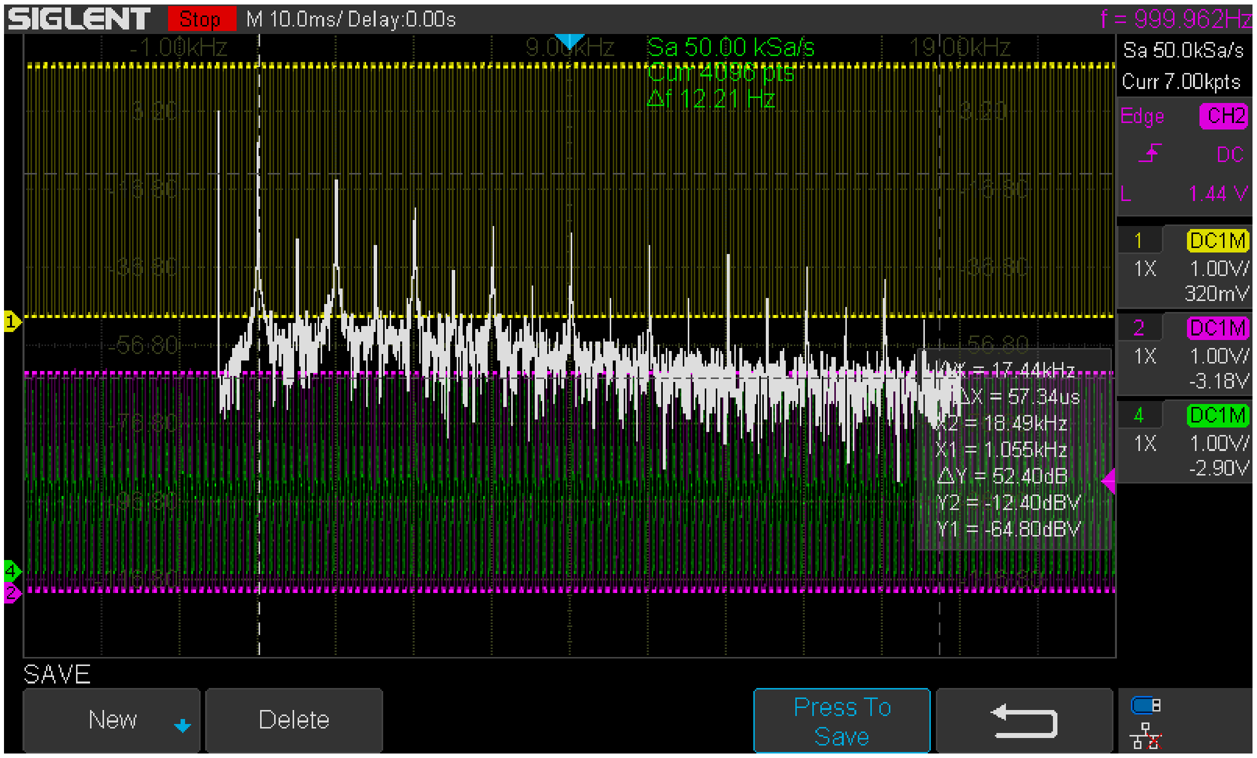
Ambient light, no light source signal



M 10.0ms/ Delay:0.00s
f = 1.07532kHz
Sa 50.00 kSa/s
Curr 4096 pts
Δf 12.21 Hz
Sa 50.0kSa/s
Curr 7.00kpts
Edge CH2
DC
L 1.44 V
1 DC1M 1X 1.00V/ 320mV
2 DC1M 1X 1.00V/ -3.18V
4 DC1M 1X 1.00V/ -2.90V
ΔX = 17.44kHz
1/ΔX = 57.34us
X2 = 18.49kHz
X1 = 1.055kHz
ΔY = 52.40dB
Y2 = -12.40dBV
Y1 = -64.80dBV

Ambient light, light source signal

Cursor on 1.055 kHz at -4.4 dBV



M 10.0ms/ Delay:0.00s

f = 999.962Hz

Sa 50.00 kSa/s
Curr 4096 pts
Δf 12.21 Hz

1/ΔX = 17.44kHz
ΔX = 57.34us
X2 = 18.49kHz
X1 = 1.055kHz
ΔY = 52.40dB
Y2 = -12.40dBV
Y1 = -64.80dBV

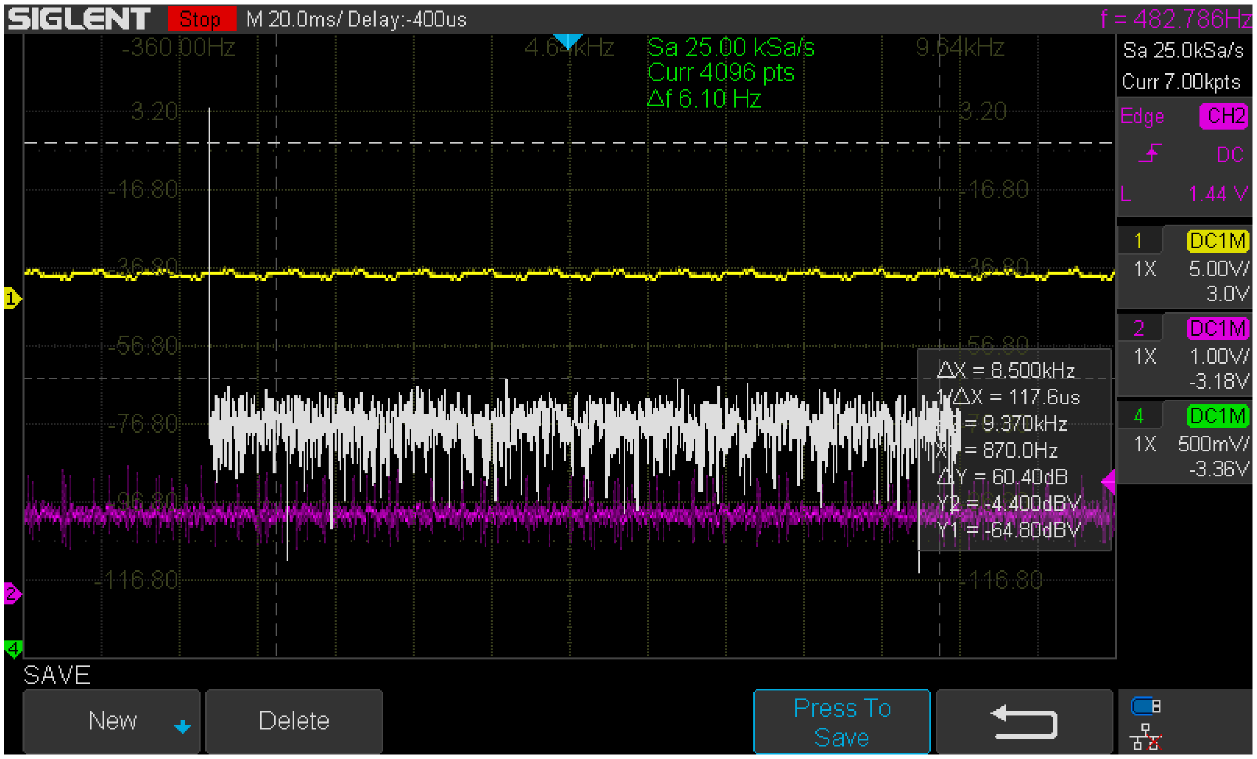
Sa 50.0kSa/s
Curr 7.00kpts
Edge CH2
DC
L 1.44 V

1 DC1M
1X 1.00V/
320mV

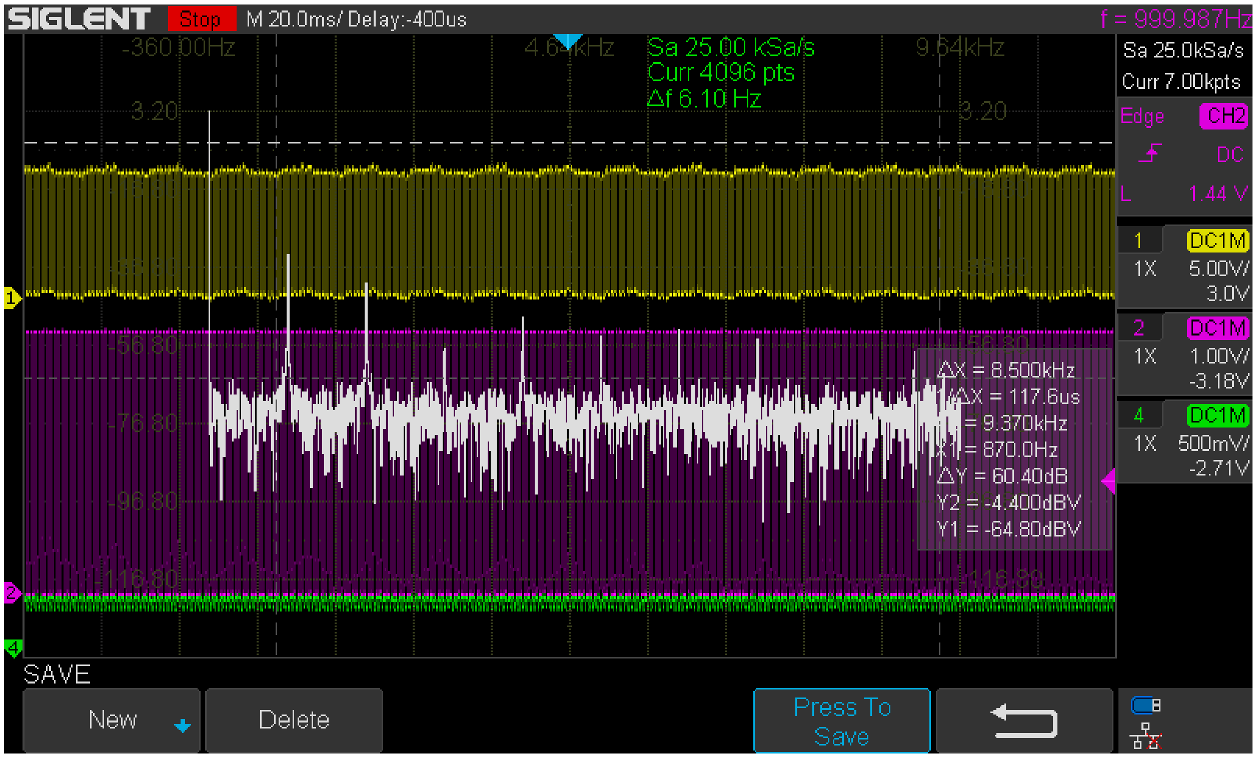
2 DC1M
1X 1.00V/
-3.18V

4 DC1M
1X 1.00V/
-2.90V

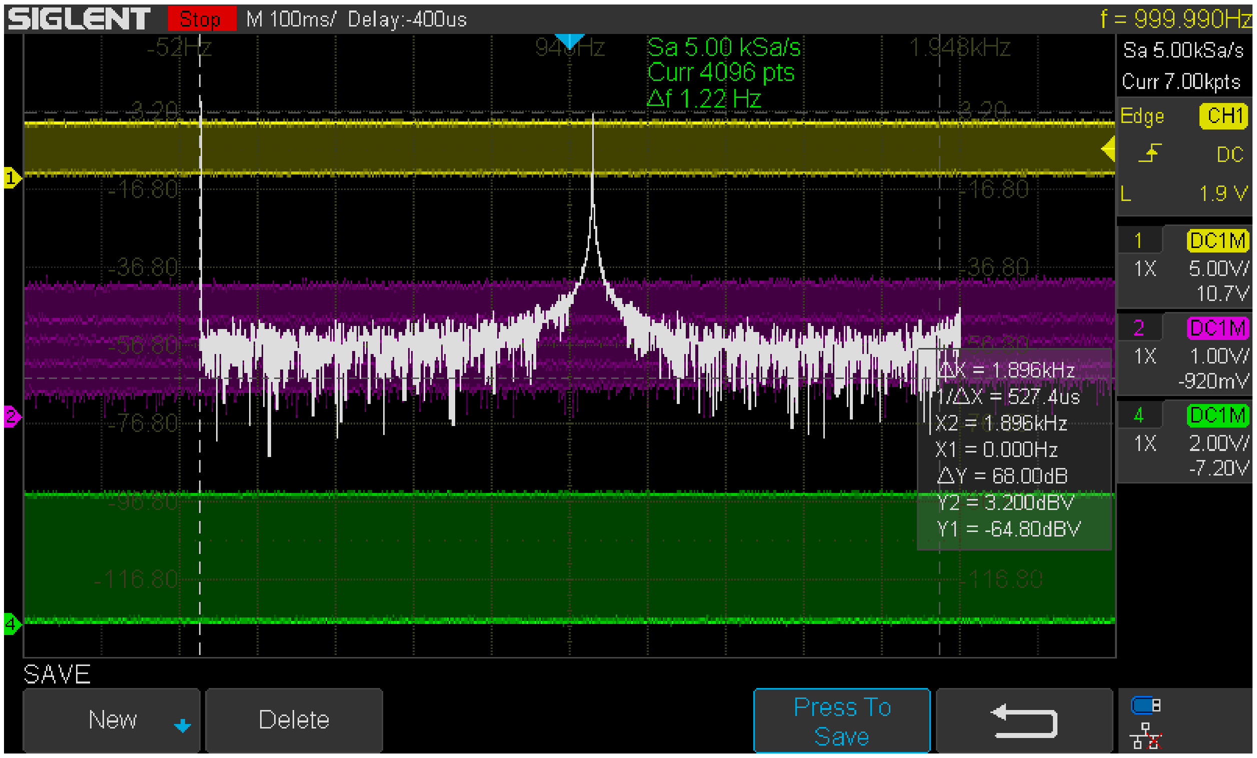
No ambient light, light source signal



Noise spectrum from just LED luminaire



Noise Spectrum from LED luminaire with 1kHz signal



```
M 100ms/  Delay:-400us                          f = 999.990Hz
Sa 5.00 kSa/s
Curr 4096 pts
Δf 1.22 Hz

ΔX = 1.896kHz
1/ΔX = 527.4us
X2 = 1.896kHz
X1 = 0.000Hz
ΔY = 68.00dB
Y2 = 3.200dBV
Y1 = -64.80dBV

Sa 5.00kSa/s
Curr 7.00kpts
Edge      CH1
          DC
L         1.9 V
1   DC1M
1X  5.00V/
    10.7V
2   DC1M
1X  1.00V/
    -920mV
4   DC1M
1X  2.00V/
    -7.20V
```

Noise spectrum with IR signal and LED luminaire ambient light

Manchester Communication Test

MATLAB script generate_manchester_waveform.m

```
% This script generates a voltage waveform to be loaded to the Koolertron
% waveform generator

clear all;

Tbit          = 200e-6;                    % bit duration in sec.
filename       = strcat('.\manchester_Tbit_', num2str(Tbit*1e6),'us.csv');

hi_level       = 3.0;
lo_level       = 0.0;

tx_data        = [65, 67, 69, 71, 73, 75, 77, 79];

one_waveform    = [0  0  0  0  1  1  1  1];   % normalized waveform between +1 and 0
(0=lo_level, 1=hi_level)
zero_waveform   = [1  1  1  1  0  0  0  0];   % normalized waveform between +1 and 0
(0=lo_level, 1=hi_level)
Lbit          = length(one_waveform);       % number of samples per bit
Ts            = Tbit/Lbit;                  % sampling time
Lsilence       = 64*3;
waveform       = zeros(1, length(tx_data)*(8*Lbit + Lsilence));

k  = 1;     % waveform sample index

for n=1:length(tx_data)
   byte = dec2bin(tx_data(n), 8);

   for m=8:-1:1
      bit = byte(m);
      if bit == '1'
         waveform( k : k+Lbit-1 ) = lo_level + (hi_level - lo_level)*one_waveform;
         k = k + Lbit;
      else
         waveform( k : k+Lbit-1 ) = lo_level + (hi_level - lo_level)*zero_waveform;
         k = k + Lbit;
      end
   end
   waveform(k:k+Lsilence-1) = hi_level*ones(1, Lsilence);  % between bytes silence
   k = k + Lsilence;

end

%------------------------------------------------------------------------
```

```
% write waveform to a file
%------------------------------------------------------------------------
fid = fopen(filename, 'w');
if fid ~= -1
  for n=1:length(waveform)
    fprintf(fid, '%2.5f\n', waveform(n));
  end
  fclose(fid);
else
  disp('Cannot open file...');
end

%------------------------------------------------------------------------
% print report
%------------------------------------------------------------------------
fprintf('---------------------------------------------\n');
fprintf(' Waveform Parameters:\n');
fprintf('   bit duration   : %3.4f ms\n', Tbit*1e3);
fprintf('   hi level       : %3.3f V\n', hi_level);
fprintf('   lo level       : %3.3f V\n', lo_level);

fprintf('   sampling freq. : %3.2f kHz\n', 1/Ts*1e-3);
fprintf('   num. samples   : %d\n', length(waveform));
fprintf('   one pulse      :');
fprintf(' %g ', one_waveform');
fprintf('\n');
fprintf('   zero pulse     :');
fprintf(' %g ', zero_waveform');
fprintf('\n');
```

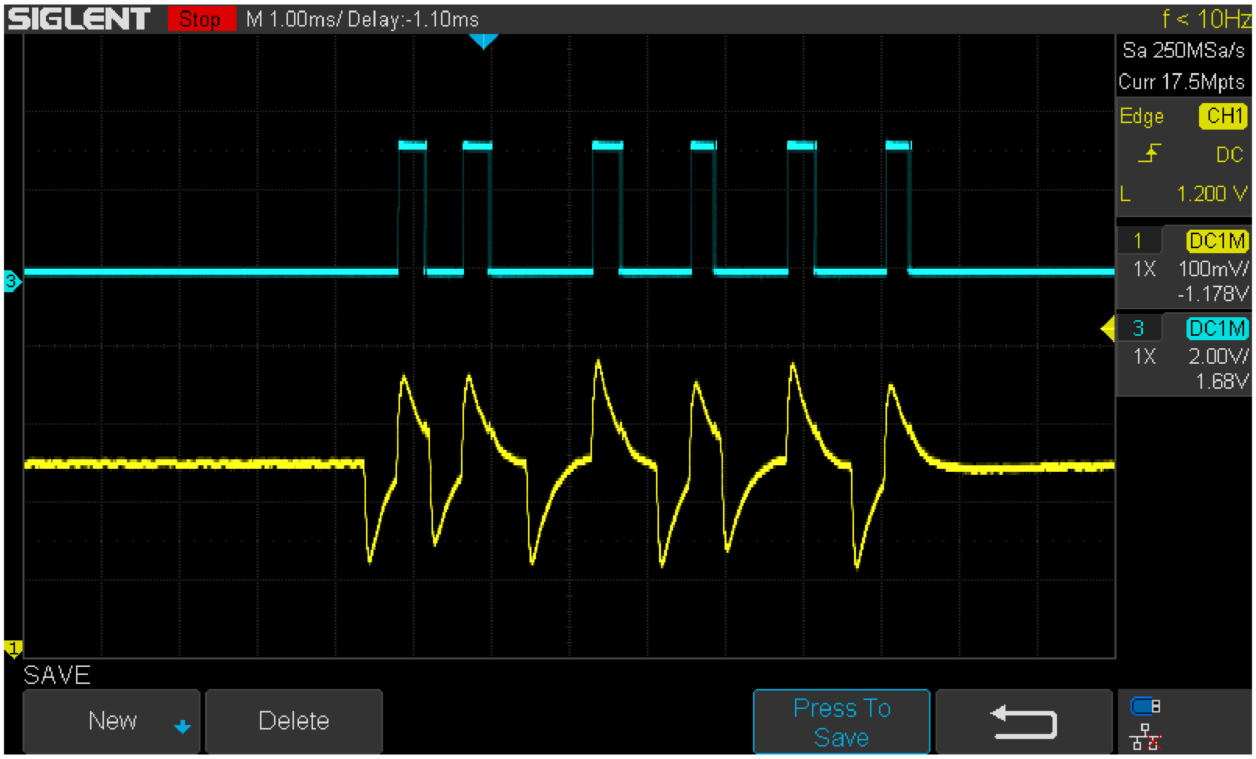
Command window output of generate_manchester_waveform.m

```
generate_manchester_waveform
---------------------------------------------
 Waveform Parameters:
   bit duration   : 0.2000 ms
   hi level       : 3.000 V
   lo level       : 0.000 V
   sampling freq. : 40.00 kHz
   num. samples   : 2048
   one pulse      : 0  0  0  0  1  1  1  1
   zero pulse     : 1  1  1  1  0  0  0  0
```

Manchester Decoding Results

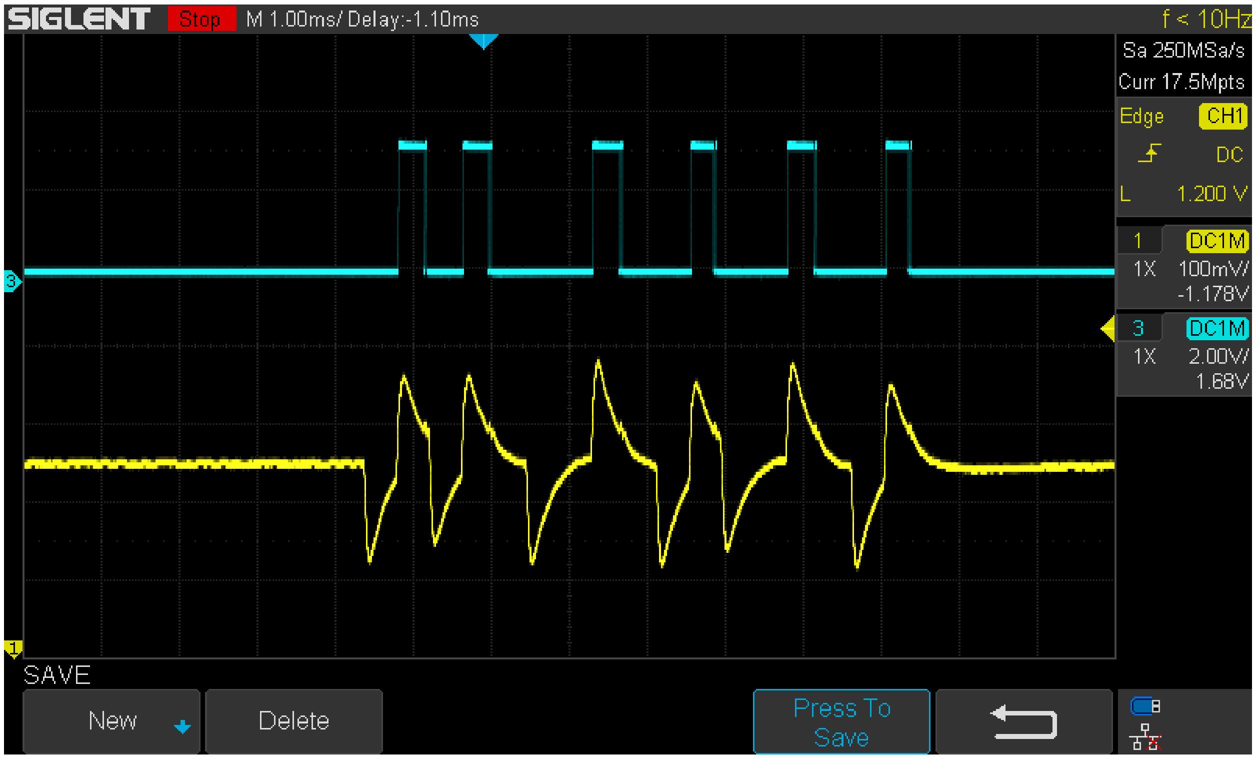
Bit rate: 2400 bps

	Input			Output			Bit Error Rate	
Packet	Bin	Dec	ASCII	Bin	Dec	ASCII	Transmission	Decode
1	11110010	79	O	1111000	15	SI	0/8	2/8
2	10000010	65	A	1000000	1	SOH	0/8	2/8
3	11000010	67	C	1100000	3	ETX	0/8	2/8
4	10100010	69	E	100000	1	SOH	0/8	3/8
5	11100010	71	G	1110000	7	BEL	0/8	2/8
6	10010010	73	I	100000	1	SOH	0/8	3/8
7	11010010	75	K	110000	3	ETX	0/8	3/8
8	10110010	77	M	100000	1	SOH	0/8	4/8
9	11110010	79	O	1111000	15	SI	0/8	2/8
10	10000010	65	A	1000000	1	SOH	0/8	2/8
11	11000010	67	C	1100000	3	ETX	0/8	2/8
12	10100010	69	E	100000	1	SOH	0/8	3/8
13	11100010	71	G	1110000	7	BEL	0/8	2/8
14	10010010	73	I	100000	1	SOH	0/8	2/8
15	11010010	75	K	110000	3	ETX	0/8	2/8
16	10110010	77	M	100000	1	SOH	0/8	4/8
17	11110010	79	O	1111000	15	SI	0/8	2/8
18	10000010	65	A	1000000	1	SOH	0/8	2/8
19	11000010	67	C	1100000	3	ETX	0/8	2/8
					Average		0%	31%


SIGLENT
Stop
M 1.00ms/ Delay:-1.10ms
f < 10Hz
Sa 250MSa/s
Curr 17.5Mpts
Edge CH1
DC
L 1.200 V
1 DC1M
1X 100mV/
-1.178V
3 DC1M
1X 2.00V/
1.68V
SAVE
New
Delete
Press To Save

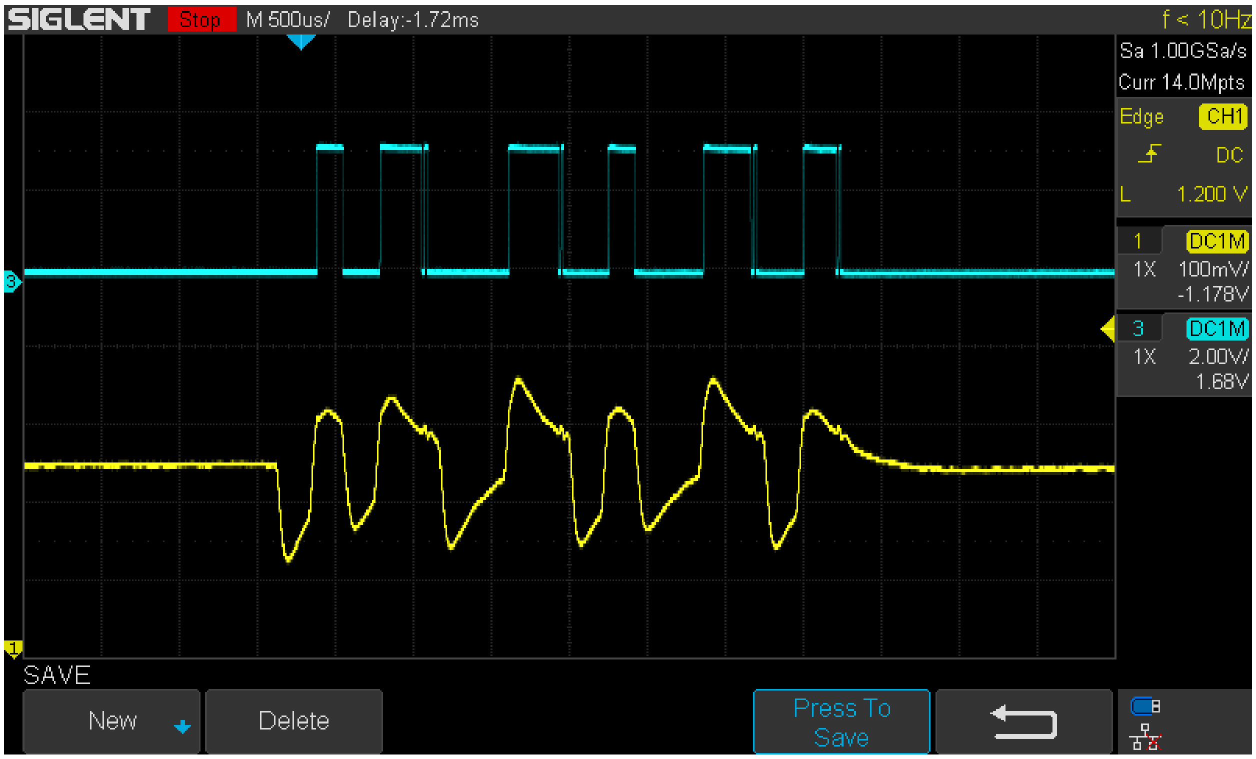
Bit rate: 3600 bps

	Input			Output			Bit Error Rate	
Packet	Bin	Dec	ASCII	Bin	Dec	ASCII	Transmission	Decode
1	10100010	69	E	1011100	29	GS	0/8	3/8
2	11100010	71	G	1111100	31	US	0/8	3/8
3	10010010	73	I	1101100	27	ESC	0/8	4/8
4	11010010	75	K	1101100	27	ESC	0/8	2/8
5	10110010	77	M	100000	1	SOH	0/8	4/8
6	11110010	79	O	1111000	15	SI	0/8	1/8
7	10000010	65	A	1000000	1	SOH	0/8	2/8
8	11000010	67	C	1111100	31	US	0/8	4/8
9	10100010	69	E	1011100	29	GS	0/8	4/8
10	11100010	71	G	1111100	31	US	0/8	4/8
11	10010010	73	I	110000	3	ETX	0/8	4/8
					Average		0%	40%



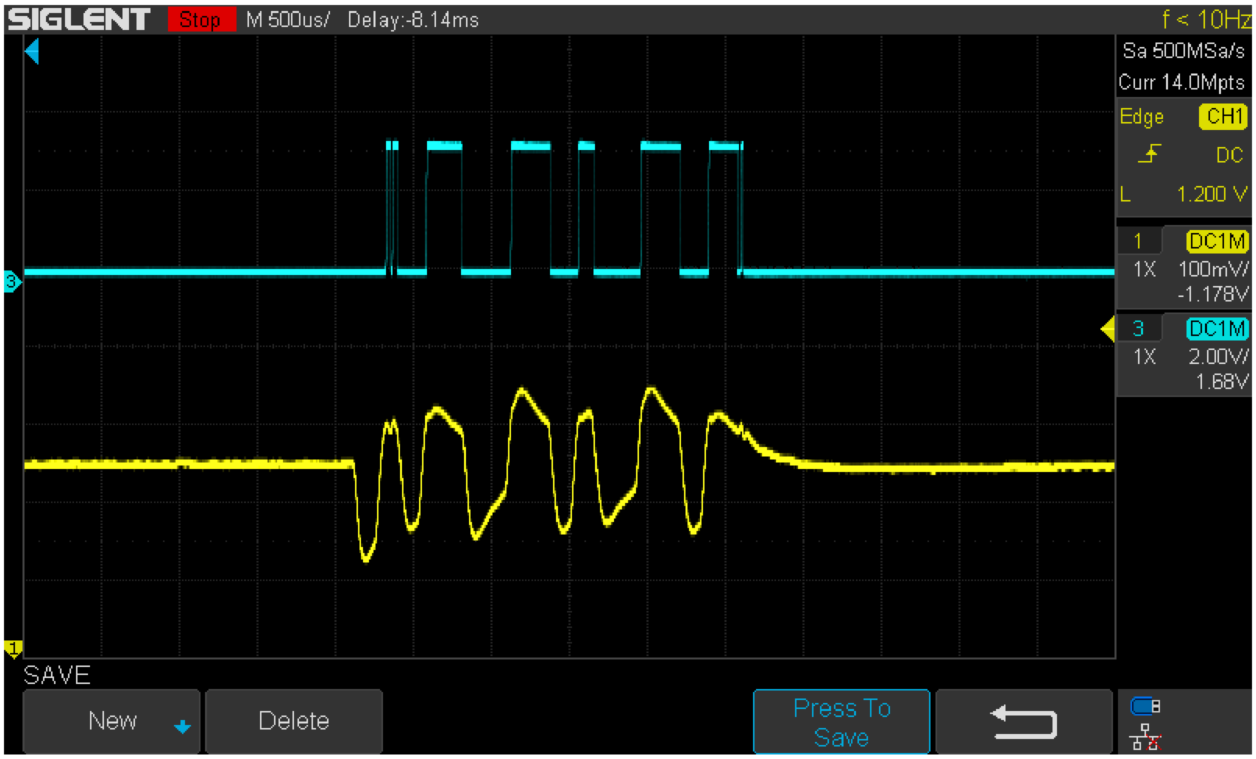
Bit rate: 4800 bps

	Input			Output			Bit Rate Error	
Packet	Bin	Dec	ASCII	Bin	Dec	ASCII	Transmission	Decode
1	10010010	73	I	10010010	73	I	0/8	0/8
2	11010010	75	K	11010010 1	75 1	K SOH	0/8	1/9
3	10110010	77	M	10010010	73	I	0/8	1/8
4	11110010	79	O	11110010	79	O	0/8	0/8
5	10000010	65	A	10000010	65	A	0/8	0/8
6	11000010	67	C	11000010	67	C	0/8	0/8
7	10100010	69	E	10100010 1	69 1	E SOH	0/8	1/9
8	11100010	71	G	11100010	71	G	0/8	0/8
9	10010010	73	I	10010010	73	I	0/8	0/8
10	11010010	75	K	11010010	75	K	0/8	0/8
11	10110010	77	M	10010010	73	I	0/8	1/8
12	11110010	79	O	11110010 1	79 1	O SOH	0/8	1/9
13	10000010	65	A	10000010	65	A	0/8	0/8
14	11000010	67	C	11000010	67	C	0/8	0/8
					Average		0%	0%



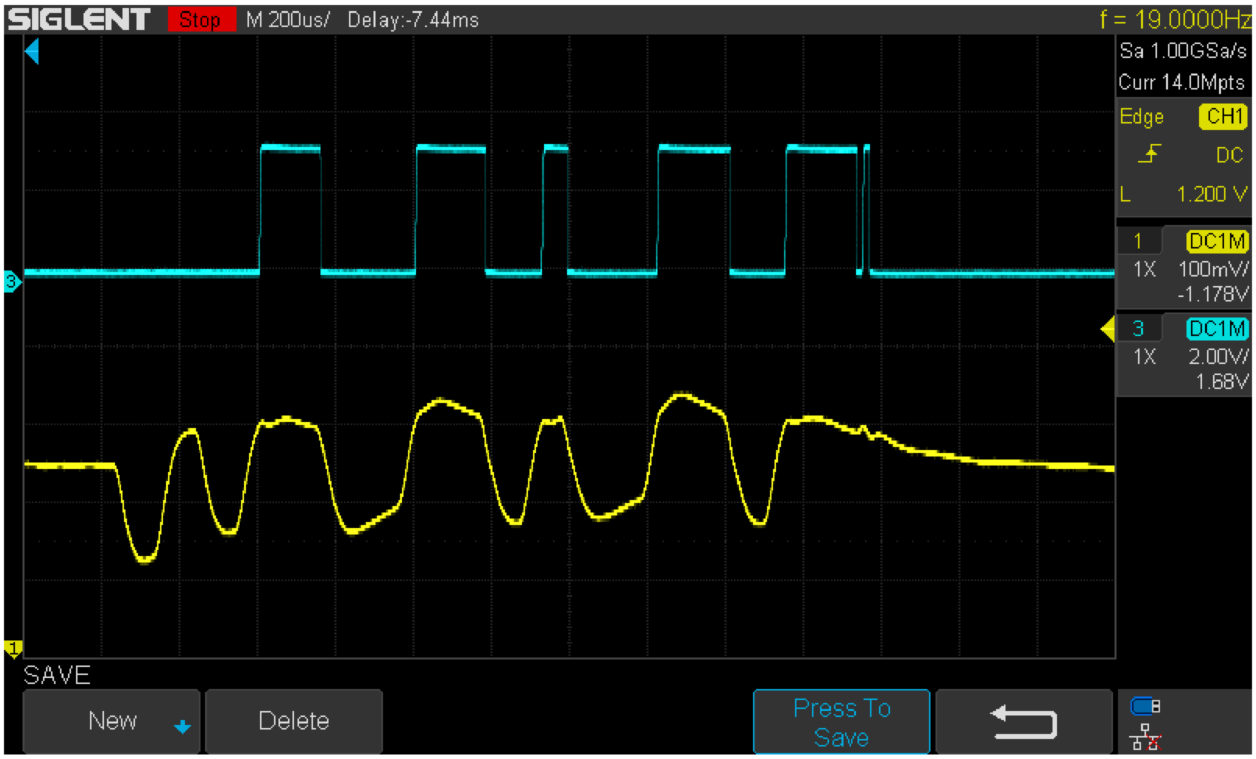
Bit rate: 7200 bps

	Input			Output			Bit Rate Error	
Packet	Bin	Dec	ASCII	Bin	Dec	ASCII	Transmission	Decode
1	11010010	75	K	11011011	219	ï¿½	0/8	2/8
2	10110010	77	M	10011011	217	ï¿½	0/8	3/8
3	11110010	79	O	11111011	223	ï¿½	0/8	2/8
4	10000010	65	A	11111011	223	ï¿½	0/8	5/8
5	11000010	67	C	11111011	223	ï¿½	0/8	4/8
6	10100010	69	E	10111011	221	ï¿½	0/8	3/8
7	11100010	71	G	11111011	223	ï¿½	0/8	3/8
8	10010010	73	I	11011011	219	ï¿½	0/8	3/8
9	11010010	75	K	11011011	219	ï¿½	0/8	2/8
10	10110010	77	M	10011011	217	ï¿½	0/8	3/8
11	11110010	79	O	11111011	223	ï¿½	0/8	2/8
					Average		0%	36%



Bit rate: 9600 bps

	Input			Output			Bit Rate Error	
Packet	Bin	Dec	ASCII	Bin	Dec	ASCII	Transmission	Decode
2	10010010	73	I	1111011	222	ï¿½	0/8	5/8
3	11010010	75	K	10111011	221	ï¿½	0/8	3/8
4	10110010	77	M	100011	196	ï¿½	0/8	3/8
5	11110010	79	O	11011010 1	91 1	[SOH	0/8	3/9
6	10000010	65	A	1011011	218	ï¿½	0/8	3/8
7	11000010	67	C	10011011	217	ï¿½	0/8	3/8
8	10100010	69	E	10011	200	ï¿½	0/8	4/8
9	11100010	71	G	11111011	223	ï¿½	0/8	5/8
10	10010010	73	I	1111011	222	ï¿½	0/8	5/8
11	11010010	75	K	10111011	221	ï¿½	0/8	3/8
12	10110010	77	M	100011	196	ï¿½	0/8	3/8
13	11110010	79	O	11011011	219	ï¿½	0/8	3/8
14	10000010	65	A	1011011	218	ï¿½	0/8	3/8
15	11000010	67	C	10011011	217	ï¿½	0/8	3/8
					Average		0%	43%


SIGLENT Stop M 200us/ Delay:-7.44ms f = 19.0000Hz

Sa 1.00GSa/s
Curr 14.0Mpts
Edge CH1
DC
L 1.200 V
1 DC1M 1X 100mV/ -1.178V
3 DC1M 1X 2.00V/ 1.68V

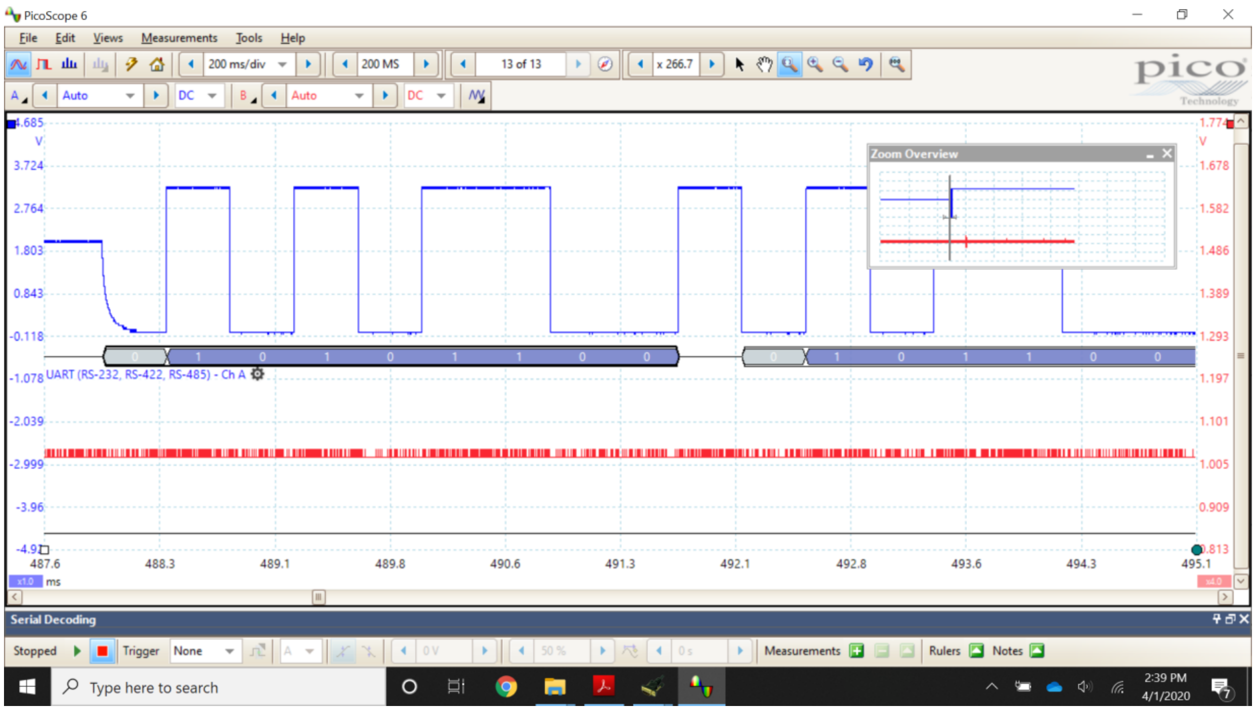
Serial Decoding Results

		Input				Output				
Bit Rate	Packet	Bin	Hex	Dec	ASCII	Bin	Hex	Dec	ASCII	BER
2400	1	10101100	35	53	5	10101100	35	53	5	0%
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
3600	1	10101100	35	53	5	10101100	35	53	5	0%
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
4800	1	10101100	35	53	5	10101100	35	53	5	0%
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
7200	1	10101100	35	53	5	10101100	35	53	5	0%
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					
9600	1	10101100	35	53	5	10101100	35	53	5	0%
	2	10110000	0D	13	CR					
	3	1010000	0A	10	LF					

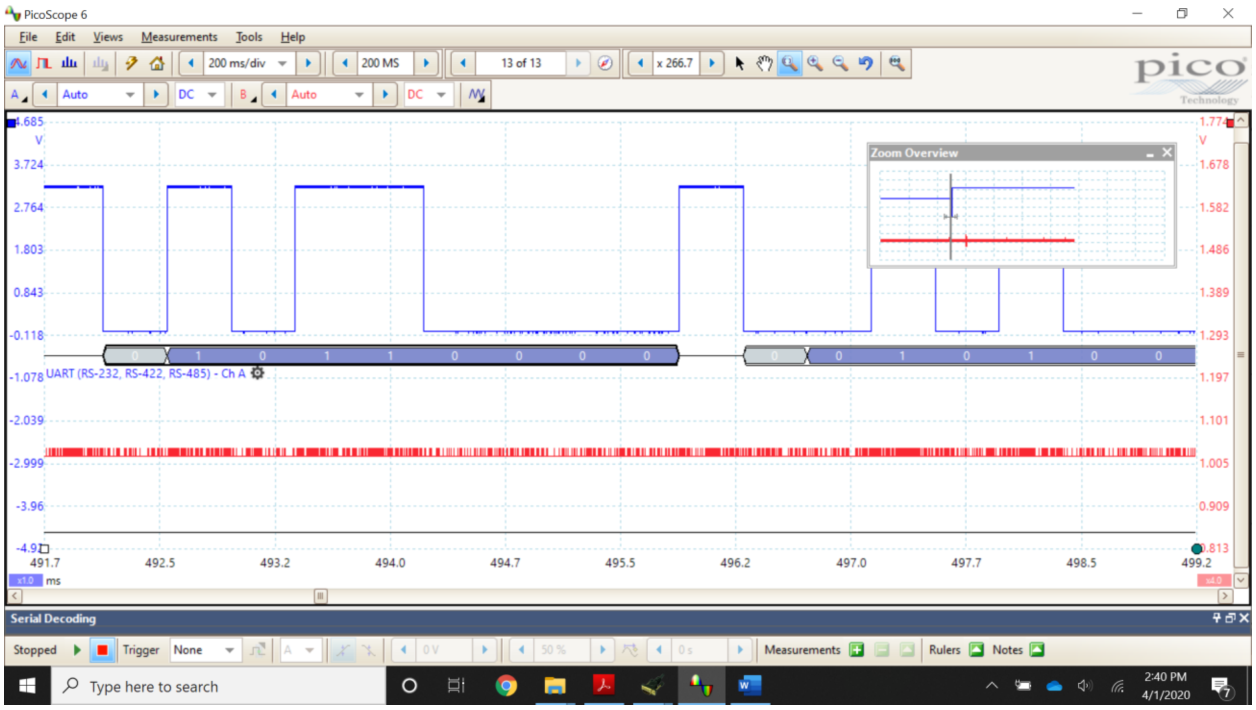
Serial Transmission at 2400 bps

Delay between packet send and receive: 110.9 ms

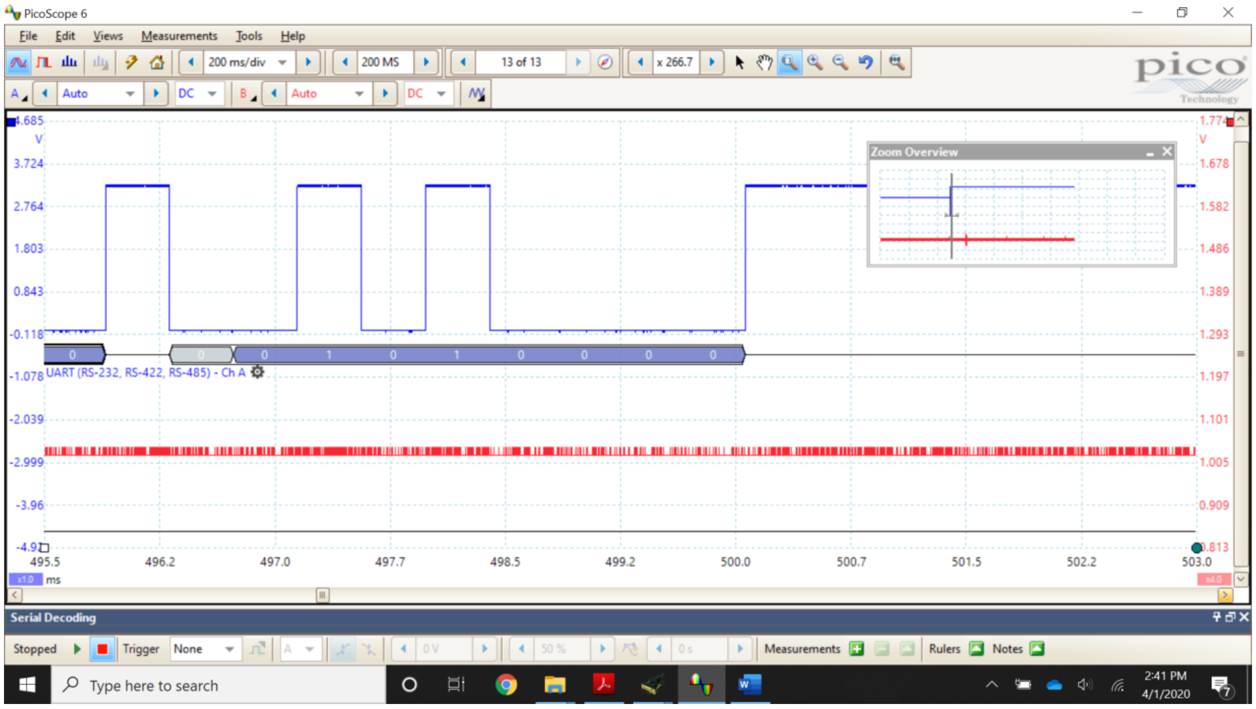
Packet 1


PicoScope 6
File Edit Views Measurements Tools Help
200 ms/div
200 MS
13 of 13
x 266.7
Auto
DC
Auto
DC
Zoom Overview
UART (RS-232, RS-422, RS-485) - Ch A
1 0 1 0 1 1 0 0
0 1 0 1 1 0 0
4.685 V 3.724 2.764 1.803 0.843 -0.118 -1.078 -2.039 -2.999 -3.96 -4.92
1.774 V 1.678 1.582 1.486 1.389 1.293 1.197 1.101 1.005 0.909 0.813
487.6 488.3 489.1 489.8 490.6 491.3 492.1 492.8 493.6 494.3 495.1 ms
Serial Decoding
Stopped Trigger None Measurements Rulers Notes
Type here to search
2:39 PM 4/1/2020

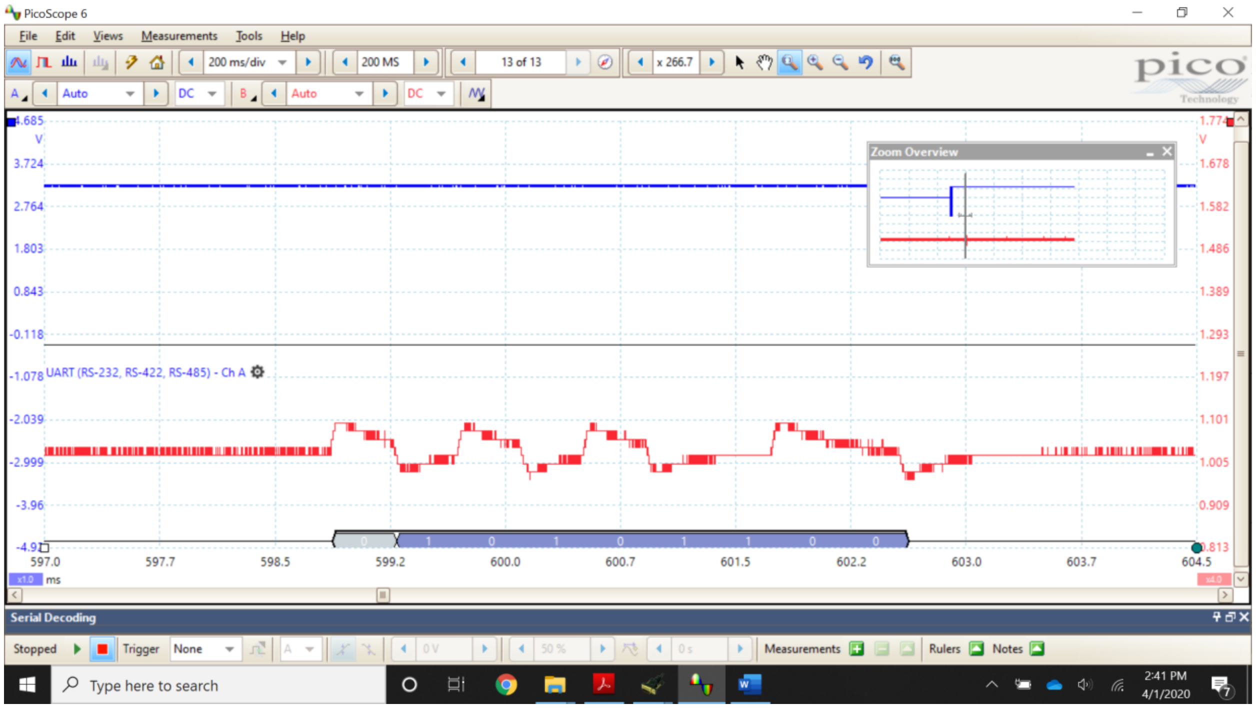
Packet 2


PicoScope 6
File Edit Views Measurements Tools Help
200 ms/div
200 MS
13 of 13
x 266.7
Auto
DC
Auto
DC
Zoom Overview
UART (RS-232, RS-422, RS-485) - Ch A
1 0 1 1 0 0 0 0
0 1 0 1 0 0
4.685 V 3.724 2.764 1.803 0.843 -0.118 -1.078 -2.039 -2.999 -3.96 -4.92
1.774 V 1.678 1.582 1.486 1.389 1.293 1.197 1.101 1.005 0.909 0.813
491.7 492.5 493.2 494.0 494.7 495.5 496.2 497.0 497.7 498.5 499.2 ms
Serial Decoding
Stopped Trigger None Measurements Rulers Notes
Type here to search
2:40 PM 4/1/2020

Packet 3



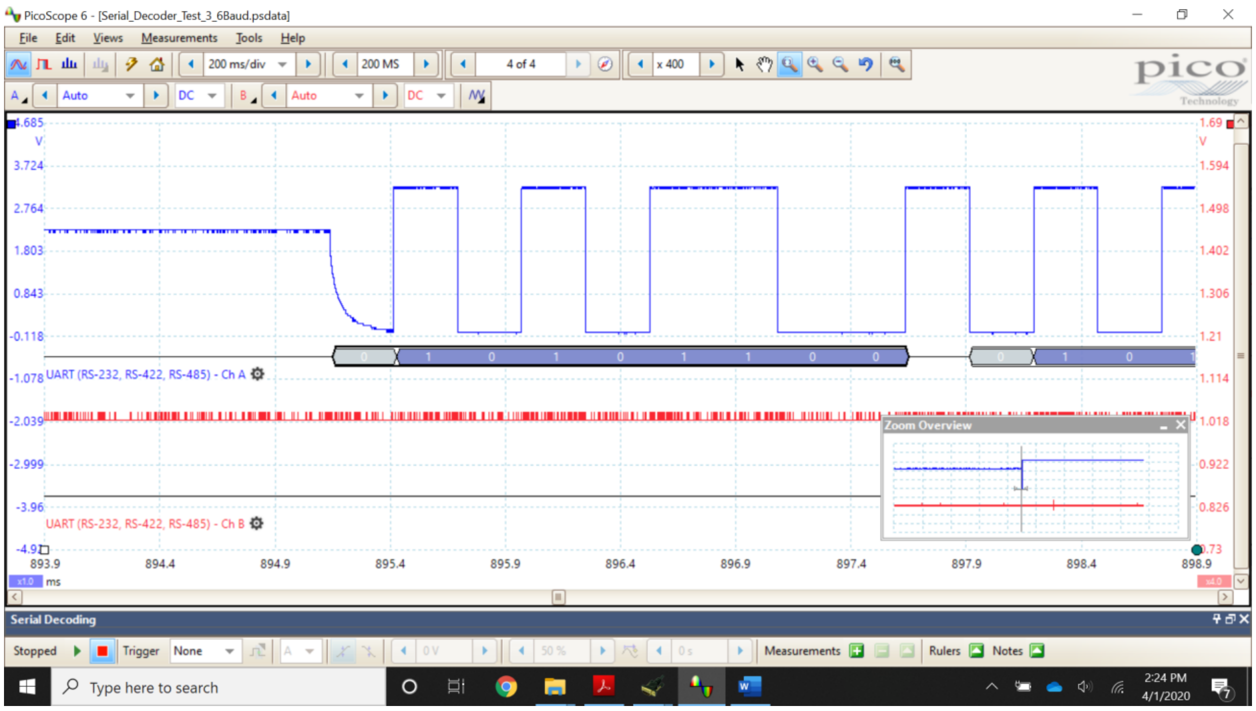
Packet 1 (received)



Serial Transmission at 3600

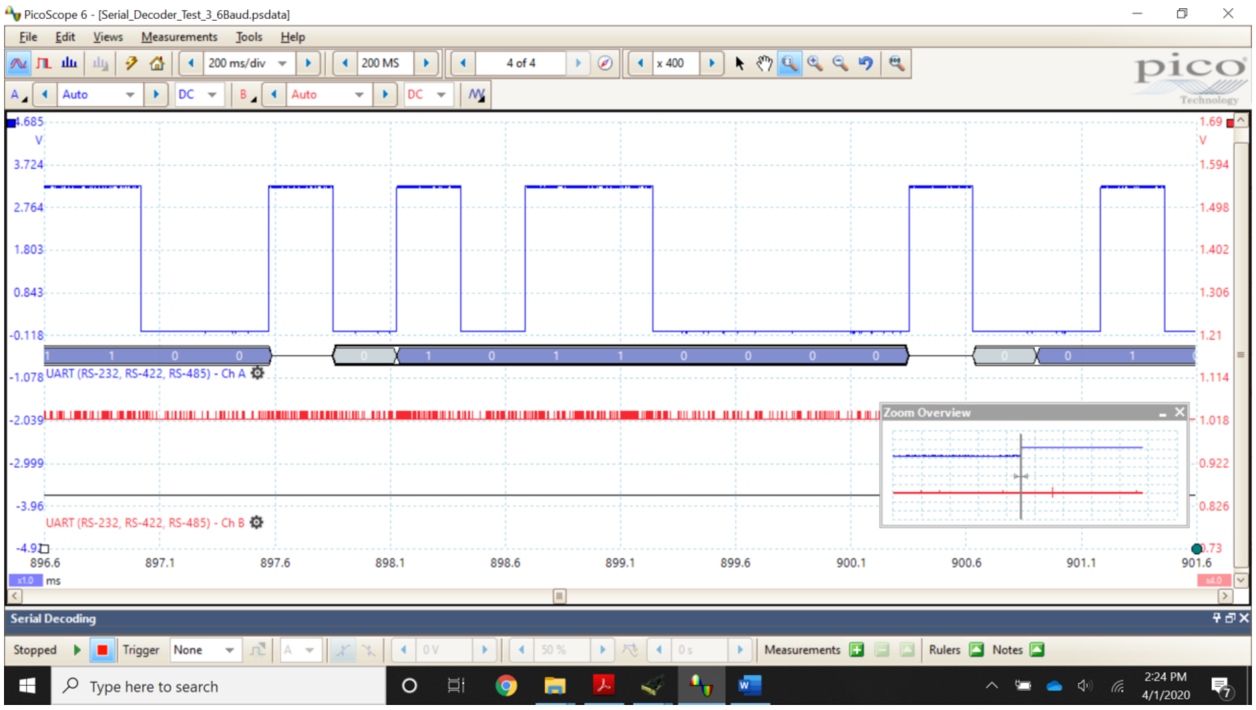
Time delay between send and receive packets: 223.1 ms

Packet 1



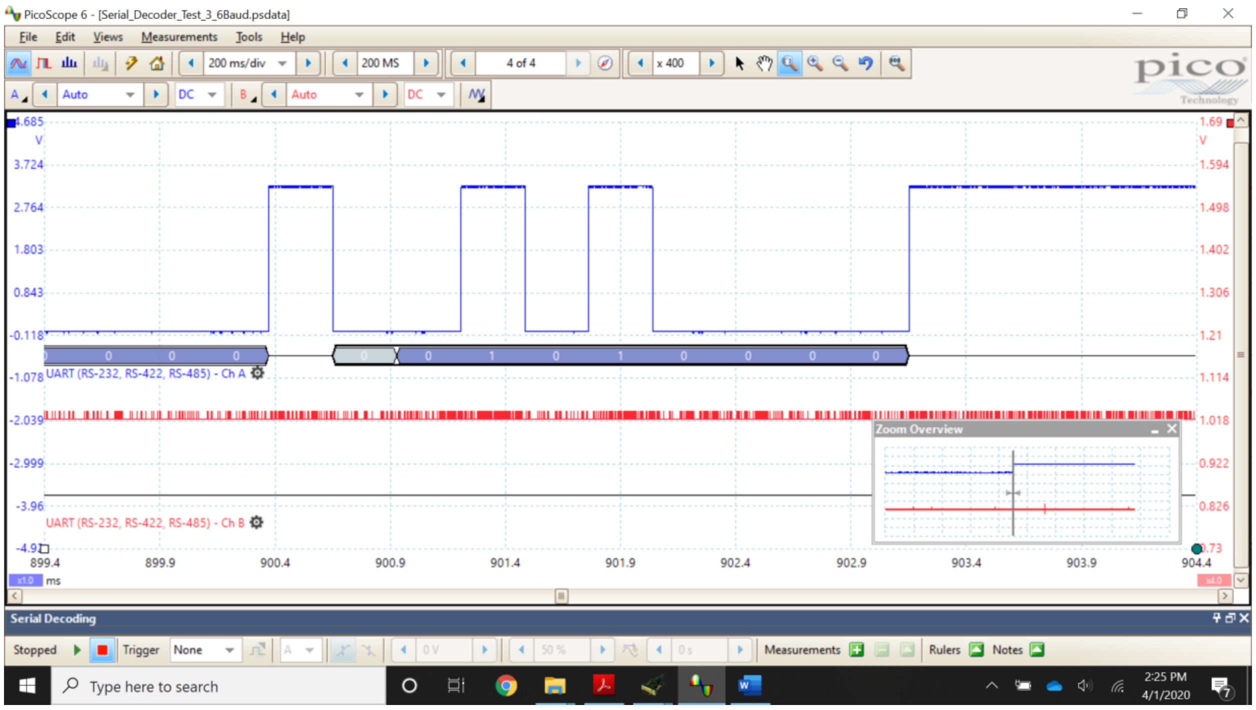
PicoScope 6 - [Serial_Decoder_Test_3_6Baud.psdata]
File Edit Views Measurements Tools Help
200 ms/div
200 MS
4 of 4
x 400
Auto DC
UART (RS-232, RS-422, RS-485) - Ch A
UART (RS-232, RS-422, RS-485) - Ch B
Zoom Overview
893.9 894.4 894.9 895.4 895.9 896.4 896.9 897.4 897.9 898.4 898.9
ms
Serial Decoding
Stopped Trigger None
Measurements Rulers Notes

Packet 2

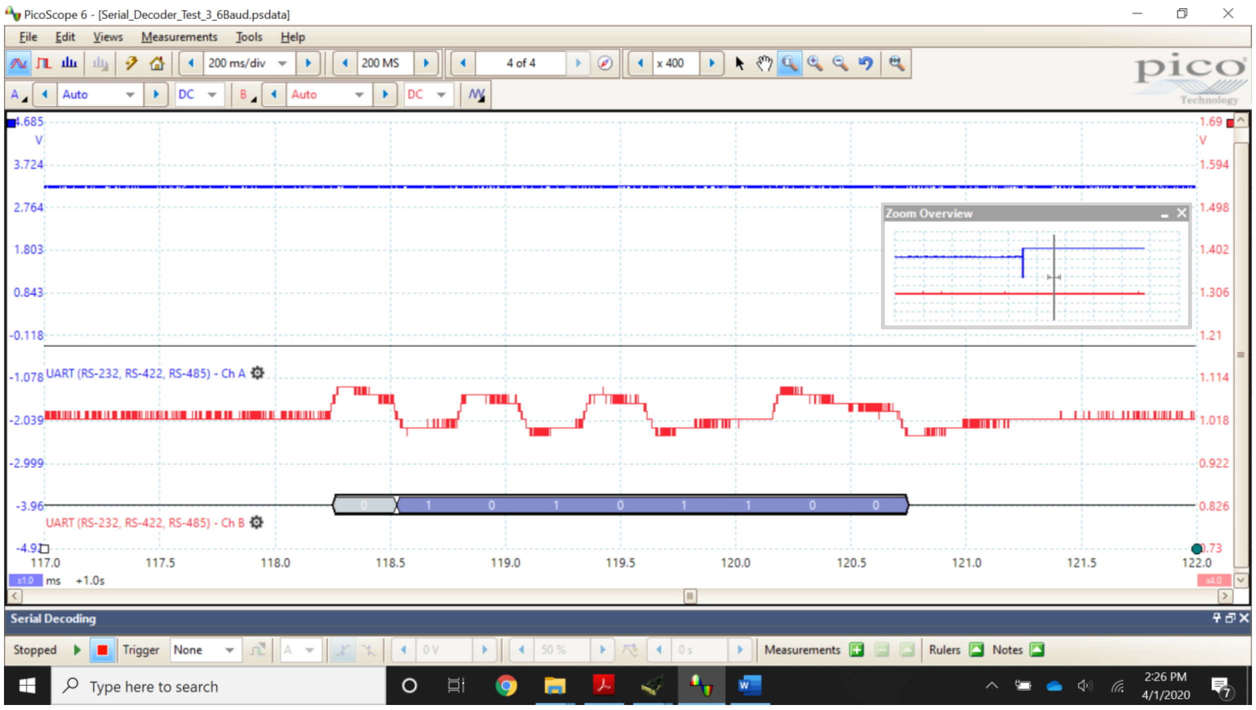


PicoScope 6 - [Serial_Decoder_Test_3_6Baud.psdata]
File Edit Views Measurements Tools Help
200 ms/div
200 MS
4 of 4
x 400
Auto DC
UART (RS-232, RS-422, RS-485) - Ch A
UART (RS-232, RS-422, RS-485) - Ch B
Zoom Overview
896.6 897.1 897.6 898.1 898.6 899.1 899.6 900.1 900.6 901.1 901.6
ms
Serial Decoding
Stopped Trigger None
Measurements Rulers Notes

Packet 3



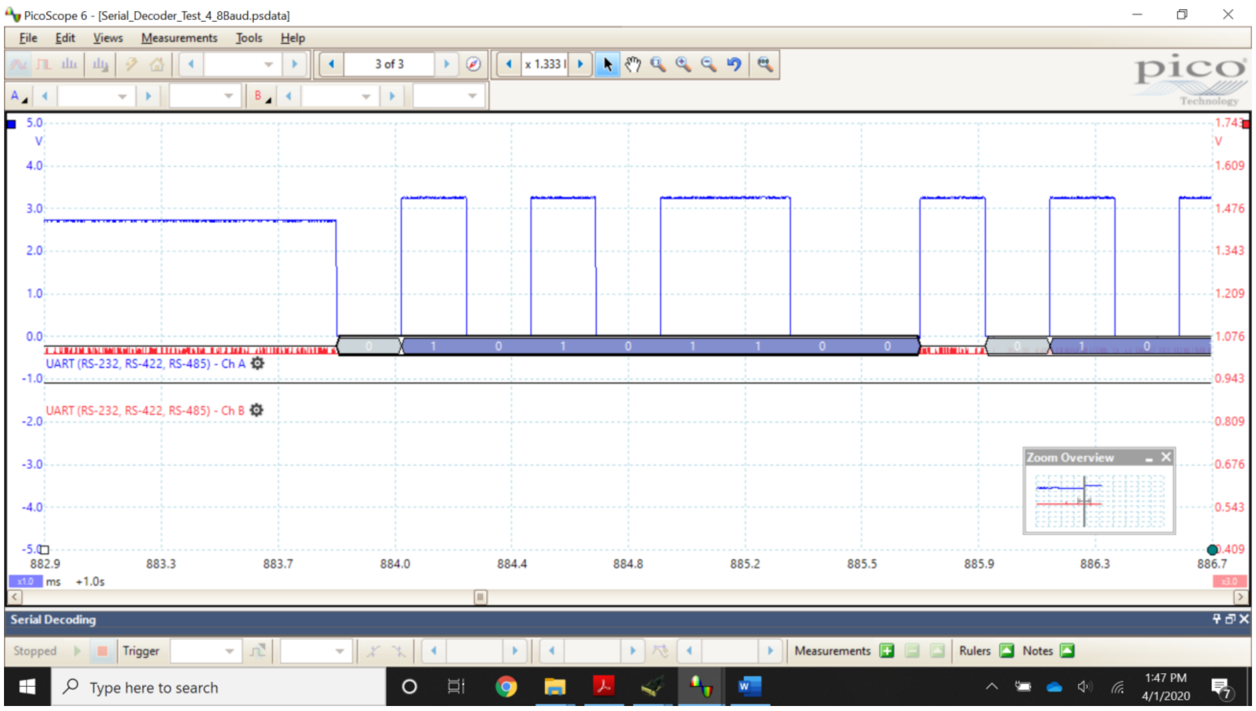
Packet 1 (received)



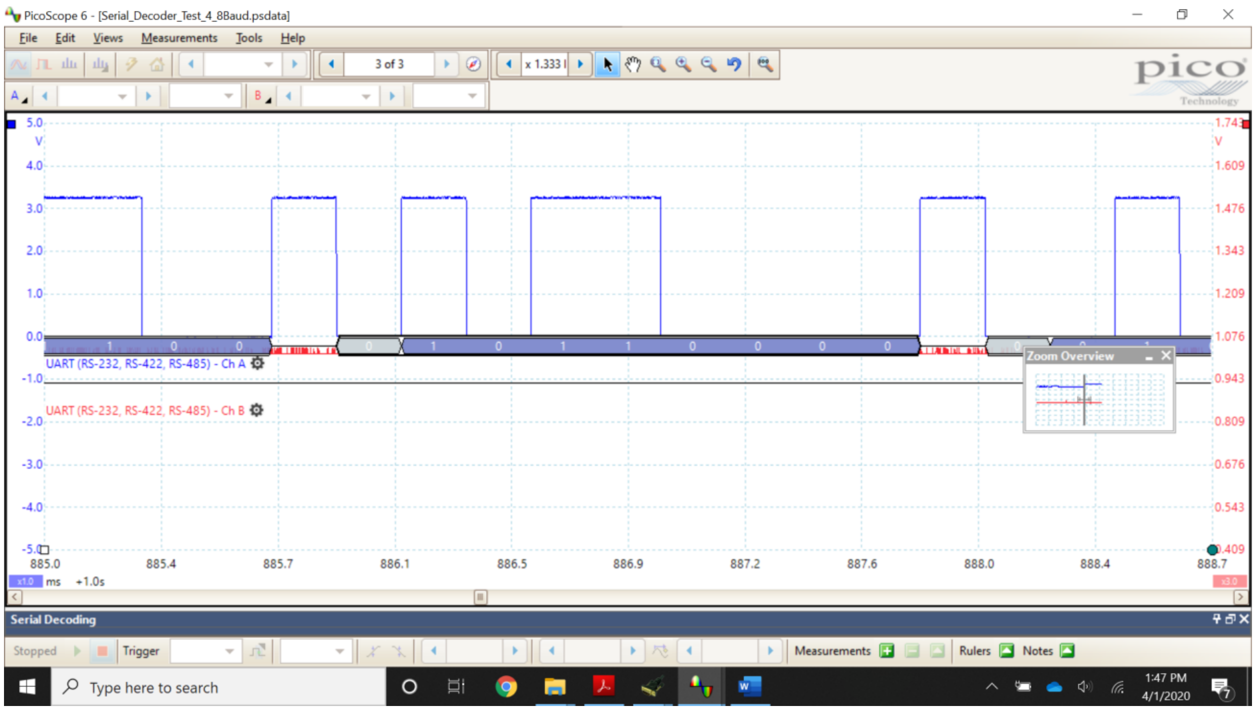
Serial Transmission at 4800

Time delay between 1 packet sent and received

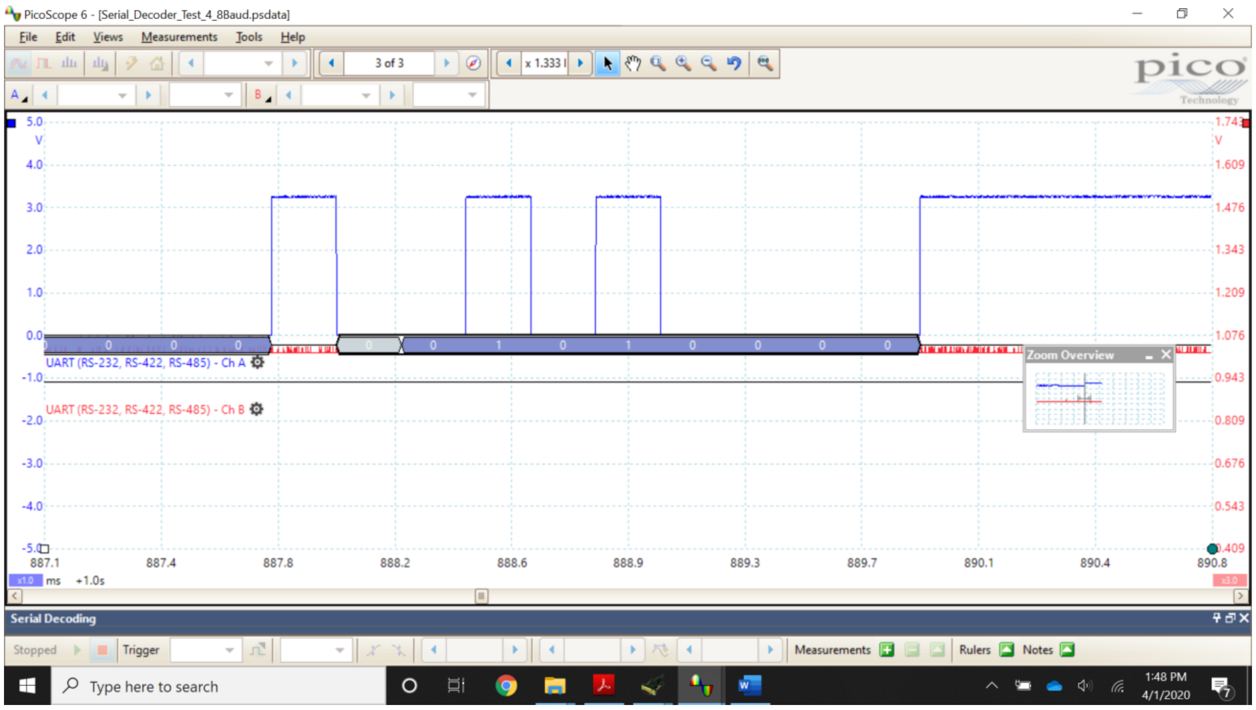
Packet 1


PicoScope 6 - [Serial_Decoder_Test_4_8Baud.psdata]
File Edit Views Measurements Tools Help
3 of 3
x 1.333
UART (RS-232, RS-422, RS-485) - Ch A
UART (RS-232, RS-422, RS-485) - Ch B
Zoom Overview
882.9 883.3 883.7 884.0 884.4 884.8 885.2 885.5 885.9 886.3 886.7
ms +1.0s
Serial Decoding
Stopped Trigger Measurements Rulers Notes
Type here to search
1:47 PM 4/1/2020

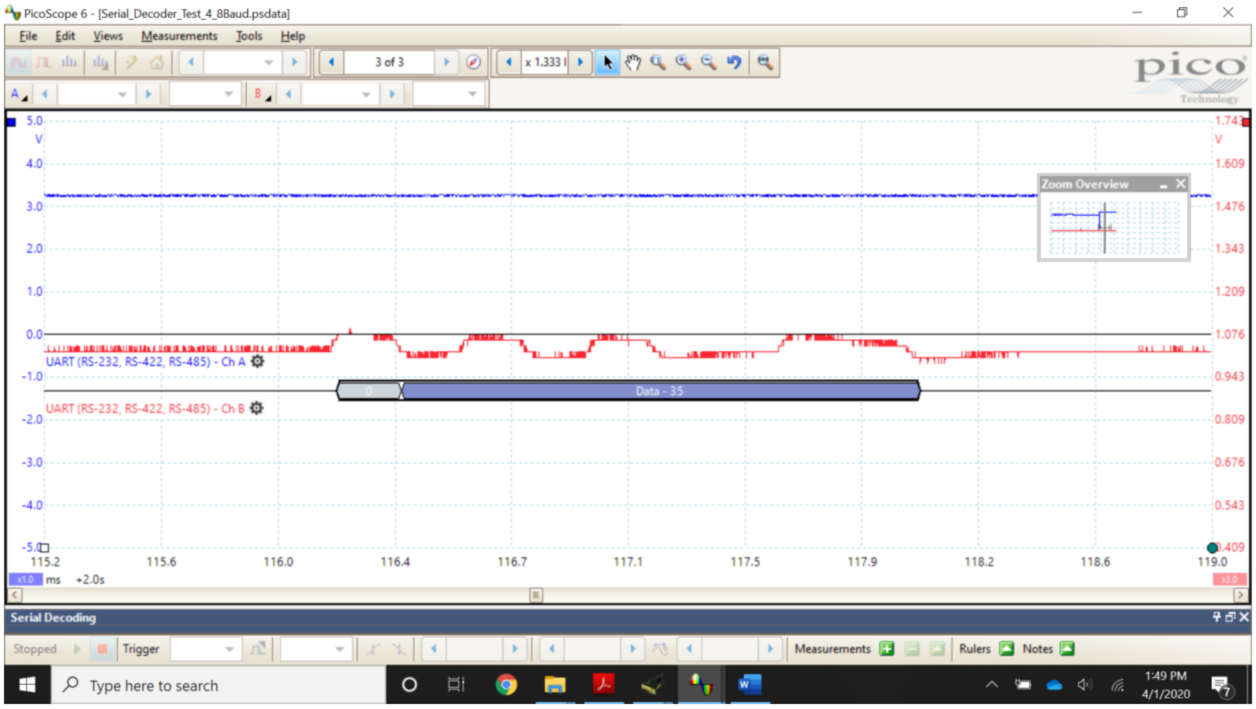
Packet 2


PicoScope 6 - [Serial_Decoder_Test_4_8Baud.psdata]
File Edit Views Measurements Tools Help
3 of 3
x 1.333
UART (RS-232, RS-422, RS-485) - Ch A
UART (RS-232, RS-422, RS-485) - Ch B
Zoom Overview
885.0 885.4 885.7 886.1 886.5 886.9 887.2 887.6 888.0 888.4 888.7
ms +1.0s
Serial Decoding
Stopped Trigger Measurements Rulers Notes
Type here to search
1:47 PM 4/1/2020

Packet 3


PicoScope 6 - [Serial_Decoder_Test_4_8Baud.psdata]
File Edit Views Measurements Tools Help
3 of 3
x 1.333
UART (RS-232, RS-422, RS-485) - Ch A
UART (RS-232, RS-422, RS-485) - Ch B
0 0 0 0 1 0 1 0 0 0 0
Zoom Overview
887.1 887.4 887.8 888.2 888.6 888.9 889.3 889.7 890.1 890.4 890.8
ms +1.0s
Serial Decoding
Stopped Trigger Measurements Rulers Notes
1:48 PM 4/1/2020

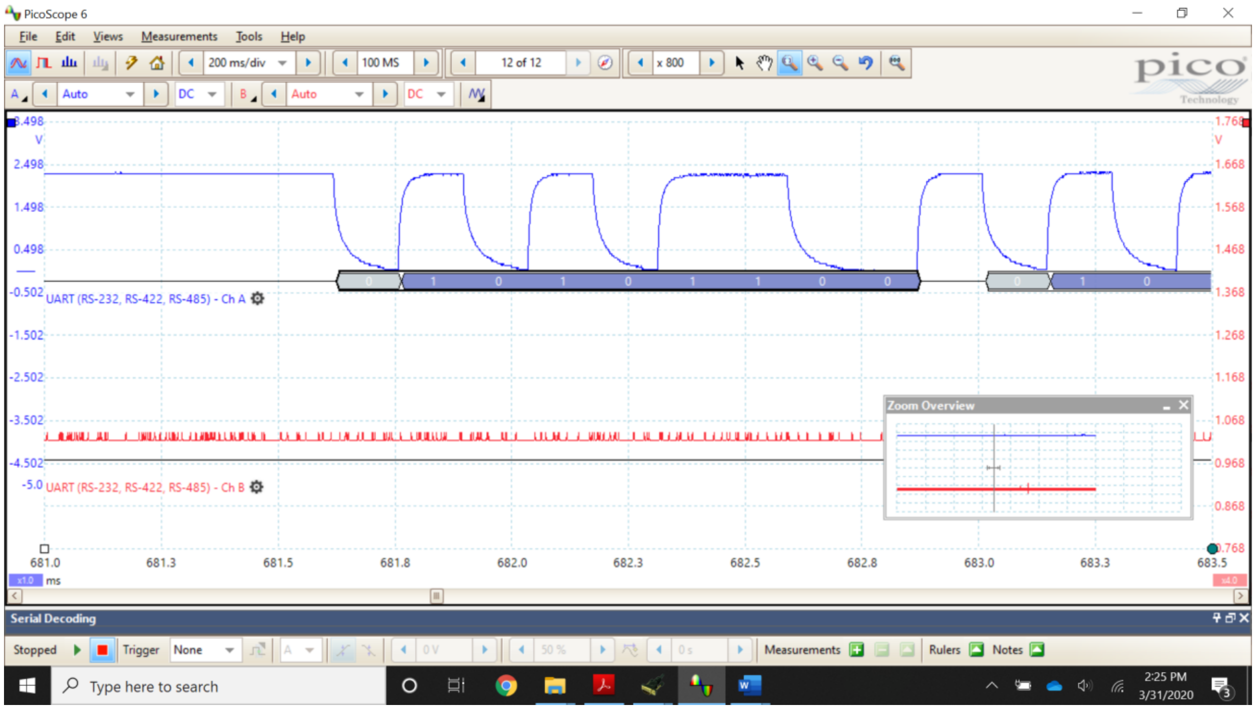
Packet 1 (received)


PicoScope 6 - [Serial_Decoder_Test_4_8Baud.psdata]
File Edit Views Measurements Tools Help
3 of 3
x 1.333
UART (RS-232, RS-422, RS-485) - Ch A
UART (RS-232, RS-422, RS-485) - Ch B
0
Data - 35
Zoom Overview
115.2 115.6 116.0 116.4 116.7 117.1 117.5 117.9 118.2 118.6 119.0
ms +2.0s
Serial Decoding
Stopped Trigger Measurements Rulers Notes
1:49 PM 4/1/2020

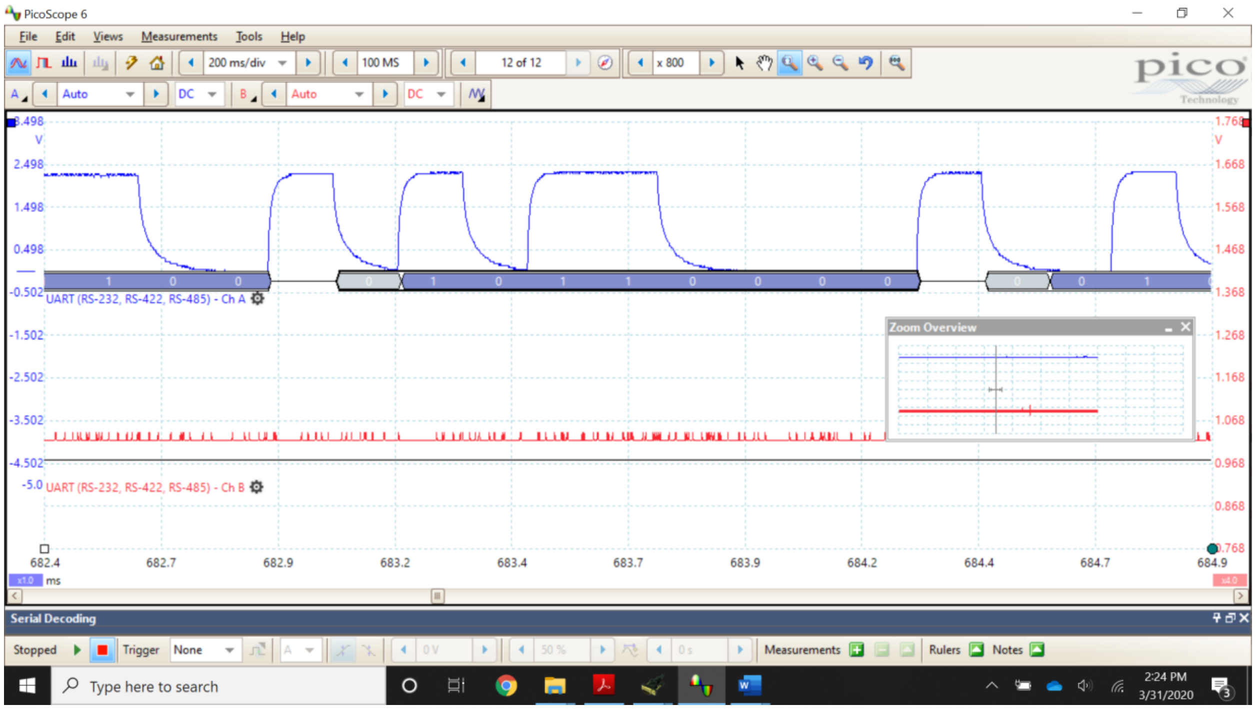
Serial transmission at 7200

Delay between sent and received packet 1: 239.7ms

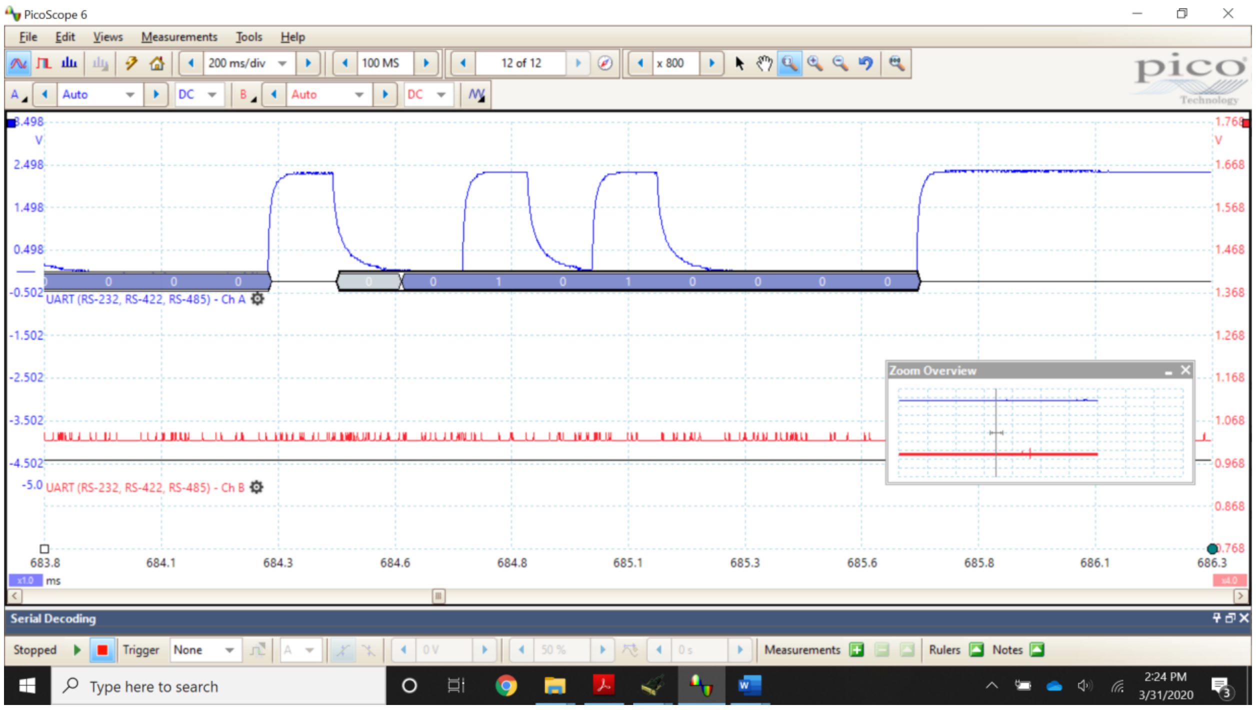
Packet 1



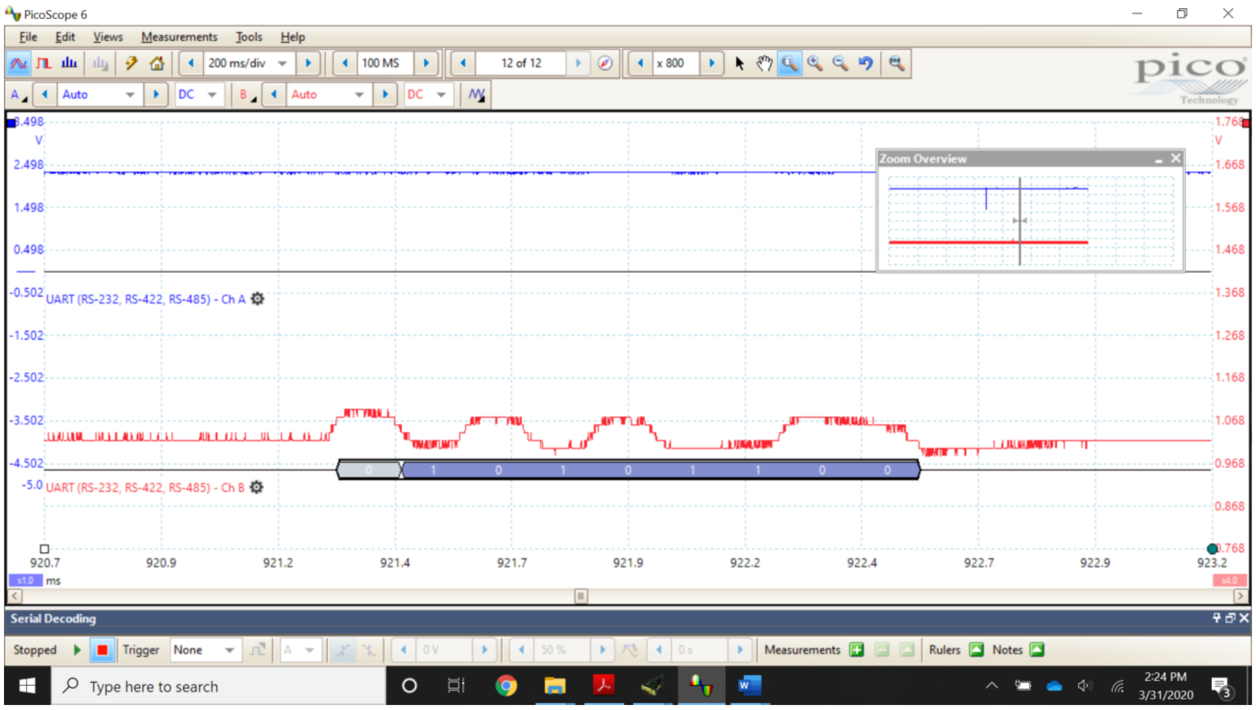
Packet 2



Packet 3



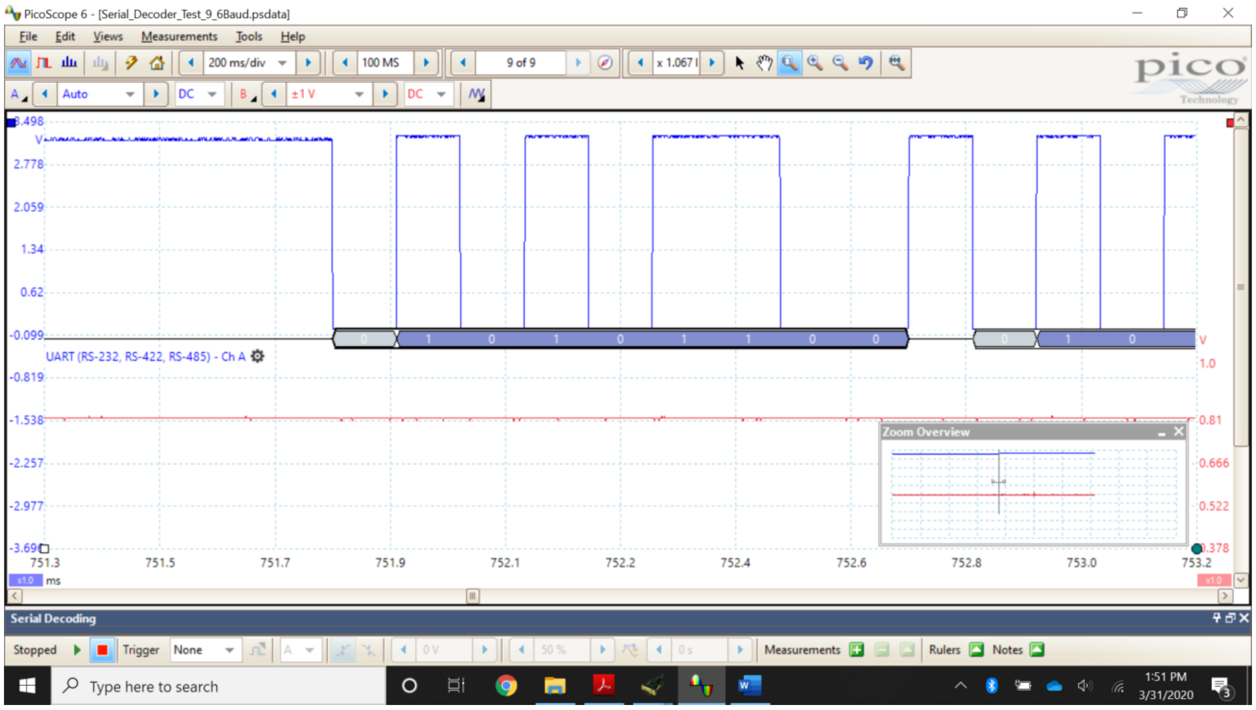
Packet 1 (received)



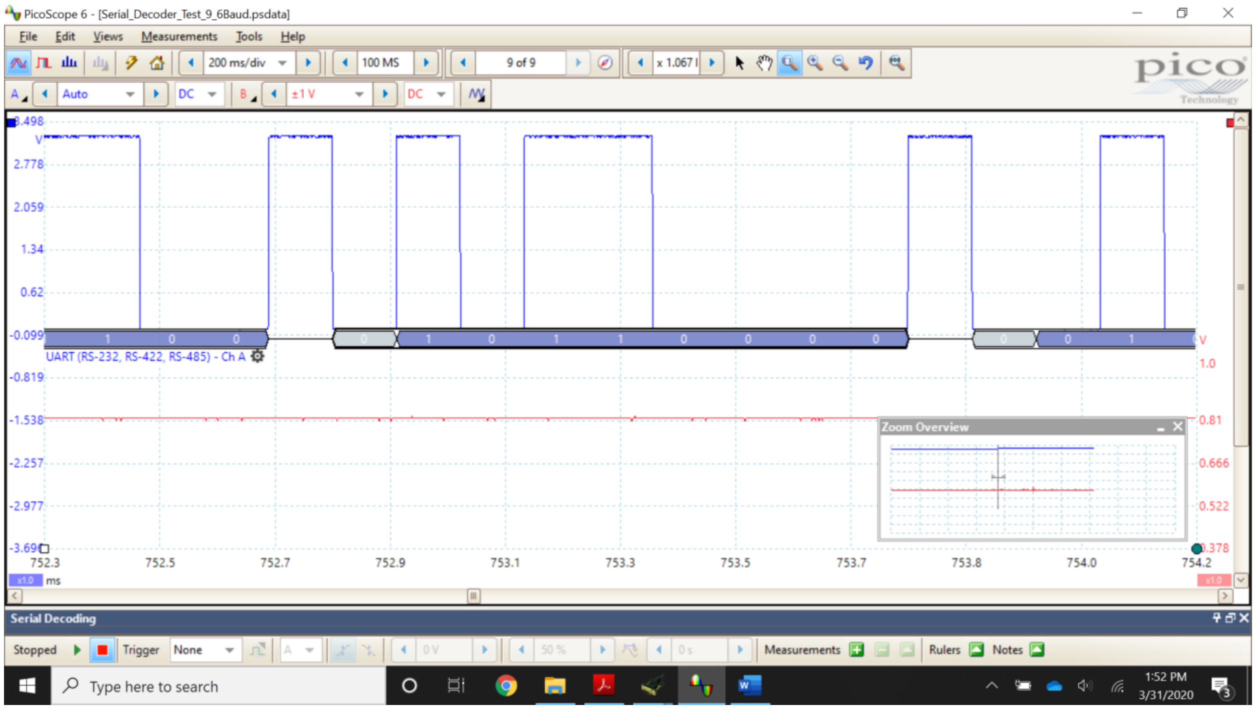
Serial Transmission at 9600

Delay between sent and received packet 1: 751.8ms

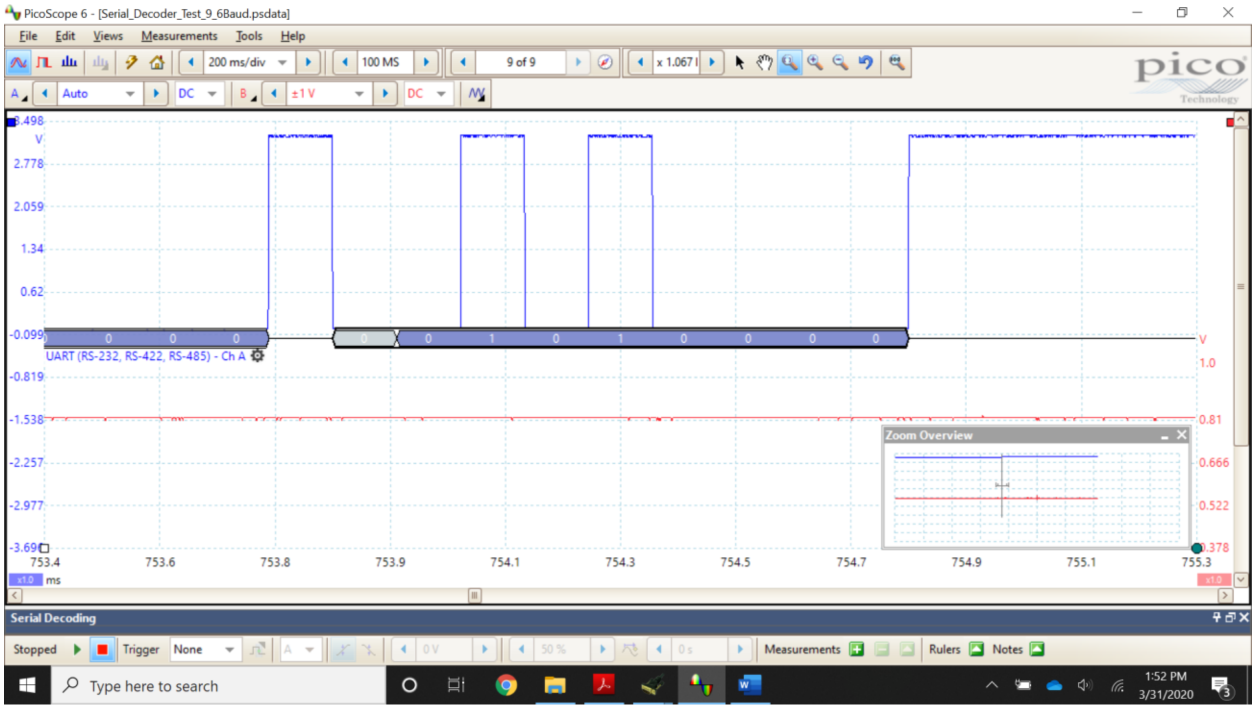
Packet 1



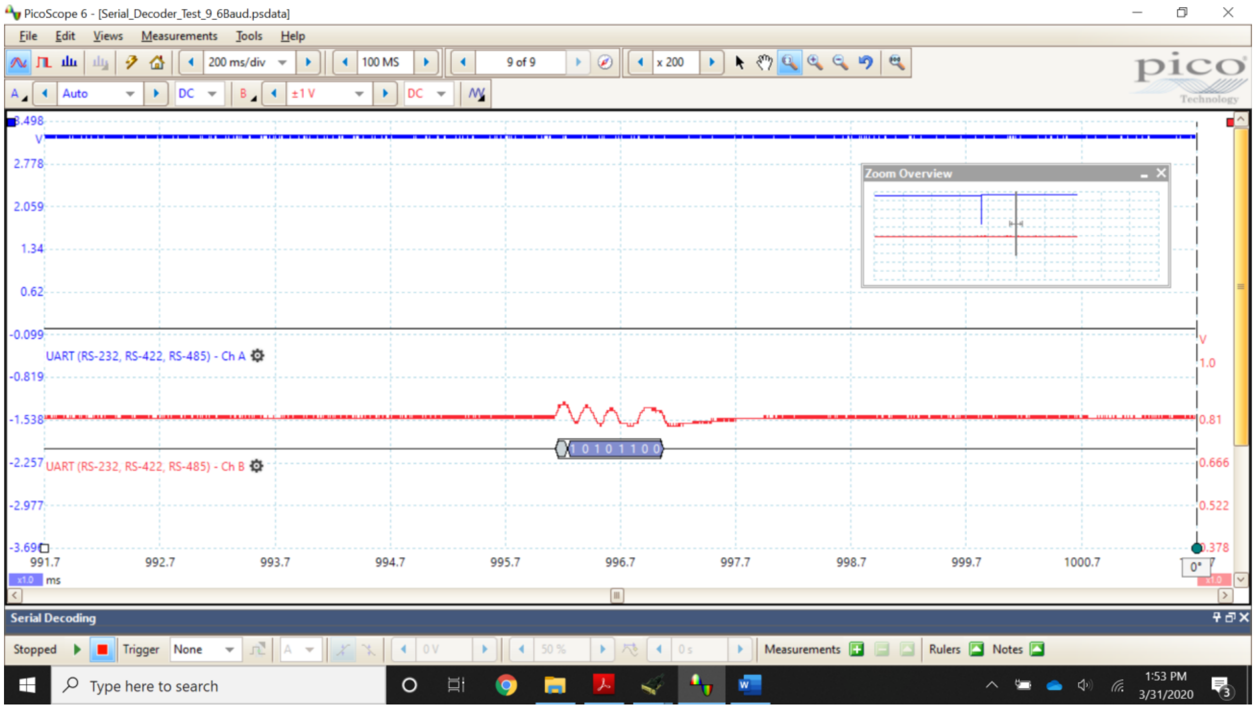
Packet 2



Packet 3



Packet 1(received)



APPENDIX C. POWERLINE MODULATION CODE

Code for powerline modulation test

Main.c

```
/**
******************************************************************************
* @file    main.c
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   Main program body
******************************************************************************
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
*
* Redistribution and use in source and binary forms, with or without modification,
* are permitted provided that the following conditions are met:
*   1. Redistributions of source code must retain the above copyright notice,
*      this list of conditions and the following disclaimer.
*   2. Redistributions in binary form must reproduce the above copyright notice,
*      this list of conditions and the following disclaimer in the documentation
*      and/or other materials provided with the distribution.
*   3. Neither the name of STMicroelectronics nor the names of its contributors
*      may be used to endorse or promote products derived from this software
*      without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
```

```
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
******************************************************************************
*/

/* Includes ------------------------------------------------------------------*/
#include "cube_hal.h"
#include "st7580_appli.h"



/** @addtogroup USER
* @{
*/

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/

/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/

int main(void);

/* Private functions ---------------------------------------------------------*/
/**
* @brief  Main program.
* @param  None
* @retval None
*/
int main(void)
{
        /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

        /* Initialize ST7580 interface */

        GPIO_PLM_Configuration();
        UART_PLM_Configuration();
```

```
	/* Initialize ST7580 PLM */
	BSP_PLM_Init();

	/* Initialize Buttons */
	BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_GPIO);

	/* Debug USART config */
	USART_PRINT_MSG_Configuration();

	/* Initialize P2P Application */
	P2P_Init();

	while(1)
	{
			/* Data Communication */
			//P2P_Process();
			AppliMasterBoard();
			//AppliSlaveBoard();
	}

}

#ifdef USE_FULL_ASSERT

/**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None
```

```
*/
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
  ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

}

#endif

/**
* @}
*/

/**
* @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

St7580_appli.c

```
/**
******************************************************************************
* @file    st7580_appli.c
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   user file to configure ST7580 PLC Modem.
*
@verbatim
==============================================================================
==========
##### How to use this driver #####
==============================================================================
==========
[..]
This file is generated automatically by STM32CubeMX and eventually modified
by the user

@endverbatim
******************************************************************************
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
*
* Redistribution and use in source and binary forms, with or without modification,
* are permitted provided that the following conditions are met:
*   1. Redistributions of source code must retain the above copyright notice,
*      this list of conditions and the following disclaimer.
*   2. Redistributions in binary form must reproduce the above copyright notice,
*      this list of conditions and the following disclaimer in the documentation
*      and/or other materials provided with the distribution.
*   3. Neither the name of STMicroelectronics nor the names of its contributors
*      may be used to endorse or promote products derived from this software
*      without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
******************************************************************************
*/

/* Includes ------------------------------------------------------------------*/
#include <string.h>
#include "cube_hal.h"
```

```
#include "st7580_appli.h"
#include "stm32_plm01a1.h"

/** @addtogroup USER
* @{
*/

/** @defgroup ST7580_APPLI
* @brief User file to configure ST7580 PLC modem.
* @{
*/

/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
#define TRIG_BUF_SIZE   1
#define ACK_BUF_SIZE      1
/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/
SM_State_t SM_State;
char MsgOut[100];
int i = 0;

/* Private function prototypes -----------------------------------------------*/
void AppliMasterBoard(void);
void AppliSlaveBoard(void);

/* Private functions ---------------------------------------------------------*/

/** @defgroup ST7580_APPLI_Private_Functions
```

```
* @{
*/

/**
* @brief  This function initializes the point-to-point communication
* @param  None
* @retval None
*/
void P2P_Init(void){

      /* Modem MIBs configuration */
  BSP_PLM_Mib_Write(MIB_MODEM_CONF, modem_config, sizeof(modem_config));
      HAL_Delay(500);

  /* Phy MIBs configuration */
  BSP_PLM_Mib_Write(MIB_PHY_CONF, phy_config, sizeof(phy_config));
      HAL_Delay(500);



      /* Check User Button state */
      if (BSP_PB_GetState(BUTTON_KEY) == GPIO_PIN_SET)
      {
            /* User Button released */
            SM_State = SM_STATE_SLAVE;
      }
      else
      {
            /* User Button pressed */
        SM_State = SM_STATE_MASTER;
      }
```

```
	return;
}

/**
* @brief  ST7580 P2P Process State machine
* @retval None.
*/
void P2P_Process() {

	switch(SM_State){
		case SM_STATE_MASTER:
		AppliMasterBoard();
		break;

		case SM_STATE_SLAVE:
		AppliSlaveBoard();
		break;
	}
	return;
}

/**
* @brief  This function handles the point-to-point Master Board Communication
* @retval None
*/
void AppliMasterBoard(){
	uint8_t ret;
	uint8_t cRxLen;
	ST7580Frame* RxFrame;
	uint8_t lastIDRcv = 0;
	int it = 0;
```

```
        uint8_t aTrsBuffer[TRIG_BUF_SIZE] = {0};
        uint8_t aRcvBuffer[ACK_BUF_SIZE];

        sprintf(MsgOut, "P2P Communication Test - Master Board Side\n\r\n\r");
        HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500);

        while(1)
        {

                /* Initialize Trigger Msg */
                aTrsBuffer[TRIG_BUF_SIZE-1]++;
                if (aTrsBuffer[TRIG_BUF_SIZE-1] > 255)
                {
                        aTrsBuffer[TRIG_BUF_SIZE-1] = 0;
                }

                sprintf(MsgOut, "Iteration %d\n\r", ++it);
                HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500);

                /* Send Trigger Msg send */
                ret = BSP_PLM_Send_Data(DATA_OPT, aTrsBuffer, TRIG_BUF_SIZE,
NULL);

                /* Check TRIGGER Msg send result */
                if(ret)
                {
                        /* Transmission Error */
                        sprintf( MsgOut, "Trigger Transmission Err\n\r");
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
                        //continue;
                }

                sprintf( MsgOut, "Trigger Msg Sent, ID: %d\n\r", aTrsBuffer[TRIG_BUF_SIZE-
1]);
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                sprintf( MsgOut, "PAYLOAD: ");
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                HAL_UART_Transmit( &pUartMsgHandle, aTrsBuffer, TRIG_BUF_SIZE,
500 );
                sprintf( MsgOut, "\n\r");
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );

                /* Wait ACK Msg sent back from slave */
                RxFrame=NULL;
                for (int j=0;((j<10) && (RxFrame==NULL));j++)
             {
                RxFrame = BSP_PLM_Receive_Frame();
                        if (RxFrame != NULL)
                        {
                                /* Check if a duplicated indication frame with STX = 03 is
received */
                                if ((RxFrame->stx == ST7580_STX_03)&&(lastIDRcv ==
RxFrame->data[3+ACK_BUF_SIZE]))
                                {
                                        RxFrame = NULL;
                                }
                                else
                                {
                                        lastIDRcv = RxFrame->data[3+ACK_BUF_SIZE];
                                        break;
                                }
                        }
                        HAL_Delay(200);
                }
                /* Check received ACK Msg */
                if (RxFrame == NULL)
                {
                        /* No ACK Msg received until timeout */
                        sprintf( MsgOut, "ACK Timeout - No ACK Received\n\r");
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
                        //continue;
                }

                cRxLen = (RxFrame->length - 4);

                sprintf( MsgOut, "ACK Msg Received\n\r");
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );

                if (cRxLen != ACK_BUF_SIZE){
                        /* ACK len mismatch */
                        sprintf( MsgOut, "Wrong ACK Length\n\r");
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
                        //continue;
```

```
		}

		/* Copy payload from RX frame */
		memcpy(aRcvBuffer,&(RxFrame->data[4]),cRxLen);

		/* Check ID to verify if the right ACK has been received */
		if (aRcvBuffer[ACK_BUF_SIZE-1] == aTrsBuffer[TRIG_BUF_SIZE-1])
		{
			sprintf( MsgOut, "ACK Msg Received, ID: %d\n\r",
aRcvBuffer[ACK_BUF_SIZE-1]);
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
		}
		else
		{
			sprintf( MsgOut, "WRONG ACK Msg Received, ID: %d\n\r",
aRcvBuffer[ACK_BUF_SIZE-1]);
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
		}
		sprintf( MsgOut, "PAYLOAD: ");
		HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
		HAL_UART_Transmit( &pUartMsgHandle, aRcvBuffer, ACK_BUF_SIZE,
500 );
		sprintf( MsgOut, "\n\r\n\r");
		HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );

		HAL_Delay(1000);
	}
```

```
}

/**
* @brief  This function handles the point-to-point Slave Board Communication
* @retval None
*/
void AppliSlaveBoard(){
        ST7580Frame* RxFrame;
        uint8_t cRxLen;
        int ret;
        uint8_t lastIDRcv = 0;
        int it =0;

        uint8_t aTrsBuffer[ACK_BUF_SIZE] = {'A','C','K'};
        uint8_t aRcvBuffer[TRIG_BUF_SIZE];

        sprintf( MsgOut, "P2P Communication Test - Slave Board Side\n\r\n\r");
        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500 );

        while(1)
        {
                sprintf(MsgOut, "Iteration %d\n\r", ++it);
                HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500);

                /* Receive Trigger Msg from M board */
                RxFrame=NULL;
                do
                {
                        RxFrame = BSP_PLM_Receive_Frame();
                        if (RxFrame != NULL)
                        {
                                /* Check if a duplicated indication frame with STX = 03 is
received */
                                if ((RxFrame->stx == ST7580_STX_03)&&(lastIDRcv ==
RxFrame->data[3+TRIG_BUF_SIZE]))
                                {
                                        RxFrame = NULL;
                                }
                                else
                                {
                                        lastIDRcv = RxFrame->data[3+TRIG_BUF_SIZE];
                                        break;
                                }
                        }
                        HAL_Delay(200);
                } while(RxFrame==NULL);
                cRxLen = (RxFrame->length - 4);
                memcpy(aRcvBuffer,&(RxFrame->data[4]),cRxLen);

                sprintf( MsgOut, "Trigger Msg Received, ID: %d\n\r",
aRcvBuffer[TRIG_BUF_SIZE-1]);
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                sprintf( MsgOut, "PAYLOAD: ");
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                HAL_UART_Transmit( &pUartMsgHandle, aRcvBuffer, TRIG_BUF_SIZE,
500 );
                sprintf( MsgOut, "\n\r");
```

```
                    HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );



                    /* Send back ACK Msg to Master Board */
                    aTrsBuffer[ACK_BUF_SIZE-1] = aRcvBuffer[TRIG_BUF_SIZE-1];
                    do
                    {
                              ret = BSP_PLM_Send_Data(DATA_OPT, aTrsBuffer, ACK_BUF_SIZE,
NULL);
                    } while (ret!=0);

                    sprintf( MsgOut, "ACK Msg Sent, ID: %d\n\r",aTrsBuffer[ACK_BUF_SIZE-1]);
                    HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                    sprintf( MsgOut, "PAYLOAD: ");
                    HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                    HAL_UART_Transmit( &pUartMsgHandle, aTrsBuffer, ACK_BUF_SIZE,
500 );
                    sprintf( MsgOut, "\n\r\n\r");
                    HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
          }

}
/**
* @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Stm32_plm01a1.h

```
/**
******************************************************************************
* @file    stm32_plm01a1.h
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   Header file for HAL related functionality of X-CUBE-PLM1
******************************************************************************
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
*
* Redistribution and use in source and binary forms, with or without modification,
* are permitted provided that the following conditions are met:
*   1. Redistributions of source code must retain the above copyright notice,
*      this list of conditions and the following disclaimer.
*   2. Redistributions in binary form must reproduce the above copyright notice,
*      this list of conditions and the following disclaimer in the documentation
*      and/or other materials provided with the distribution.
*   3. Neither the name of STMicroelectronics nor the names of its contributors
*      may be used to endorse or promote products derived from this software
*      without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
```

```
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
******************************************************************************
*/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __PLM_MODULE_CONF_H
#define __PLM_MODULE_CONF_H

/* Includes ------------------------------------------------------------------*/

#ifdef USE_STM32L0XX_NUCLEO
#include "stm32l0xx_hal.h"
#include "stm32l0xx_nucleo.h"
#include "stm32l0xx_hal_rcc.h"
#include "stm32l0xx_hal_rcc_ex.h"
#include "stm32l0xx_ll_usart.h"
#endif
```

```
#ifdef USE_STM32F4XX_NUCLEO
#include "stm32f4xx_hal.h"
#include "stm32f4xx_nucleo.h"
#include "stm32f4xx_hal_rcc.h"
#include "stm32f4xx_hal_rcc_ex.h"
#include "stm32f4xx_ll_usart.h"
#endif

#include "ST7580_Serial.h"
/* Exported types ------------------------------------------------------------*/

typedef struct sPlmDriver
{
   void( *Init )();
   int ( *Reset )();
           int ( *MibRead )(uint8_t , uint8_t * , uint8_t );
           int ( *MibWrite )(uint8_t, const uint8_t*,  uint8_t );
           int ( *MibErase )(uint8_t index );
           int ( *Ping )(const uint8_t*, uint8_t );
           int ( *PhyData )(uint8_t, const uint8_t*, uint8_t, uint8_t* );
           int ( *DlData )(uint8_t, const uint8_t*, uint8_t, uint8_t* );
           int ( *SsData )(uint8_t, const uint8_t*, uint8_t, uint8_t, uint8_t* );
           ST7580Frame * ( *NextIndicationFrame )();
}PlmDriver_t;

/* Exported constants --------------------------------------------------------*/



/* Exported macro ------------------------------------------------------------*/
/* PLM control usart */
#define PLM_USART_RxBufferSize 512

#define PLM_USART                          USART1
#define PLM_USART_BAUDRATE                 57600
#define PLM_USART_CLK_ENABLE()             __USART1_CLK_ENABLE();
#define PLM_USART_RX_GPIO_CLK_ENABLE()     __GPIOA_CLK_ENABLE()
#define PLM_USART_TX_GPIO_CLK_ENABLE()     __GPIOA_CLK_ENABLE()

#define PLM_USART_FORCE_RESET()            __USART1_FORCE_RESET()
#define PLM_USART_RELEASE_RESET()          __USART1_RELEASE_RESET()

/* Definition for USARTx Pins */
#define PLM_USART_TX_PIN                   GPIO_PIN_9
#define PLM_USART_TX_GPIO_PORT             GPIOA

#define PLM_USART_RX_PIN                   GPIO_PIN_10
#define PLM_USART_RX_GPIO_PORT             GPIOA

/* Definition for USARTx's NVIC */
#define PLM_USART_IRQn                     USART1_IRQn
#define PLM_USART_IRQHandler               USART1_IRQHandler

#if defined(USE_STM32F4XX_NUCLEO)
#define PLM_USART_TX_AF                    GPIO_AF7_USART1
#define PLM_USART_RX_AF                    GPIO_AF7_USART1
#elif defined(USE_STM32L0XX_NUCLEO)
#define PLM_USART_TX_AF                    GPIO_AF4_USART1
#define PLM_USART_RX_AF                    GPIO_AF4_USART1
#endif
```

```
/* Message debug usart */
#define MSG_USART                          USART2
#define MSG_USART_BAUDRATE                 9600
#define MSG_USART_CLK_ENABLE()             __USART2_CLK_ENABLE();
#define MSG_USART_RX_GPIO_CLK_ENABLE()     __GPIOA_CLK_ENABLE()
#define MSG_USART_TX_GPIO_CLK_ENABLE()     __GPIOA_CLK_ENABLE()

#define MSG_USART_FORCE_RESET()            __USART2_FORCE_RESET()
#define MSG_USART_RELEASE_RESET()          __USART2_RELEASE_RESET()

/* Definition for USARTx Pins */
#define MSG_USART_TX_PIN                   GPIO_PIN_2
#define MSG_USART_TX_GPIO_PORT             GPIOA

#define MSG_USART_RX_PIN                   GPIO_PIN_3
#define MSG_USART_RX_GPIO_PORT             GPIOA

/* Definition for USARTx's NVIC -- added */
#define MSG_USART_IRQn                                          USART2_IRQn
#define MSG_USART_IRQHandler                            USART2_IRQHandler
/* End added */

#if defined(USE_STM32F4XX_NUCLEO)
#define MSG_USART_TX_AF                    GPIO_AF7_USART2
#define MSG_USART_RX_AF                    GPIO_AF7_USART2

#elif defined(USE_STM32L0XX_NUCLEO)
#define MSG_USART_TX_AF                    GPIO_AF4_USART2
#define MSG_USART_RX_AF                    GPIO_AF4_USART2
#endif
```

```
*/

/* PLM Gpio */
#define PLM_GPIO_T_REQ_PORT                     GPIOA
#define PLM_GPIO_T_REQ_PIN                      GPIO_PIN_5
#define PLM_GPIO_T_REQ_CLOCK_ENABLE()           __GPIOA_CLK_ENABLE()
#define PLM_GPIO_T_REQ_CLOCK_DISABLE()          __GPIOA_CLK_DISABLE()
#define PLM_GPIO_T_REQ_SPEED                    GPIO_SPEED_HIGH
#define PLM_GPIO_T_REQ_PUPD                     GPIO_NOPULL

/****************************************************************************
*/

#define PLM_GPIO_RESETN_PORT                    GPIOA
#define PLM_GPIO_RESETN_PIN                     GPIO_PIN_8
#define PLM_GPIO_RESETN_CLOCK_ENABLE()          __GPIOA_CLK_ENABLE()
#define PLM_GPIO_RESETN_CLOCK_DISABLE()         __GPIOA_CLK_DISABLE()
#define PLM_GPIO_RESETN_SPEED                   GPIO_SPEED_HIGH
#define PLM_GPIO_RESETN_PUPD                    GPIO_NOPULL

/****************************************************************************
*/

#define PLM_PL_TX_ON_PORT                       GPIOC
#define PLM_PL_TX_ON_PIN                        GPIO_PIN_0
#define PLM_PL_TX_ON_CLOCK_ENABLE()             __GPIOC_CLK_ENABLE()
#define PLM_PL_TX_ON_CLOCK_DISABLE()            __GPIOC_CLK_DISABLE()
#define PLM_PL_TX_ON_SPEED                      GPIO_SPEED_HIGH
#define PLM_PL_TX_ON_PUPD                       GPIO_NOPULL

/****************************************************************************
*/
```

```
#define PLM_PL_RX_ON_PORT                          GPIOC
#define PLM_PL_RX_ON_PIN                           GPIO_PIN_1
#define PLM_PL_RX_ON_CLOCK_ENABLE()                __GPIOC_CLK_ENABLE()
#define PLM_PL_RX_ON_CLOCK_DISABLE()               __GPIOC_CLK_DISABLE()
#define PLM_PL_RX_ON_SPEED                         GPIO_SPEED_HIGH
#define PLM_PL_RX_ON_PUPD                          GPIO_NOPULL
#define PLM_PL_RX_ON_EXTI_LINE                     GPIO_PIN_1
#define PLM_PL_RX_ON_EXTI_MODE                     GPIO_MODE_IT_RISING_FALLING
//#define PLM_PL_RX_ON_EXTI_IRQN                   EXTI1_IRQn
#define PLM_PL_RX_ON_EXTI_PREEMPTION_PRIORITY      2
#define PLM_PL_RX_ON_EXTI_SUB_PRIORITY             2
#define PLM_PL_RX_ON_EXTI_IRQ_HANDLER              EXTI1_IRQHandler


/* Exported Variables -------------------------------------------------------*/
extern UART_HandleTypeDef pUartPlmHandle;
extern UART_HandleTypeDef pUartMsgHandle;

extern PlmDriver_t *pPlmDriver;

void GPIO_PLM_Configuration(void);
void UART_PLM_Configuration(void);
void USART_PRINT_MSG_Configuration(void);

void BSP_PLM_Init(void);
int BSP_PLM_Reset(void);
int BSP_PLM_Mib_Write(uint8_t indexMib, const uint8_t* bufMib, uint8_t lenBuf);
int BSP_PLM_Mib_Read(uint8_t indexMib, uint8_t* bufMib, uint8_t lenBuf);
int BSP_PLM_Mib_Erase(uint8_t indexMib);
int BSP_PLM_Ping(const uint8_t* pingBuf, uint8_t pingLen);
int BSP_PLM_Send_Data(uint8_t plmOpts, const uint8_t* dataBuf, uint8_t dataLen, uint8_t*
confData);
int BSP_PLM_Send_Secure_data(uint8_t plmOpts, const uint8_t* dataBuf, uint8_t clrLen,
uint8_t encLen, uint8_t* retData);
ST7580Frame *BSP_PLM_Receive_Frame(void);
#endif //__PLM_MODULE_CONF_H
```

Powerline Modulation Communication Test

MasterBoard

Main.c

```
/**
******************************************************************************
* @file    main.c
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   Main program body
******************************************************************************
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
*
* Redistribution and use in source and binary forms, with or without modification,
* are permitted provided that the following conditions are met:
*   1. Redistributions of source code must retain the above copyright notice,
*      this list of conditions and the following disclaimer.
*   2. Redistributions in binary form must reproduce the above copyright notice,
*      this list of conditions and the following disclaimer in the documentation
*      and/or other materials provided with the distribution.
*   3. Neither the name of STMicroelectronics nor the names of its contributors
*      may be used to endorse or promote products derived from this software
*      without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE
```

```
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
******************************************************************************
*/

/* Includes ------------------------------------------------------------------*/
#include "cube_hal.h"
#include "st7580_appli.h"

// Adding from led_test code
//#include "usart.h" things in this defined elsewhere
#include <stdio.h>



/** @addtogroup USER
* @{
*/
```

```
/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#define GETCHAR_PROTOTYPE int __io_getchar(void)
/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/
int test = 0;
/* Private function prototypes -----------------------------------------------*/

int main(void);

/* Private functions ---------------------------------------------------------*/
/**
* @brief  Main program.
* @param  None
* @retval None
*/
int main(void)
{
	/* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

	/* Initialize ST7580 interface */
```

```
GPIO_PLM_Configuration();
UART_PLM_Configuration();

/* Initialize ST7580 PLM */
BSP_PLM_Init();

/* Initialize Buttons */
BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_GPIO);

/* Debug USART config */
USART_PRINT_MSG_Configuration();

/* Initialize P2P Application */
P2P_Init();

// Turning off buffers so io happens immediately hopefully doesnt mess up plm...........
setvbuf(stdin, NULL, _IONBF, 0);
setvbuf(stdout, NULL, _IONBF, 0);
setvbuf(stderr, NULL, _IONBF, 0);

uint8_t c;

   //printf( "Enter a value :");
   c = getchar( );

   //printf( "\nYou entered: ");
   //putchar( c );
   //printf( "\nDoing PLM\n");
   AppliMasterBoard(c);
   //printf("s"); //printf( "\nPLM Success\n");
```

```
		return 0;

//	while(1)
//	{
//				/* Take button press from st7580_appli.c  */
//			if (BSP_PB_GetState(BUTTON_KEY) == GPIO_PIN_SET)
//			{
//				/* User Button released */
//			}
//			else{
//				AppliMasterBoard(test);
//				test++;
//			}
//
//			//P2P_Process(); //Remove for master only
//	}

}

PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
  HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)&ch, 1, 0xFFFF); //&pUartMsgHandle
&huart2

  return ch;
}

GETCHAR_PROTOTYPE
```

```
{
HAL_StatusTypeDef Status = HAL_BUSY;
uint8_t Data;

while(Status != HAL_OK)
Status = HAL_UART_Receive(&pUartMsgHandle, &Data, 1, 10); //&pUartMsgHandle &huart2

return(Data);
}

#ifdef USE_FULL_ASSERT

/**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None
*/
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
  ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

}

#endif

/**
* @}
*/
```

```
/**
* @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

St7580_appli.c

```
/**
******************************************************************************
* @file    st7580_appli.c
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   user file to configure ST7580 PLC Modem.
*
@verbatim
==============================================================================
==========
##### How to use this driver #####
==============================================================================
==========
[..]
This file is generated automatically by STM32CubeMX and eventually modified
by the user

@endverbatim
******************************************************************************
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
```

```
*
******************************************************************************
*/

/* Includes ------------------------------------------------------------------*/
#include <string.h>
#include <stdio.h> //added due to warning
#include "cube_hal.h"
#include "st7580_appli.h"
#include "stm32_plm01a1.h"

/** @addtogroup USER
* @{
*/

/** @defgroup ST7580_APPLI
* @brief User file to configure ST7580 PLC modem.
* @{
*/

/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
#define TRIG_BUF_SIZE   1 //21
#define ACK_BUF_SIZE      1 //17
/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/
SM_State_t SM_State;
char MsgOut[100];
```

```
/* Private function prototypes -----------------------------------------------*/
void AppliMasterBoard(int data);
void AppliSlaveBoard(void);

/* Private functions ---------------------------------------------------------*/

/** @defgroup ST7580_APPLI_Private_Functions
* @{
*/

/**
* @brief  This function initializes the point-to-point communication
* @param  None
* @retval None
*/
void P2P_Init(void){

        /* Modem MIBs configuration */
  BSP_PLM_Mib_Write(MIB_MODEM_CONF, modem_config, sizeof(modem_config));
        HAL_Delay(500);

  /* Phy MIBs configuration */
  BSP_PLM_Mib_Write(MIB_PHY_CONF, phy_config, sizeof(phy_config));
        HAL_Delay(500);



        //Remove button press to select master/slave status
//      /* Check User Button state */
//      if (BSP_PB_GetState(BUTTON_KEY) == GPIO_PIN_SET)
//      {
//              /* User Button released */
//			SM_State = SM_STATE_SLAVE;
//		}
//		else
//		{
//			/* User Button pressed */
//		  SM_State = SM_STATE_MASTER;
//		}
//
//		return;
}

/**
* @brief  ST7580 P2P Process State machine
* @retval None.
*/

//not using this
//void P2P_Process() {
//		switch(SM_State){
//			case SM_STATE_MASTER:
//			AppliMasterBoard();
//			break;
//
//			case SM_STATE_SLAVE:
//			AppliSlaveBoard();
//			break;
//		}
//		return;
//}

/**
```

```
* @brief  This function handles the point-to-point Master Board Communication
* @retval None
*/
void AppliMasterBoard(int data){
        uint8_t ret;
        uint8_t cRxLen;
        ST7580Frame* RxFrame;
        uint8_t lastIDRcv = 0;
        int it = 0;

        uint8_t aTrsBuffer[TRIG_BUF_SIZE] = {0}; //{'T','R','I','G','G','E','R',' ',\
                                                                                    //
        'M','E','S','S','A','G','E',' ',\
                                                                                    //
        'I','D',':',' ','@'};
        uint8_t aRcvBuffer[ACK_BUF_SIZE];

        //sprintf(MsgOut, "P2P Communication Test - Master Board Side\n\r\n\r");
        //HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500);

        //while(1) //removed while
        //{             //removed while remove this too
                /* Initialize Trigger Msg */
                aTrsBuffer[TRIG_BUF_SIZE-1] = data;

// no while = no purpose for this
//              if (aTrsBuffer[TRIG_BUF_SIZE-1] > 255) //from 'Z' to 255
//              {
//                      aTrsBuffer[TRIG_BUF_SIZE-1] = 0; //from 'A' to 0
//              }
```

```
			//not meaningful for debugging w/o while loop
//		sprintf(MsgOut, "Iteration %d\n\r", ++it);
//		HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500);

			/* Send Trigger Msg send */
			ret = BSP_PLM_Send_Data(DATA_OPT, aTrsBuffer, TRIG_BUF_SIZE,
NULL);

			/* Check TRIGGER Msg send result */
			if(ret)
			{
				/* Transmission Error */
				sprintf( MsgOut, "Trigger Transmission Err\n\r");
				HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
				//continue; //removed while remove this too
			}

			//sprintf( MsgOut, "Trigger Msg Sent, ID: %d\n\r",
aTrsBuffer[TRIG_BUF_SIZE-1]); //change %c to %d
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
			//sprintf( MsgOut, "PAYLOAD: ");
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
			//HAL_UART_Transmit( &pUartMsgHandle, aTrsBuffer, TRIG_BUF_SIZE,
500 );
			//sprintf( MsgOut, "\n\r");
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
```

```
		/* Wait ACK Msg sent back from slave */
		RxFrame=NULL;
		for (int j=0;((j<10) && (RxFrame==NULL));j++)
		{
			RxFrame = BSP_PLM_Receive_Frame();
			if (RxFrame != NULL)
			{
				/* Check if a duplicated indication frame with STX = 03 is
received */
				if ((RxFrame->stx == ST7580_STX_03)&&(lastIDRcv ==
RxFrame->data[3+ACK_BUF_SIZE]))
				{
					RxFrame = NULL;
				}
				else
				{
					lastIDRcv = RxFrame->data[3+ACK_BUF_SIZE];
					break;
				}
			}
			HAL_Delay(200); //initial 200
		}
		/* Check received ACK Msg */

		//hopefully no timeout
//		if (RxFrame == NULL)
//		{
//			/* No ACK Msg received until timeout */
//			sprintf( MsgOut, "ACK Timeout - No ACK Received\n\r");
//			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
```

```
//				//continue; //removed while remove this too
//			}

			cRxLen = (RxFrame->length - 4);

			//sprintf( MsgOut, "ACK Msg Received\n\r");
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );

			//ignoring for now
			if (cRxLen != ACK_BUF_SIZE){
				/* ACK len mismatch */
				sprintf( MsgOut, "Wrong ACK Length\n\r");
				HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
				//continue; //removed while remove this too
			}

			/* Copy payload from RX frame */
			memcpy(aRcvBuffer,&(RxFrame->data[4]),cRxLen);

			/* Check ID to verify if the right ACK has been received */
//			if (aRcvBuffer[ACK_BUF_SIZE-1] == aTrsBuffer[TRIG_BUF_SIZE-1])
//			{
				//sprintf( MsgOut, "ACK Msg Received, ID: %d\n\r",
aRcvBuffer[ACK_BUF_SIZE-1]); //change %c to %d
				//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
//			}
//			else
//			{
```

```
//                              sprintf( MsgOut, "WRONG ACK Msg Received, ID: %d\n\r",
aRcvBuffer[ACK_BUF_SIZE-1]); //change %c to %d
//                              HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
//                      }
                        //sprintf( MsgOut, "PAYLOAD: ");
                        //HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
                        //HAL_UART_Transmit( &pUartMsgHandle, aRcvBuffer, ACK_BUF_SIZE,
500 );
                        //sprintf( MsgOut, "\n\r\n\r");
                        //HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );

                        HAL_Delay(1000);
            //} //removed while remove this too

}

/**
* @brief  This function handles the point-to-point Slave Board Communication
* @retval None
*/
void AppliSlaveBoard(){
            ST7580Frame* RxFrame;
            uint8_t cRxLen;
            int ret;
            uint8_t lastIDRcv = 0;
            int it =0;

            uint8_t aTrsBuffer[ACK_BUF_SIZE] = {'A','C','K'}; //,' ','M','E','S','S',\
// 'A','G','E',' ','I','D',':',' ',\

// '@'};
        uint8_t aRcvBuffer[TRIG_BUF_SIZE];

        sprintf( MsgOut, "P2P Communication Test - Slave Board Side\n\r\n\r");
        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500 );

        while(1)
        {
                sprintf(MsgOut, "Iteration %d\n\r", ++it); //change %c to %d
                HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500);

                /* Receive Trigger Msg from M board */
                RxFrame=NULL;
                do
                {
                        RxFrame = BSP_PLM_Receive_Frame();

                        if (RxFrame != NULL)
                        {
                                /* Check if a duplicated indication frame with STX = 03 is
received */
                                if ((RxFrame->stx == ST7580_STX_03)&&(lastIDRcv ==
RxFrame->data[3+TRIG_BUF_SIZE]))
                                {
                                        RxFrame = NULL;
                                }
                                else
                {
                    lastIDRcv = RxFrame->data[3+TRIG_BUF_SIZE];
                    break;
                }
            }
            HAL_Delay(200);
        } while(RxFrame==NULL);

        cRxLen = (RxFrame->length - 4);
        memcpy(aRcvBuffer,&(RxFrame->data[4]),cRxLen);

        sprintf( MsgOut, "Trigger Msg Received, ID: %d\n\r",
aRcvBuffer[TRIG_BUF_SIZE-1]);//change %c to %d
        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
        sprintf( MsgOut, "PAYLOAD: ");
        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
        HAL_UART_Transmit( &pUartMsgHandle, aRcvBuffer, TRIG_BUF_SIZE,
500 );
        sprintf( MsgOut, "\n\r");
        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );



        /* Send back ACK Msg to Master Board */
        aTrsBuffer[ACK_BUF_SIZE-1] = aRcvBuffer[TRIG_BUF_SIZE-1];
        do
        {
            ret = BSP_PLM_Send_Data(DATA_OPT, aTrsBuffer, ACK_BUF_SIZE,
NULL);
```

```
                        } while (ret!=0);

                        sprintf( MsgOut, "ACK Msg Sent, ID: %d\n\r",aTrsBuffer[ACK_BUF_SIZE-
1]);//change %c to %d
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                        sprintf( MsgOut, "PAYLOAD: ");
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                        HAL_UART_Transmit( &pUartMsgHandle, aTrsBuffer, ACK_BUF_SIZE,
500 );
                        sprintf( MsgOut, "\n\r\n\r");
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
}

}
/**
* @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

SlaveBoard

Main.c

```
/**
******************************************************************************
* @file    main.c
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   Main program body
```

```
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
******************************************************************************
*/

/* Includes ------------------------------------------------------------------*/
#include "cube_hal.h"
#include "st7580_appli.h"



// Adding from led_test code
#include <stdio.h>

/** @addtogroup USER
* @{
*/

/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#define GETCHAR_PROTOTYPE int __io_getchar(void)
/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/
```

```
int main(void);

/* Private functions ---------------------------------------------------------*/
/**
* @brief  Main program.
* @param  None
* @retval None
*/
int main(void)
{
	/* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

	/* Initialize ST7580 interface */

	GPIO_PLM_Configuration();
	UART_PLM_Configuration();

	/* Initialize ST7580 PLM */
	BSP_PLM_Init();

	/* Initialize Buttons */
	BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_GPIO);

	/* Debug USART config */
	USART_PRINT_MSG_Configuration();

	/* Initialize P2P Application */
	P2P_Init();

		// Turning off buffers so io happens immediately hopefully doesnt mess up
plm...........
		setvbuf(stdin, NULL, _IONBF, 0);
		setvbuf(stdout, NULL, _IONBF, 0);
		setvbuf(stderr, NULL, _IONBF, 0);

	while(1)
	{
			/* Data Communication */
			//P2P_Process();
			AppliSlaveBoard();
	}

}

PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
  HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)&ch, 1, 0xFFFF); //&pUartMsgHandle
&huart2

  return ch;
}

GETCHAR_PROTOTYPE
```

```
{
HAL_StatusTypeDef Status = HAL_BUSY;
uint8_t Data;

while(Status != HAL_OK)
Status = HAL_UART_Receive(&pUartMsgHandle, &Data, 1, 10); //&pUartMsgHandle &huart2

return(Data);
}

#ifdef USE_FULL_ASSERT

/**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None
*/
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
  ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

}

#endif

/**
* @}
*/
```

```
/**
* @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

St7580_appli.c

```
/**
******************************************************************************
* @file    st7580_appli.c
* @author  CLAB
* @version 1.1.0
* @date    18-Sept-2017
* @brief   user file to configure ST7580 PLC Modem.
*
@verbatim
==============================================================================
==========
##### How to use this driver #####
==============================================================================
==========
[..]
This file is generated automatically by STM32CubeMX and eventually modified
by the user

@endverbatim
******************************************************************************
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
```

```
*
******************************************************************************
*/

/* Includes ------------------------------------------------------------------*/
#include <string.h>
#include <stdio.h> //added due to warning
#include "cube_hal.h"
#include "st7580_appli.h"
#include "stm32_plm01a1.h"

/** @addtogroup USER
* @{
*/

/** @defgroup ST7580_APPLI
* @brief User file to configure ST7580 PLC modem.
* @{
*/

/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
#define TRIG_BUF_SIZE   1 //21
#define ACK_BUF_SIZE      1 //17
/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/
SM_State_t SM_State;
char MsgOut[100];
```

```
/* Private function prototypes -----------------------------------------------*/
void AppliMasterBoard(int data);
void AppliSlaveBoard(void);

/* Private functions ---------------------------------------------------------*/

/** @defgroup ST7580_APPLI_Private_Functions
* @{
*/

/**
* @brief  This function initializes the point-to-point communication
* @param  None
* @retval None
*/
void P2P_Init(void){

	/* Modem MIBs configuration */
  BSP_PLM_Mib_Write(MIB_MODEM_CONF, modem_config, sizeof(modem_config));
	HAL_Delay(500);

  /* Phy MIBs configuration */
  BSP_PLM_Mib_Write(MIB_PHY_CONF, phy_config, sizeof(phy_config));
	HAL_Delay(500);



	//Remove button press to select master/slave status
//	/* Check User Button state */
//	if (BSP_PB_GetState(BUTTON_KEY) == GPIO_PIN_SET)
//	{
//		/* User Button released */
//			SM_State = SM_STATE_SLAVE;
//		}
//		else
//		{
//			/* User Button pressed */
//		  SM_State = SM_STATE_MASTER;
//		}
//
//		return;
}

/**
* @brief  ST7580 P2P Process State machine
* @retval None.
*/

//not using this
//void P2P_Process() {
//		switch(SM_State){
//			case SM_STATE_MASTER:
//			AppliMasterBoard();
//			break;
//
//			case SM_STATE_SLAVE:
//			AppliSlaveBoard();
//			break;
//		}
//		return;
//}

/**
```

```
* @brief  This function handles the point-to-point Master Board Communication
* @retval None
*/
void AppliMasterBoard(int data){
        uint8_t ret;
        uint8_t cRxLen;
        ST7580Frame* RxFrame;
        uint8_t lastIDRcv = 0;
        int it = 0;

        //make length 1
        uint8_t aTrsBuffer[TRIG_BUF_SIZE] = {0}; //{'T','R','I','G','G','E','R',' ',\
                                                                                    //
        'M','E','S','S','A','G','E',' ',\
                                                                                    //
        'I','D',':',' ','@'};
        uint8_t aRcvBuffer[ACK_BUF_SIZE];

        sprintf(MsgOut, "P2P Communication Test - Master Board Side\n\r\n\r");
        HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500);

        //while(1) //removed while
        //{             //removed while remove this too
                /* Initialize Trigger Msg */
                aTrsBuffer[TRIG_BUF_SIZE-1] = data;

// no while = no purpose for this
//              if (aTrsBuffer[TRIG_BUF_SIZE-1] > 255) //from 'Z' to 255
//              {
//                      aTrsBuffer[TRIG_BUF_SIZE-1] = 0; //from 'A' to 0
//              }
```

```
			//not meaningful for debugging w/o while loop
//		sprintf(MsgOut, "Iteration %d\n\r", ++it);
//		HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500);

			/* Send Trigger Msg send */
			ret = BSP_PLM_Send_Data(DATA_OPT, aTrsBuffer, TRIG_BUF_SIZE,
NULL);

			/* Check TRIGGER Msg send result */
			if(ret)
			{
				/* Transmission Error */
				sprintf( MsgOut, "Trigger Transmission Err\n\r");
				HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
				//continue; //removed while remove this too
			}

			sprintf( MsgOut, "Trigger Msg Sent, ID: %d\n\r", aTrsBuffer[TRIG_BUF_SIZE-
1]); //change %c to %d
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
			sprintf( MsgOut, "PAYLOAD: ");
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
			HAL_UART_Transmit( &pUartMsgHandle, aTrsBuffer, TRIG_BUF_SIZE,
500 );
			sprintf( MsgOut, "\n\r");
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
```

```
		/* Wait ACK Msg sent back from slave */
		RxFrame=NULL;
		for (int j=0;((j<10) && (RxFrame==NULL));j++)
		{
			RxFrame = BSP_PLM_Receive_Frame();
			if (RxFrame != NULL)
			{
				/* Check if a duplicated indication frame with STX = 03 is
received */
				if ((RxFrame->stx == ST7580_STX_03)&&(lastIDRcv ==
RxFrame->data[3+ACK_BUF_SIZE]))
				{
					RxFrame = NULL;
				}
				else
				{
					lastIDRcv = RxFrame->data[3+ACK_BUF_SIZE];
					break;
				}
			}
			HAL_Delay(200);
		}
		/* Check received ACK Msg */
		if (RxFrame == NULL)
		{
			/* No ACK Msg received until timeout */
			sprintf( MsgOut, "ACK Timeout - No ACK Received\n\r");
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
			//continue; //removed while remove this too
		}

		cRxLen = (RxFrame->length - 4);

		sprintf( MsgOut, "ACK Msg Received\n\r");
		HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );

		if (cRxLen != ACK_BUF_SIZE){
			/* ACK len mismatch */
			sprintf( MsgOut, "Wrong ACK Length\n\r");
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
			//continue; //removed while remove this too
		}

		/* Copy payload from RX frame */
		memcpy(aRcvBuffer,&(RxFrame->data[4]),cRxLen);

		/* Check ID to verify if the right ACK has been received */
		if (aRcvBuffer[ACK_BUF_SIZE-1] == aTrsBuffer[TRIG_BUF_SIZE-1])
		{
			sprintf( MsgOut, "ACK Msg Received, ID: %d\n\r",
aRcvBuffer[ACK_BUF_SIZE-1]); //change %c to %d
			HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
		}
		else
		{
			sprintf( MsgOut, "WRONG ACK Msg Received, ID: %d\n\r",
aRcvBuffer[ACK_BUF_SIZE-1]); //change %c to %d
```

```
                        HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
                }
                sprintf( MsgOut, "PAYLOAD: ");
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
                HAL_UART_Transmit( &pUartMsgHandle, aRcvBuffer, ACK_BUF_SIZE,
500 );
                sprintf( MsgOut, "\n\r\n\r");
                HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );

                HAL_Delay(1000);
        //} //removed while remove this too

}

/**
* @brief  This function handles the point-to-point Slave Board Communication
* @retval None
*/
void AppliSlaveBoard(){
        ST7580Frame* RxFrame;
        uint8_t cRxLen;
        int ret;
        uint8_t lastIDRcv = 0;
        int it =0;
        int c; //added

        uint8_t aTrsBuffer[ACK_BUF_SIZE] = {0}; //,' ','M','E','S','S',\
// 'A','G','E',' ','I','D',':',' ',\

// '@'};
        uint8_t aRcvBuffer[TRIG_BUF_SIZE];

        //test
        //sprintf( MsgOut, "P2P Communication Test - Slave Board Side\n\r\n\r");
        //HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut), 500 );

        while(1)
        {
                //test
                //sprintf(MsgOut, "Iteration %d\n\r", ++it); //change %c to %d
                //HAL_UART_Transmit(&pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500);

                /* Receive Trigger Msg from M board */
                RxFrame=NULL;
                do
                {
                        RxFrame = BSP_PLM_Receive_Frame();

                        if (RxFrame != NULL)
                        {
                                /* Check if a duplicated indication frame with STX = 03 is
received */
                                if ((RxFrame->stx == ST7580_STX_03)&&(lastIDRcv ==
RxFrame->data[3+TRIG_BUF_SIZE]))
                                {
                                        RxFrame = NULL;
```

```
                    }
                    else
                    {
                              lastIDRcv = RxFrame->data[3+TRIG_BUF_SIZE];
                              break;
                    }
          }
          HAL_Delay(200);
} while(RxFrame==NULL);

cRxLen = (RxFrame->length - 4);
memcpy(aRcvBuffer,&(RxFrame->data[4]),cRxLen);

//                  sprintf( MsgOut, "Trigger Msg Received, ID: %d\n\r",
aRcvBuffer[TRIG_BUF_SIZE-1]);//change %c to %d
//                  HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
//                  sprintf( MsgOut, "PAYLOAD: ");
//                  HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );
//                  HAL_UART_Transmit( &pUartMsgHandle, aRcvBuffer, TRIG_BUF_SIZE,
500 );
//                  sprintf( MsgOut, "\n\r");
//                  HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut, strlen(MsgOut),
500 );

                    /* Upload new data */
                    //putchar( RxFrame );//if not used causes 01 output instead of 1
      //printf( "Enter a value :");
      //c = getchar( );
      //double inputs are because it receiving the plm data on interrupt???
```

```
	//printf( "\nYou entered: ");
	//( c );

	//printf( "\nUpdate trsbuffer ");
	//aTrsBuffer[ACK_BUF_SIZE-1] = c;

			/* Send back ACK Msg to Master Board */
			aTrsBuffer[ACK_BUF_SIZE-1] = aRcvBuffer[TRIG_BUF_SIZE-1];
			do
			{
				ret = BSP_PLM_Send_Data(DATA_OPT, aTrsBuffer, ACK_BUF_SIZE,
NULL);
			} while (ret!=0);

			//sprintf( MsgOut, "ACK Msg Sent, ID: %d\n\r",aTrsBuffer[ACK_BUF_SIZE-
1]);//change %c to %d
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
			//sprintf( MsgOut, "PAYLOAD: ");
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
			HAL_UART_Transmit( &pUartMsgHandle, aTrsBuffer, ACK_BUF_SIZE,
500 );
			//sprintf( MsgOut, "\n\r\n\r");
			//HAL_UART_Transmit( &pUartMsgHandle, (uint8_t *)MsgOut,
strlen(MsgOut), 500 );
}

}
/**
```

```
* @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```