REMOTE SENSING FOR BRIDGE SCOUR

by

Kirsten Louis McCane

BS, University of Pittsburgh, 2007

Submitted to the Graduate Faculty of

The Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2009

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Kirsten Louis McCane

It was defended on

March 26, 2009

and approved by

James T Cain, Professor Emeritus, Department of Electrical and Computer Engineering

Ronald G. Hoelzeman, Associate Professor, Department of Electrical and Computer Engineering

Zhi-Hong Mao, Assistant Professor, Department of Electrical and Computer Engineering and Department of Bioengineering

Thesis Advisor: Marlin H. Mickle, Nickolas A. DeCecco Professor, Department of Electrical and Computer Engineering

REMOTE SENSING FOR BRIDGE SCOUR

Kirsten Louis McCane, MS

University of Pittsburgh, 2009

Pennsylvania has one of the largest number of bridges in the nation with over 22,000 bridges statewide [1]. Most of these bridges are over waterways. As a result these bridges are susceptible to bridge scour, the washing away of fill around structures, which compromises the safety of the bridge. Bridge inspections have a limited frequency at which they can occur. In between these inspections, events may happen that would indicate immediate action be taken. In some cases, the action necessary may be to shut down the bridge to protect the safety of travelers. Additionally, during floods, even if an inspection team is present at a bridge, their traditional means in acquiring data about the state of the bridge could be severely limited. In an effort to continually monitor the health of these bridges, state and federal departments of transportations are investigating different monitoring methods and instrumentation. Notwithstanding, the Pennsylvania Department of Transportation (PennDOT) needs a low cost monitoring solution to supplement their current monitoring infrastructure.

Float out devices are an option that PennDOT has identified as a promising addition to their bridge scour monitoring capabilities. The float out device concept is to bury transmitters at various locations around a bridge structure which would eventually cause them to be released due to the scour's removal of material around the device. A receiver on the bridge would receive the transmission and perform an action. A float out device system would provide an initial indication of scour severity for further investigation. A prototype system using a float out device has been developed for use by PennDOT. This system uses commercial off the shelf (COTS) parts and conforms to the available installation and operations means of the department. The

system has been tested in a laboratory setting emulating its target environment and has proven adept at performing its required tasks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I would like to give special thanks to Dr. Mickle and Dr. Hawrylak for all their advice and support. I am forever grateful for the opportunities they have provided me.

1.0 INTRODUCTION

The monitoring of bridge scour has been pursued with increasing priority in recent years. Research into different monitoring options by state departments of transportation and federal departments has been mandated. The following subsections will introduce bridge scour, its importance, and how its prevention and monitoring is being approached. The research exhibited within this document is in response to PennDOT's pursuit of remote monitoring of bridge scour. PennDOT has considered several monitoring techniques and technologies including the method pursued in the project presented in this document. These monitoring solutions will also be reviewed.

1.1 BACKGROUND

One of the biggest safety hazards to bridge stability is the deterioration and removal of materials around a bridge's piers and abutments**.** This is known as bridge scour. Bridge scour mainly occurs as a result of flowing water moving material from the bed and banks surrounding the bridge's piers and abutments. This effect is magnified in flood situations. The result of bridge scour is the weakening of the bridge structure such that failure or collapse may occur [2][5].

Bridge failures merit a great deal of attention due to their cost of life and the large monetary cost to local, state, and federal government. Documented failures due to bridge scour and its results are numerous. For example, a 1993 flood in the upper Mississippi basin, caused 23 bridge failures for an estimated damage of $15 million. The modes of bridge failures were 14 from abutment scour, two from pier scour, three from pier and abutment scour, two from lateral bank migration, one from debris load, and one from unknown cause [1]. In another case, it is reported that 7,650 bridges are run by the Forest Service, U.S. Department of Agriculture, on National Forest lands. Almost all of these bridges are located over water. It was shown that scour is the single most common cause for bridge damage and failure on these National Forest lands[5]. Often, these bridge failures result in death. A highly studied and documented case of such a bridge failure is the collapse of the Schoharie Creek Bridge on the New York State Thruway in April 1987 during a record level flood. This collapse killed 10 people and resulted in a great amount of national attention. The National Transportation Safety Board concluded that bridge scour was the main cause of the collapse [12].

The prevention and monitoring of bridge scour is clearly necessary for the safety of bridges around the country. In response to events such as the Schoharie Creek Bridge collapse, agencies such as the Federal Highway Administration began setting up guidelines for how transportation departments should conduct bridge scour analysis and potential solutions. The Technical Advisory issued by FHWA in 1988 provided recommendations for developing and implementing a scour evaluation program [13]. A collection of three manuals contains these guidelines and are the standard for scour evaluation programs within the field. These documents are HEC-18 Evaluating Scour at Bridges, HEC-20 Stream Stability at Highway Structures, HEC-23 Bridge Scour and Stream Instability Countermeasures. The third document contains a review

of countermeasures implemented by state departments of transportation. In this case, countermeasures are described as measures incorporated into a highway-stream crossing system to monitor, control, inhibit, change, delay, or minimize stream instability and bridge scour problems.

Among these countermeasures, monitoring has become a particular area of research focus. Much of this has to do with the reality of the states of bridges nationally and what can be done in response. Ideally, all bridges identified with problems would be repaired in a timely manner. Unfortunately, statistics published by the American Society of Civil Engineers in 2007 classifies 25.6 percent of highway bridges as having as being 'structurally deficient' or 'functionally obsolete' [10]. Here, structurally deficient has to do with a bridge having a reduced load capacity while functionally obsolete deals with the geometrics of the bridge not meeting current design standards. Furthermore, the monetary deficit for the backlog of bridges deemed deficient is 32.1 billion dollars. Estimations by the ASCE predict a need for around 12.4 billion dollars annually to repair these bridges, while the federal funding is only around 4 billion per year[10][8]. As all bridges cannot be repaired under the current fiscal conditions, monitoring becomes important to the safety travelers of bridges. While bridges are inspected about every two years under federal and state regulations, continual monitoring of the structure should be established [7]. Several different types of monitoring solutions currently exist.

1.2 CURRENT BRIDGE SCOUR MONITORING SOLUTIONS

Several bridge scour monitoring techniques have been used. This section will first briefly discuss how a bridge is manually inspected for bridge scour by an inspection crew. It will then

go over the various monitoring instrumentations that have been used or proposed. These instruments fall in the categories of fixed of portable monitoring instrumentation. Tests and review of these instruments have been done by various agencies and departments. There were two main projects were initiated in the investigation of monitoring instrumentation. The first is by the Transportation Research Board (TRB) under the National Cooperative Highway Research Program (NCHRP) titled NCHRP Project 21-3 "Instrumentation for Measuring Scour at Bridge Piers and Abutments" [14]. The other is the Demonstration Project 97 (DP97) by the Federal Highway Administration (FHWA) [3]. These two projects along with attempts using monitoring instrumentation by various state departments provide a comprehensive look at what is available and working.

1.2.1 Inspection for Bridge Scour

Bridges are inspected for bridge scour in a two year cycle. Given the criteria described in HEC 18 they can be designated as being a scour critical bridge. If a bridge is scour critical it would be subject to further action such as the addition of countermeasures. Additionally, scour critical bridges would have a high priority for inspection in a flood situation.

The two main goals of an inspection team when assessing a bridge for scour are to ascertain the present condition of the bridge and identify potential for future scour problems [2]. In order to accomplish this, the inspector will take into account several factors seen at the bridge and the environment around it and apply these factors to the assessment criteria. One of the most critical steps is obtaining an underway analysis of the physical state of the environment surrounding the bridge structure. The inspector should acquire a cross section of the river or

stream bed as well as measure any scour holes seen. To do this probing, divers, as well as monitoring devices will be used [2][7].

1.2.2 Portable Monitoring Instrumentation

Portable monitoring instrumentation is used by inspection personnel to determine physical characteristics of the river bottom and scour holes. There are three main types of portable monitoring instruments used to make scour measurements [3],[4],[6],[9][13].

Physical Probes - These are rods or lead lines used to determine the depth. They are sometimes called sounding poles and sounding weights. These probes work by physically reaching the bottom bed and measuring the distance. The rods are long poles that are extended to the river bottom. Lead lines are weights connected to a cable that extend to the river bottom.

Sonar - This instrument uses sound pulses to measure depth. Given the time is takes a sound pulse to travel to the river bottom and return, a distance can be calculated. Several types of sonar have been used for scour measurement. These include simple fish finder sonar, side scan, and multi-beam sonar.

Geophysical - Geophysical instruments use wave propagations to determine distances to the river bottom. In the same manner as sonar, the elapsed time between transmission and the reception of the reflection is used to determine distance. The difference in using geophysical instruments is that several different reflections will be made based on the different materials at the river bottom. This will give a picture of the makeup of the river bottom.

1.2.3 Fixed Monitoring Instrumentation

Fixed monitoring instrumentations are devices that are installed at the bridge site to continually or intermittently make measurement related to bridge scour. There are four main types of fixed monitoring Instrumentation [3][4][9][13].

Sounding Rods - This type of device refers to a rod connected to the bridge structure with the foot of the rod resting on the surface river bottom. As the surface below the rod is scoured away the rod will slide down resulting in a measurement of distance in how far it has dropped.

Buried Rods - These devices bury a rod vertically within the target footing. The device will either have a collar attached to it that slides down as the footing drops or a sensors installed on the side of the rod that sense the removal of material around them.

Sonar – Sonar is also used as a fixed monitoring device option. It works in the same way as described for in the portable instrumentation section using acoustic reflection calculations to determine distance. In this case however, the unit is installed on the bridge.

Other Buried Devices – There are three types of other non-rod buried devices that are used to measure scour. The first and oldest is a chain buried vertically in the river bottom. After scour has been removed from the area surrounding a segment of the chain, it can be visually inspected how far down the chain scour has occurred. A newer type of device is the buried transmitter.

Here, several transmitters are buried at various locations and when scour removes the material covering them, the transmitters will float to the water surface and transmit a signal to a receiver on the bridge. Since each transmitter can send an identifier corresponding to its location, estimations can be made on bridge the amount of bridge scour encountered.

1.3 LIMITATIONS OF CURRENT SOLUTIONS

Given all the solutions presented previously, none of those is agreed upon as the best solution. The choice of the monitoring instrumentation used deals with several factors particular to the group implementing the instrument and the bridge the instrument will be used on. These factors mainly deal with cost, labor, training, accuracy, and site limitations.

Generally, portable monitoring is cheaper, can be used at multiple bridges, does not require extensive maintenance, and can provide a more complete mapping for the inspector than fixed monitoring. On the other hand, it is more labor intensive, does not provide a continuous measurement, and does require extensive training. However, there are exceptions within each group that differ from the general mold. For instance, sonar and geophysical portable methods may be extremely expensive in comparison with fixed devices depending on the quality of the sonar or geophysical device used. The trade off for this cost is the accuracy of the device [9]. Another major factor in the choice of the device is the site itself. Both categories can be limited by bridge site. The portable monitoring can be limited by the depth of the site, the velocity of the water especially during flood events, and the accumulation of debris. On the other hand it may be too hard or costly to even install fixed monitoring systems at some bridges [3] [9]. Overall, a combination of these devices must be used to properly monitor a scour critical bridge.

2.0 PROBLEM STATEMENT

The creation of a float out device to remotely sense bridge scour would give the Pennsylvania Department of Transportation another value tool in the evaluation of bridge stability. The fact that the devices to be buried are relatively small will allow for accurate estimation of scour around the bridge structure. This remote sensing system must be custom made to conform to PennDOT's requirements. It will be able to be assembled using COTS items and deployed using current PennDOT equipment. Using COTS components will reduce the overall cost of the system and allow for convenient production of the system. Designing the system for use with current PennDOT equipment will avoid the need for any change in the current infrastructure of the department. Its overall operation should be simplified such that minimal training is necessary. The system will use wireless RF signals to transfer data with proven reliability. The physical characteristics of the enclosures will also play a critical role in the system's success. Given the harsh environment the devices may endure while deployed, the physical units must be rugged and reliable. Overall, for the system to be adopted by the state a prototype must be designed that proves to be reliable, economical, and effective in its estimation of bridge scour.

The prototype remote sensing system will be designed to have three main components that work together to indicate bridge scour. The first component is a transmitter coupled with circuitry and encased such that it can be buried under the materials surrounding the bridge structure. It must be able to be armed such that it will remain dormant while buried and become

active upon its release and rise to the surface of the water. Lastly, the communication link and time it takes the unit to rise must be such that transmission of a message can consistently occur. This component in its entirety will be referred to as the **Sensor Unit** within this document.

The second component is a receiver coupled with circuitry that will be encased and installed on the bridge overpass. The receiver should be able to store and interpret the RF messages sent by the Sensor Unit. Given this interpretation it should be able to enact a response by interfacing with another unit. The receiver circuitry should have the capability to be powered continually. This component in its entirety will be referred to as the **Receiver Unit**. The third component set of lights encased with its supporting circuitry. The purpose of these lights is to correspond to a particular Sensor Unit. This will allow for immediate comprehension by the inspector. This component must be able to interface with Receiver Unit as the light that will be lit will be activated in response to Receiver Unit actions. This component in its entirety will be referred to as the **Light Indicator.**

2.1 REQUIREMENTS

2.1.1 Wireless Link Communication Frequency

Several frequency ranges are available for use in this system, including 433 MHz and 915 MHz. Lower frequencies, such as 125 kHz and 13.56 MHz, have a shorter range with smaller antenna dimensions and configurations compared to the 433 MHz and 915 MHz based systems. The 915 MHz based systems are more susceptible to interference from debris (soil, branches,

etc.) that may be present during a flood. Commercial off the shelf (COTS) devices exist for both the 433 MHz and 915 MHz based systems.

The 433 MHz based systems are less susceptible to interference from debris because the lower frequency can be expected to more easily penetrate such debris and is compatible with typical sensor dimensions. For these reasons, the 433 MHz based solution is required.

2.1.2 Sensor Unit Capsule Dimensions and Tethering Option

Given any frequency and transmitter/receiver link properties there will be a limit on the range data can be reliably transmitted. As a result, several options will have to be weighed in the choices related to the Sensor Unit capsule and whether or not to tether the Sensor Unit. Sensor Unit capsule size and material are the main factors in how quickly the Sensor Unit rises to the water surface where it can transmit to the Receiver Unit. The system requires that the Sensor Unit capsule be such that it rises within transmission range. Increasing the capsule's volume is the most direct way to increase the speed at which it rises to the surface. However, the Sensor Unit should not become overly large such that it cannot be installed using 3.3/16 hollow stem auger drill used by PennDOT. Additionally, if the capsule is made too long, the Sensor Unit loses the precision of its location and the extent to which it can identify scour. If all capsule configurations prove infeasible, a tether may have to be attached in order to keep the Sensor Unit within range. The tether would be anchored at the point where the Sensor Unit is buried. The drawback of this option is that it introduces an additional failure point and complicates installation. An analysis and decision will be required on this topic.

2.1.3 Data Requirements

The Sensor Unit must contain information to identify its location and depth as well as a bridge identification to link it to the bridge at which it is deployed. Storing the bridge identification enables the receiver units to distinguish between sensors belonging to the bridge they are mounted on and those sensors from other bridges that will follow the stream flow. This feature will also provide the capability to link multiple receiver units together and transfer information among them, allowing a receiver unit on one bridge to potentially pick up a sensor from another bridge and report that information to the other bridge or a centralized location.

The following data items will be stored in non-volatile memory on the sensor unit.

1. Bridge Identification Number (BMS)
2. Depth
3. Location of Sensor #

The information in these three fields will be coordinated with the receiver unit on the bridge during installation. The Receiver Unit needs to relate each location and depth with a particular action indicating the level of bridge scour. The Receiver Unit should be updated each time a new sensor unit is deployed.

The three fields mentioned above will be stored in non-volatile memory as groups of bytes. Each data field will be stored in a specific location in the Sensor Unit non-volatile memory. This data will be transmitted to the Receiver Unit once the Sensor Unit is activated by its release from the soil. Error detection information such as a CRC or checksum will be added to the message as a whole. Adding the error detection information will increase the amount of data to transmit and thus, increases, the time for one transmission. The CRC offers the more robust error detection capability than the checksum option. Therefore a CRC will be computed beforehand and stored

in non-volatile memory at the same time the information above is being written to the Sensor Unit.

The Bridge Identification Number will need to be transmitted to the Receiver Unit because each bridge may have multiple Receiver Units and a sensor may be heard by multiple receivers. Including the Bridge Identification Number in the message will prevent a sensor unit assigned to one Receiver Unit from triggering an alert on another Receiver Unit. The Sensor Unit will be designed so that it can hold all three values in non-volatile memory, and the set of data items to transmit can be easily changed in software.

2.1.4 System Interfacing and Programming

The Sensor and Receiver Unit will be communicated with by a technician in two ways. First, the units will have to be programmed by some means. Second, data should be able to be downloaded from and to the units easily, as well as reconfigure basic settings. The connection from some host PC used to do this should be as simple as possible and require a standard connection means i.e. USB connection or parallel port. Here a USB connection would be preferred as it is typically present on all computing machines. Also, simple interfaces to both download the program code and transfer program data needs to be developed.

2.1.5 Power Considerations

The Sensor Unit and Receiver Unit each have particular power requirements. The Light Indicator will be connected to a Receiver Unit and thus will depend on it for a power source. The Sensor Unit will require batteries as its power source. The batteries used by the Sensor Unit

will not have to transmit for an extended amount of time. The batteries need to support the functioning of the Sensor Unit long enough that a message can be transmitted, preferably several times. However, the Receiver Unit only needs to have one of these messages sent without error and received correctly. Given the speed of current microcontroller, this would require seconds but the unit should be capable of transmitting for minutes to be safe. The battery should also have the available current to support all Sensor Unit circuitry operating simultaneously. Lastly, the Sensor Unit will be required to remain buried for several years. As such, the battery technology chosen should have as long a shelf life in a dormant state. The Receiver Unit will have several options for its power source as it will be installed above the surface of the water. Consequently, it should be designed to accept any available AC or DC power source.

2.2 PROBLEM AREAS

In order to fulfill the requirements of the system the following problem areas were dealt with within this thesis:

- Overall System Design
- Sensor Design
  - Orientation Sensitive
  - Locking Switch
- Receiver Design
- Light Indicator Design
- Software Design
- Buoyancy Analysis
- Drag Analysis
- Collision Avoidance
- Communication
  - JTAG
  - Serial
  - Protocol
- Testing Methodology
  - RF Strength
  - Components
  - Watertight Capsule
  - Rise Time
  - Single Sensor
  - Multiple Sensor

3.0 SYSTEM SPECIFICATIONS

The system will consist of three types of devices, (1) Sensor Units, (2) Receiver Units, and (3) Light Indicators. The Sensor Units will be buried in the riverbed or streambed and will remain dormant until they float away as a result of a scour event. Each installation (bridge site) will have at least one, but potentially multiple Sensor Units. The Receiver Unit will be mounted on the bridge. The Receiver Unit will listen for Sensor Unit messages indicating scour and will operate a Light Indicator (green, yellow, orange, and red lights) in response to detected scour events. Typically a bridge will have a single Receiver Unit, but multiple Receiver Units per bridge are possible. The Light Indicator component will be designed to handle input from multiple Receiver Units.

Figure 1 shown below illustrates the basic idea of the system given a tether option or non-tethered option. On the left hand side, the tethered option is shown with a possible solution for the anchor and tether. One solution for the tether option is to coil the tether, in this case a wire, around a rod that is attached to the anchor. The wire will unravel as the Sensor Unit rises and this should reduce the risk of the wire getting hung up during both installation and operation. This causes problems if multiple Sensor Units are buried at the same location, i.e. above other Sensor Units at lower depths. The right hand side illustrates the un-tethered option, which is simpler and is not affected by the problem of the tether wire getting caught in debris or soil.

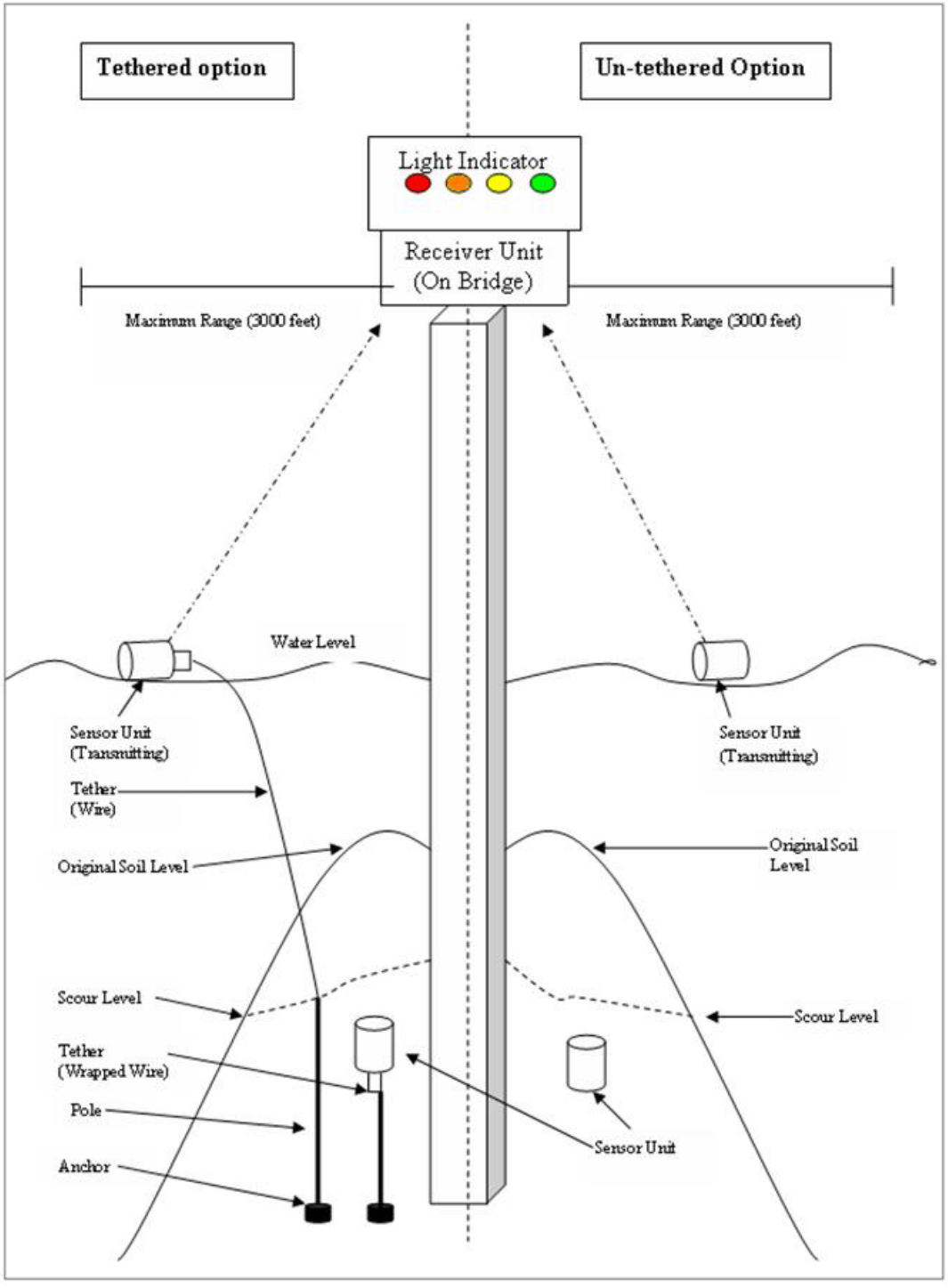


Figure 1: System diagram with tethered option on the left and un-tethered option on the right

Figure 2 below shows the high-level operation of the overall system. The general operation, including installation, of the system is explained in this diagram. The indication lights will only transition to a more severe state in response to a message from a Sensor Unit, i.e. no transition from red to green is possible. To clear a light state, the *Manual Reset* on the Receiver Unit must be used. The Receiver Unit will incorporate the Manual Reset and provide protection to allow only PennDOT personnel to activate it, i.e., switch will be in a locked box or enclosure, PennDOT personnel would have the key. The specific processing flow for both the Sensor Unit and Receiver Unit are discussed in detail in following sections.

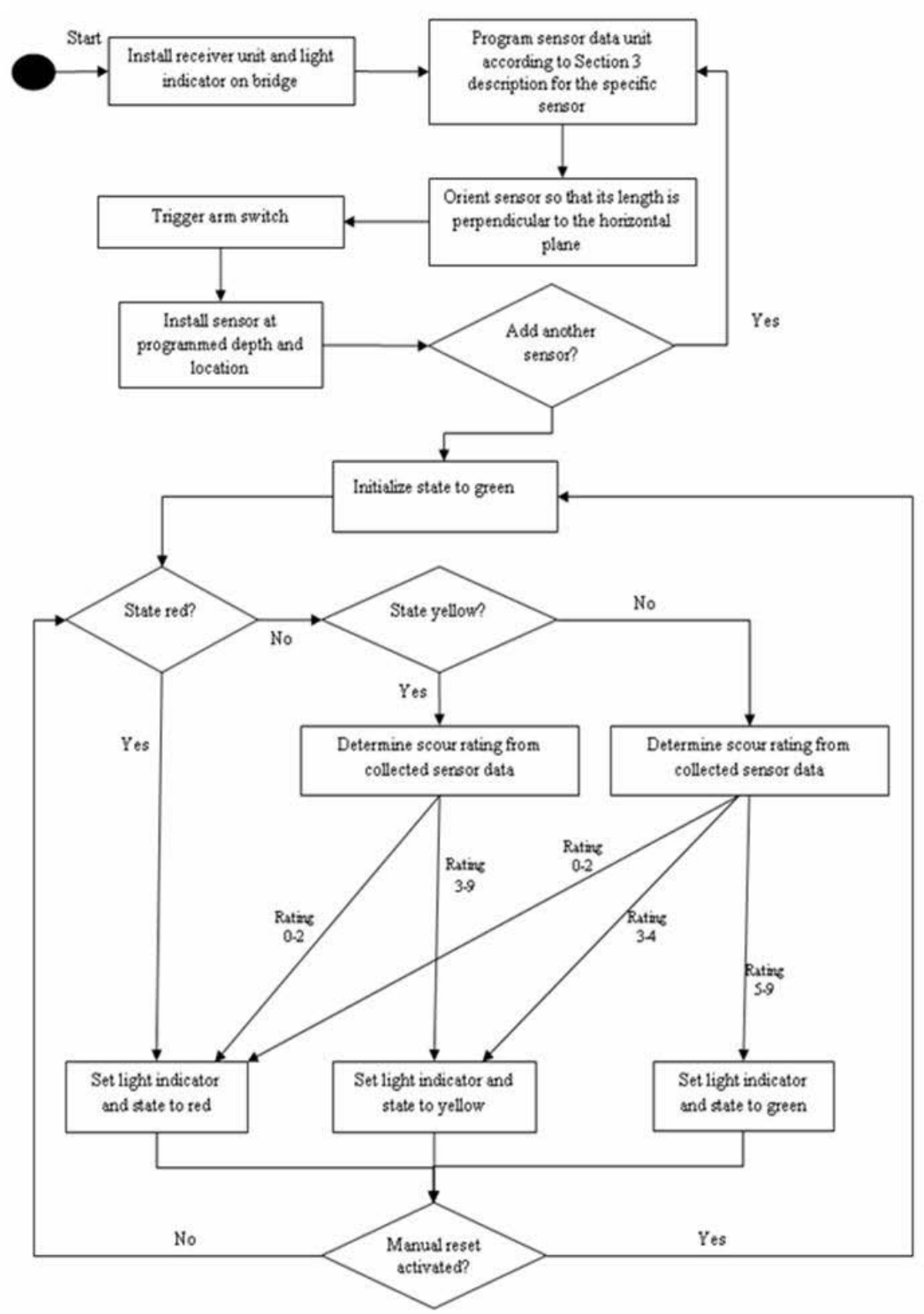


Figure 2: High-level system flow diagram.

3.1 SENSOR UNIT SPECIFICATIONS

The specifications for the Sensor Unit device are presented in this section. The Sensor Unit will be buried in the stream or river bed. During a scour event the Sensor Unit will float to the surface of the water once the material covering the Sensor Unit has been removed by the scour. The Sensor Unit will repeatedly transmit a message to the Receiver Unit. The Sensor Unit will provide information as to the bridge it monitors, the depth it was buried at, the nearest bridge structure ID, and the color code of the Sensor Unit.

3.1.1 Senor Unit Hardware Architecture

This sub-section presents the hardware architecture of the Sensor Unit. The figure below provides a top-level block diagram of the architecture of the system. This includes all major components for the Sensor Unit. Specifications for each of the major components in Figure 3 are presented in this section.

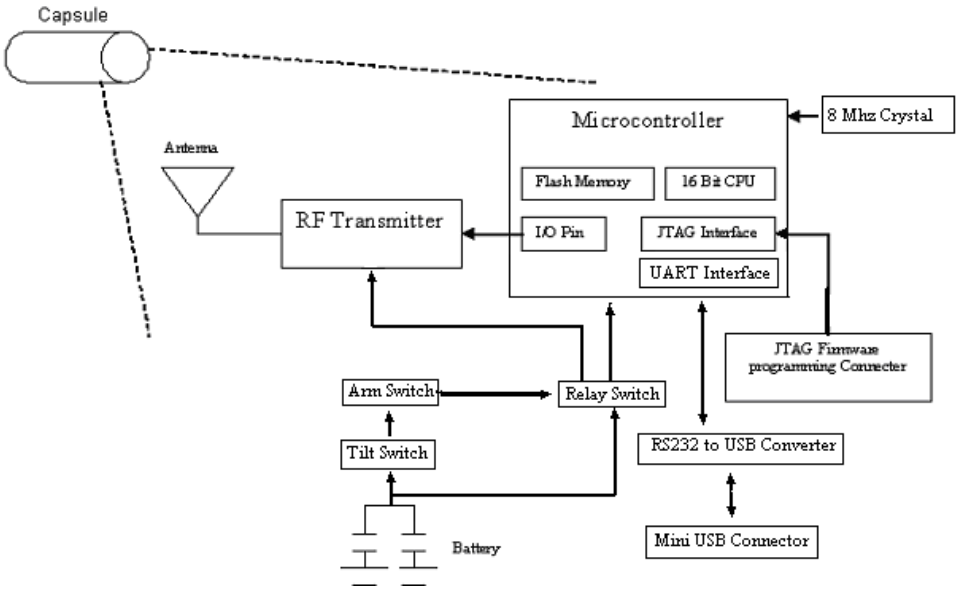


Figure 3: Top-level block diagram of the Sensor Unit

Microcontroller: The microcontroller will control the operation of the Sensor Unit. The microcontroller must wake-up after being triggered by tilt switches and transmit the Sensor Unit Data Block to the Receiver Unit. Thus, the microcontroller must read the Sensor Unit Data Block from non-volatile memory, setup and initialize the RF transmitter, and interface with the RF transmitter to send the Sensor Unit Data Block to the Receiver Unit. The microcontroller will store the program code (software) and the Sensor Unit Data Block in on-board non-volatile memory. The software will be written in *C* or *assembly language*. The microcontroller will be programmed through the JTAG interface.

RF Transmitter: A commercial off the shelf (COTS) RF transmitter transmitting at 433 MHz will be used. The microcontroller will interface with the RF transmitter and will control the operation of the transmitter. The RF transmitter will be capable of transmitting at a data-rate of 10 Kbps (kilo-bits per second) and have a range of 3000 ft. The RF transmitter will use on-off

keying (OOK) to modulate the data stream from the microcontroller. The RF transmitter will connect to a COTS 50 Ohm antenna.

Relay Switch: A relay switch will be used to form a permanent connection between the battery supply and the active components of the device. The need for a relay switch is due to the use of a tilt switch to active the device. This tilt switch however cannot be relied upon to continually provide a steady connection as the orientation of the Sensor Unit will most likely be in a state of continual change. Once the tilt switch forms a closed circuit, the relay switch will form a permanent connection.

Arm Switch: The arm switch will connect the tilt switch to the power supply (battery). When in the off position, the arm switch will act as an open circuit, and when in the on position, the arm switch will act as a short circuit, connecting the tilt switch to the battery. The arm switch will be a COTS device that will provide for easy activation but is resistant to inadvertently being switch to the off position, i.e., because of turbulence.

Tilt Switches: The tilt switches provide a connection between the arm switch and the battery supply. Each tilt switch is composed of two leads that can connect to a common electrode based on the orientation of the tilt switch. At 35 degrees from the horizon in either direction, the common electrode connects to one of the leads and the tilt switch acts as a short circuit. When the tilt switch is less than 35 degrees from the horizon in either direction the tilt switch acts as an open circuit. The Sensor Unit will be designed so that when the Sensor Unit reaches the surface,

one of the Tilt Switches will be orientated to be at least 35 degrees from the horizon and thus act as a short circuit.

JTAG Connection: The JTAG Connection will provide the means to transfer (write) the software and data to the microprocessor's memory as well as debug the system while in the testing phase.

RS232 to USB Converter: A COTS chip will be used to convert the serial RS232 communication used by the microcontroller to USB communication.

Mini USB Connection: The mini USB connection will be used to interface the device with a USB connection of a host PC.

Battery: The battery will provide power to the Sensor Unit. The battery will be a COTS and will provide 3.0 V.

Antenna: The Antenna will be connected to the output of the RF transmitter. The antenna will be a COTS device that is designed for 50 Ohm micro strip. The antenna will conform to the form factor and size of the Sensor Unit.

Capsule: The capsule will be of a cylinder shape and will be watertight. The capsule will be buoyant and will be made of sufficient material to prevent damage once activated (floating). The

electronics will be contained within the capsule. The arm switch will be accessible from the outside of the capsule.

3.1.2 Sensor Unit Data Block

This section presents a detailed description of how the data items will be stored in the Sensor Unit non-volatile memory. This is the data that will be transmitted to the Receiver Unit. These items are in accordance with PennDOT labeling practices. The first three data items are:

1. Bridge Identification Number (BMS)
2. Serial Number
3. Location of Sensor
    - 3.0 Nearest Structure Unit ID – item 5D02 of BMS2 [15]
    - 3.1 Color Code – this will be used to determine if bridge is in need of inspection or should be closed

The 14-digit Structure ID number that PennDOT uses for the BMS2 system (BMS2 item 5A01) will be used as the format of the Bridge Identification Number [15]. The Bridge Identification Number will be designed to hold 14 numerical (1 byte per numerical digit) to accommodate the 14-digit Structure ID number.

The Serial Number will be a unique identifier for each Sensor Unit stored at a particular bridge location. The Serial Number will be stored in a 2-byte field. The range of possible numbers is from 0 to 65536.

The Location of Sensor field will consist of one data item used by the BMS2 system; the ID of the nearest structural unit (BMS2 item 5D02); and Color Code. The ID of the nearest structural unit can be used to provide inspectors an indication of where scour was detected. The nearest structure unit ID sub-field will be 4-bytes and will store four numerical characters. The Color

Code will be used to indicate the criticality of the scour risk the Sensor Unit is monitoring and will simplify the Receiver Unit software. Thus, the Color Code will depend on the depth the Sensor Unit is buried at and the location of the Sensor Unit. Where each color is buried at will be specific to each bridge and should be determined by PennDOT staff. The Color Code sub-field will be 1-byte and will initially represent 4 different colors. This can be expanded to up to 256 colors if necessary. The Location of Sensor field will require 5-bytes and will store five numerical characters. The first 4-bytes (bytes 1 to 4) will contain the nearest structure unit ID sub-field and the last byte (byte 5) will store the Color Code sub-field. The following table shows the breakdown of the Location of Sensor field with sub-field lengths and addresses.

Table 1: Breakdown of the Location of Sensor Field

Location of Sensor Field					
Byte	1	2	3	4	5
Offset[1]	0	1	2	3	4
Sub-Field	**Nearest Structure Unit ID**				**Color Code**

The Color Code sub-field will be used by the receiver unit to determine what action to take, i.e., the LED indicator setting (green, yellow, orange, or red), when a Sensor Unit is detected (released from stream or river bed). The Sensor Unit will utilize the Color Code field to indicate the severity of the scour it is monitoring, i.e., the severity of scour hole to the depth the Sensor Unit was buried. The Receiver Unit will interpret the Color Code as defined in the following table.

[1] Byte offset from start of the Location of Sensor field.

Table 2: Interpretation and action resulting from the color code sub-field by the Receiver Unit.

Color Code	Integer Representation	Receiver Unit Interpretation	Receiver Unit Action
Green	0	No Scour	Indicator: Green
Yellow	1	Variable	Indicator: Yellow
Orange	2	Variable	Indicator: Orange
Red	3	Severe Scour	Indicator: Red

The entire message will have a 16-bit CRC appended to it for error detection. The CRC will enable the Receiver Unit to detect and then discard any message containing errors. Adding the CRC will add two bytes to the length of the message (data packet). The CRC will be computed as follows:

- CCITT polynomial ($x^{16} + x^{12} + x^5 + 1$) will be used[2]
- The computation will be initialized with zeros (0x0000)

The data will be stored in the non-volatile memory of the Sensor Unit at specific locations. The data can start at any valid memory location provided that there is enough contiguous memory bytes to hold the entire data block. Otherwise the start of data must be pushed back to a lower memory location. The data block will consist of 4 fields with the Location of Sensor field containing two sub-fields,

1. Bridge Identification Number (BMS)
2. Serial Number
3. Location of Sensor
    a. Nearest Structure Unit ID – item 5D02 of BMS2 [15]

[2] CCITT polynomial reference: ITU-T Recommendation V.41 (Blue Book)

b. Color Code – this will be used to determine if bridge is in need of inspection or should be closed

4. CRC-16 (error detection mechanism)

The length, in bytes, of the Sensor Unit data block is 23 bytes. The following table lists the lengths, in bytes, of each field in the data block.

Table 3: Length of the data fields stored in the Sensor Unit data block.

Field Name	**Number of Bytes**
Bridge Identification Number	14 Bytes
Serial Number	2 Bytes
Location of Sensor	5 Bytes
CRC-16	2 Bytes

The following table shows the layout of the Sensor Unit Data Block.

Table 4: Sensor Unit Data Block

Byte	1	2	3	4	5	6	7	8
Offset	0	1	2	3	4	5	6	7
Field	Bridge Identification Number							
Sub-Field	N/A							
Byte	9	10	11	12	13	14	15	16
Offset	8	9	10	11	12	13	14	15
Field	Bridge Identification Number						Serial Number	
Sub-Field	N/A						N/A	
Byte	17	18	19	20	21	22	23	
Offset	16	17	18	19	20	21	22	
Field	Location of Sensor					CRC-16		
Sub-Field	Nearest Structure Unit ID				Color Code	N/A		

3.1.3 Sensor Unit Software Architecture

The software (program) will control the operation of the Sensor Unit. The software will (1) setup and initialize the RF transmitter; (2) retrieve the Sensor Unit Data Block from non-volatile memory; and (3) interface with the RF transmitter to transmit the Sensor Unit Data Block to the Receiver Unit.

The setup and initialization of the RF transmitter is a simple process of asserting a few signals. The software will read the Sensor Unit Data Block from non-volatile memory and will convert that into the appropriate format expected by the RF transmitter. This includes calculating a 2 byte CRC. The software will then output a synchronizing message and/or preamble to the RF transmitter, which precedes each message and informs the Receiver Unit that a message is present. Next, the software will output the contents of the Sensor Unit Data Block

to the RF transmitter. The software will send the preamble and message, repeatedly, until the Sensor Unit is deactivated or losses power (battery dies). Figure 4 shows the high-level flow chart for the Sensor Unit software.

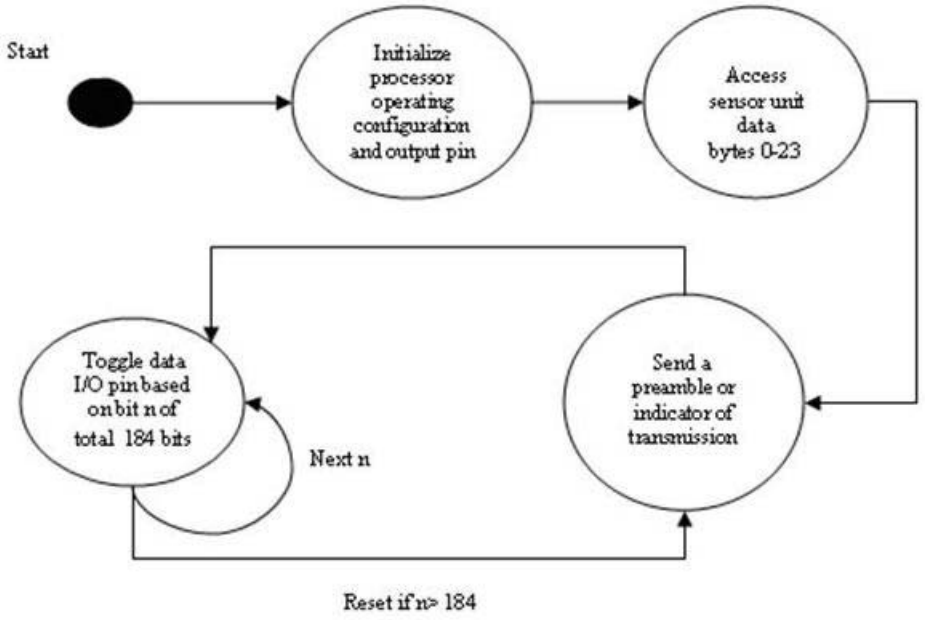


Figure 4: Sensor Unit software flow chart.

3.1.4 Receiver Unit Hardware Architecture

The Receiver Unit will be mounted on the bridge structure and will control a Light Indicator signifying if the bridge is safe to cross or not. The Receiver Unit will (1) receive message from Sensor Units; (2) control the Light Indicator; and (3) accept input through UART communication. The UART communication will allow PennDOT personnel to program the unit's bridge ID, clear a green, yellow, orange, or red condition (scour has been detected), and

read out data collected. Figure 5 shows the block diagram of the Receiver Unit, each of the main components are described in this section.

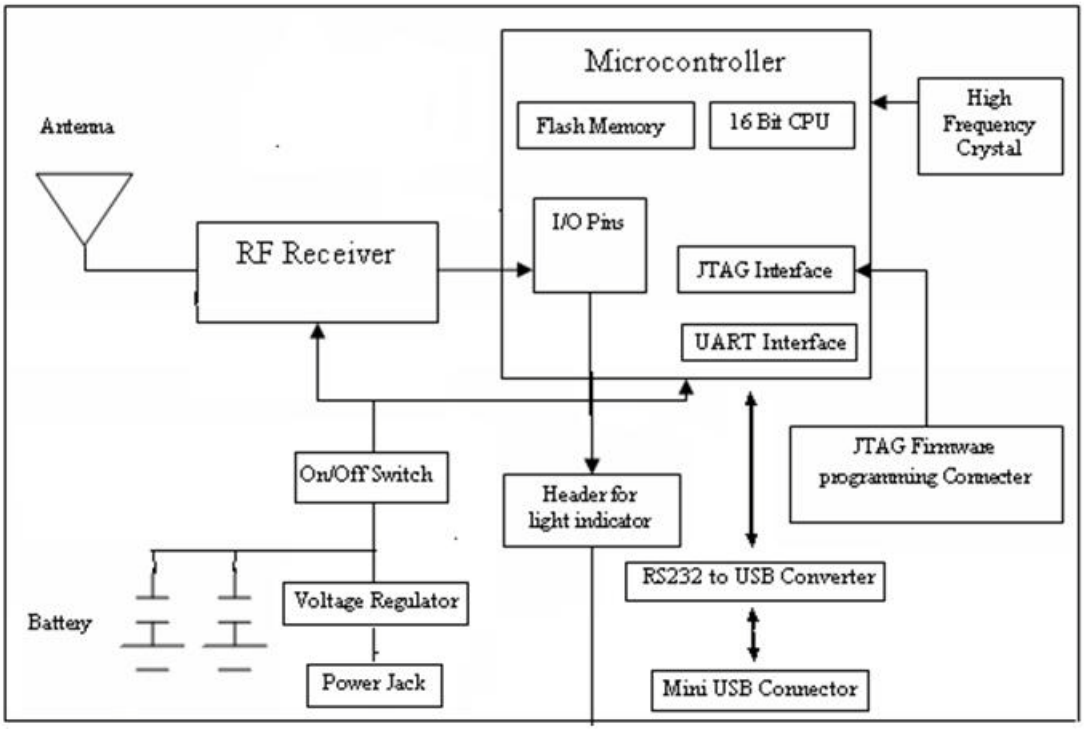


Figure 5: Receiver Unit block diagram.

Microcontroller: The microcontroller will contain the software program to interface with and control the other blocks that make up the Receiver Unit. The microcontroller will interface with and control the RF Receiver, and will monitor the data output line of the RF Receiver for messages from Sensor Units. The data from the RF Receiver will be a serial string of bits and will be input to a single data I/O pin on the microcontroller. The microcontroller will update the state of the Light Indicator based on the data in the Sensor Unit message. The microcontroller will use four output pins to control the four light system. The microcontroller will accept input from UART communication, which is used to clear the Light Indicator condition. The software

program will be stored in non-volatile memory on-board the microcontroller. The software will be written in *C* or *assembly language*. The microcontroller will be programmed through the JTAG interface.

On/Off Switch: The On/Off switch will be a switch or button that will connect or disconnect the Receiver Unit to power. The On/Off switch will be physically protected, i.e. enclosed in a locked box, from unauthorized use. Only PennDOT personnel will have access to the On/Off switch.

RF Receiver: The RF Receiver will receive and decode data transmitted by the Sensor Unit. The RF Receiver will be a COTS device that is compatible with the RF transmitter used in the Sensor Unit. The RF Receiver will demodulate an OOK signal. The RF Receiver will connect to a 50 Ohm antenna.

JTAG Connection: The JTAG Connection will provide the means to transfer (write) the software and data to the microprocessor's memory.

RS232 to USB Converter: A COTS chip will be used to convert the serial RS232 communication used by the microcontroller to USB communication.

Mini USB Connection: The mini USB connection will be used to interface the device with a USB connection of a host PC.

Header for Light Indicator: The header for the light indicator will be a six pin header connecting the four microcontroller I/O pins assigned to the Light Indicator, a power, and ground connection. Any signal conditioning required in converting between operating levels of the microcontroller and the traffic light will be done in this block.

Battery: The battery will provide power to the Sensor Unit. The battery will be a COTS device and will provide 3.0 V.

Antenna: The Antenna will be connected to the output of the RF Transmitter. The antenna will be a COTS device that is designed for 50 Ohms. The antenna will conform to the form factor and size of the Receiver Unit.

Light Indicator: The Light Indicator will contain four lights of red, orange, yellow, and green. The red light will indicate that the bridge is not safe to cross. The yellow and orange lights will indicate that the bridge may be safe to cross with caution, and a green light or no light will indicate that the bridge is safe to cross. For initial prototyping purposes, a simple circuit of four light emitting diodes (LEDs) with the colors of red, orange, yellow, and green will be used.

3.1.5 Receiver Unit Software Architecture

The Receiver Unit software will control the operation of the Receiver Unit. The software (1) will setup and control the interface with the RF Receiver; (2) will monitor the data output of the

RF Receiver for Sensor Unit preamble and message; (3) decode Sensor Unit messages; (4) parse the Sensor Unit message and update the Light Indicator to reflect the scour severity reported in the Sensor Unit message; (5) will take input from the UART and perform the required action such as resetting the system. The software will verify that the Sensor Unit Data Block was received without error by computing the CRC. The software will use the Color Code of the Sensor Unit Data Block to determine what condition to set the system to. The software will discard all messages containing errors (invalid CRC). The software will not set the Light Indicator to a less severe condition in response to a Sensor Unit message. Thus, the Light Indicator may change from green to yellow, orange, or red, or from yellow to orange or red..etc, but not from yellow to green or orange to yellow or green..etc. Figure 6 shows the high-level flow of the Receiver Unit software.

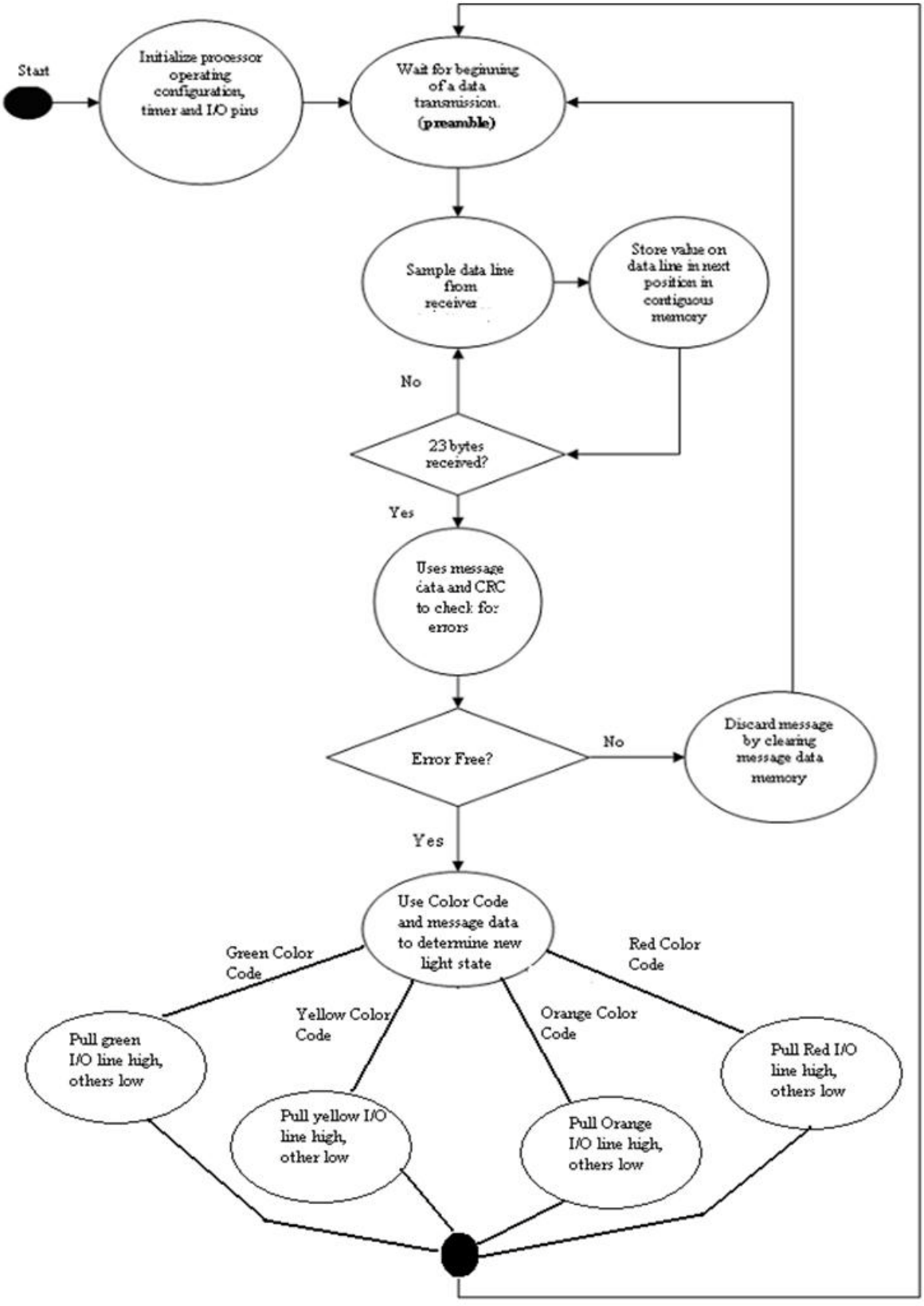


Figure 6: Flow chart of the Receiver Unit software.

3.1.6 Light Indicator Hardware Architecture

This sub-section presents the hardware architecture of the Light Indicator. The figure below provides a top-level block diagram of the architecture. This includes all major components for the Light Indicator. Specifications for each of the major components in Figure 7 are presented in this section.

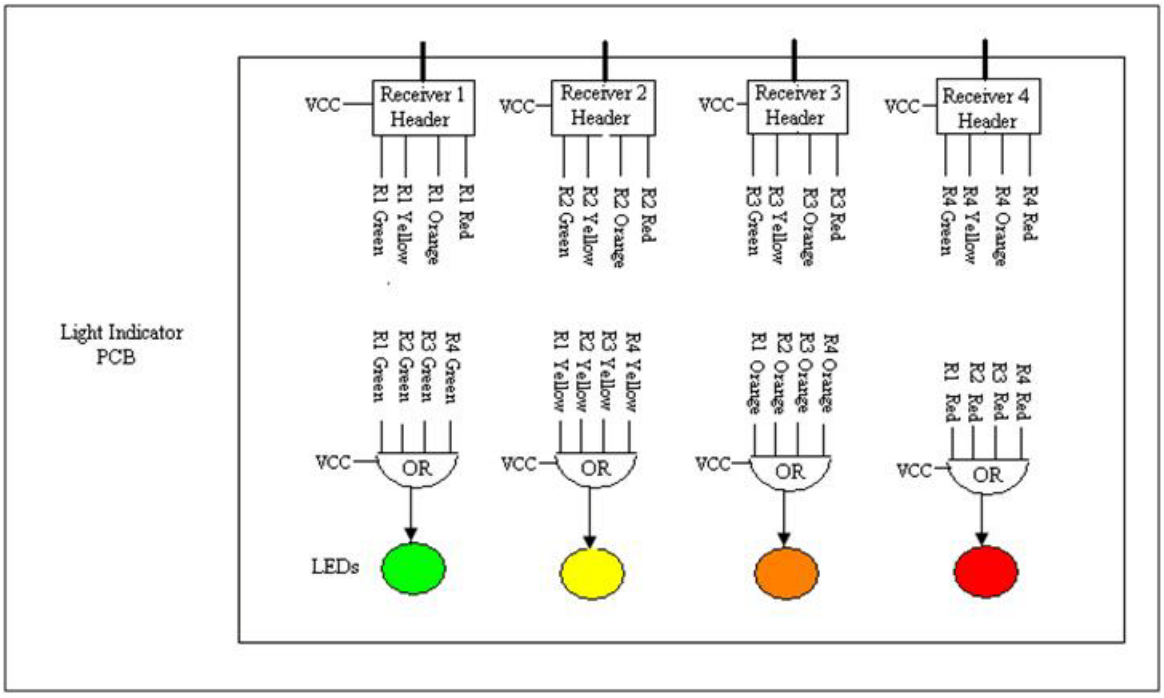


Figure 7: Light Indicator Hardware Diagram

6 Pin Header: Four 6 pin headers are located on the Light Indicator. Each can take input from a separate Receiver Unit. The case of multiple Receiver Units would only occur if redundancy is desired. Of the six pins on each header, four are to correspond to a LED each, one is to connect to the power of the Receiver Unit, and one is to connect to the ground of the Receiver Unit.

OR Gate: OR gates will be used to drive the individual LEDs. Each OR gate will connect to the inputs on the 6 pin header corresponding to LED it is driving. Each OR gate will OR four inputs.

LEDs: Four LEDs will be used on the Light Indicator. These LEDs are of the colors green, yellow, orange, and red.

4.0 HARDWARE DESIGN

This section will provide detailed documentation of the design of Sensor Unit, Receiver Unit, and Light Indicator. The following subsection will look at the design of each of these main components individually. For each of these components, all information regarding its design will be described. This includes explanation of parts and their interfaces as well as any schematics and design files created.

4.1 SENSOR UNIT DESIGN

This section contains the design of the Sensor Unit. The first subsection contains a description of the operational functionality required by the system. The second subsection partitions the Sensor Unit into a set of functional blocks. The third subsection provides a detailed description of the interfaces between the functional blocks presented in the second subsection.

4.1.1 Sensor Unit Description of Operation

The first operational requirement of the Sensor Unit is the ability of the Sensor Unit to be *armed* upon installation. When the Sensor Unit is *armed*, the tilt switch controls the connection to the power supply (battery) of the Sensor Unit. Hence, when the Sensor Unit is *armed* and the tilt

switch is asserted, the relay switch will be triggered connecting the unit's parts to power. The Sensor Unit will then become active and will transmit the Sensor Unit Data Block. Conversely, when the Sensor Unit is *not armed* there is no path from the unit's parts to the power supply (battery) of the Sensor Unit. Hence, when it is *not armed* and the tilt switch is asserted the Sensor Unit remains inactive and does not transmit any data. The Sensor Unit will be armed immediately before it is placed into the hollow-stemmed auger. This requires the use of the arming switch located externally on the Sensor Unit. Prior to the arming of the Sensor Unit, the circuitry within the Sensor Unit should not be active and the Sensor Unit will not transmit under any circumstances. Upon installation, the arming switch will set the circuit to a state where its activation can be triggered at the correct orientation.

Once armed, the Sensor Unit will be inactive while in a vertical orientation and active while in a horizontal or *tilted* from vertical orientation. Vertical and horizontal/tilted orientation in this context means the Sensor Unit length (end cap to end cap) will be perpendicular to ground and water surface (vertical orientation) or parallel to the ground and water surface (horizontal orientation). The actuation of this orientation sensitive activity is realized through the use of three tilt switches. The switches will be unconnected when in the Sensor Unit is in a vertical and buried state as shown in Figure 8. All switches will form a 55-degree angle with the horizon in this state. In the horizontal orientation one of the tilt switches should become active. The three tilt switches are positioned such that when the face of the PCB (the where the majority of the components are located) is furthest down or either side of the PCB is furthest down, one of the tilt switches will form a 35 degree angle below the horizon. This will cause one of the tilt switches to become active. The activation points for each tilt switch are shown in Figure 9, Figure 10, and Figure 11. The tilt switch activated is shown in red. All angles in between these

three positions would also result in the triggering of a tilt switch as well. Given the turbulent environment the Sensor Unit could be introduced to, this increases the certainty of the activation of the Sensor Unit.

The design is also based on the observed orientation the Sensor Unit floats in due to the weight of the PCB and its components. The Sensor Unit has a tendency towards the orientation in Figure 11 and turbulent waters would only cause a momentary divergence from this orientation. The position shown in Figure 11 is the position the Sensor Unit normally floats in. It can be considered to add a fourth tilt switch to cover the last axis of orientation, however it is very unlikely the sensor would be oriented in this position for any extended period of time.

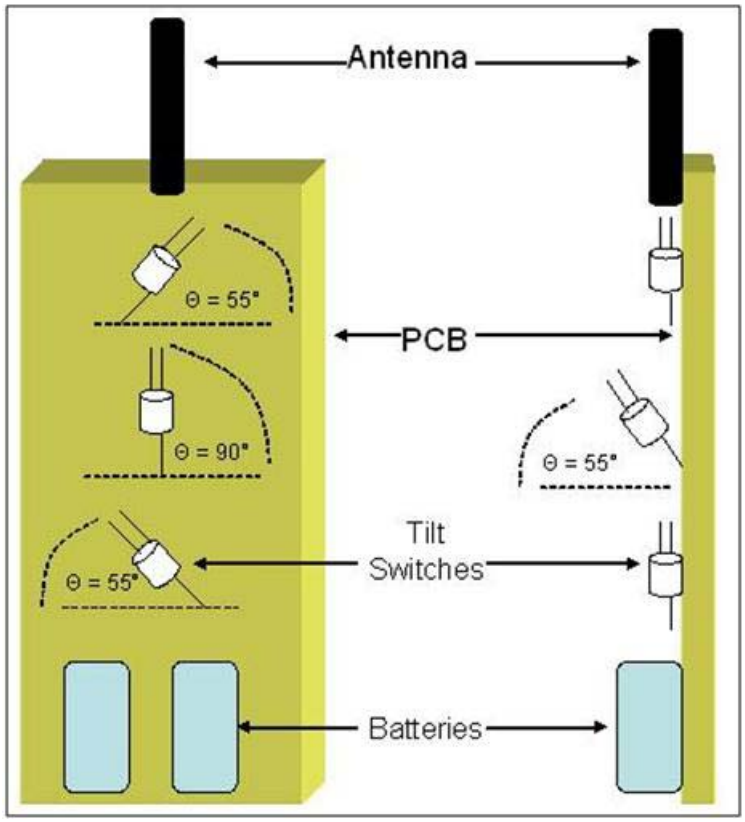


Figure 8: Tilt Switch Diagram

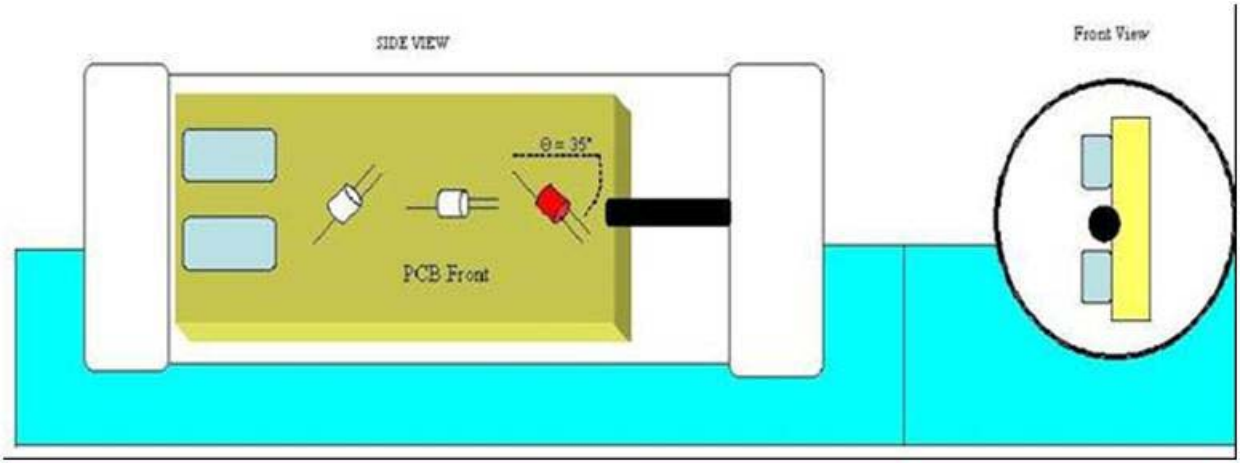


Figure 9: Activation position of first Tilt Switch

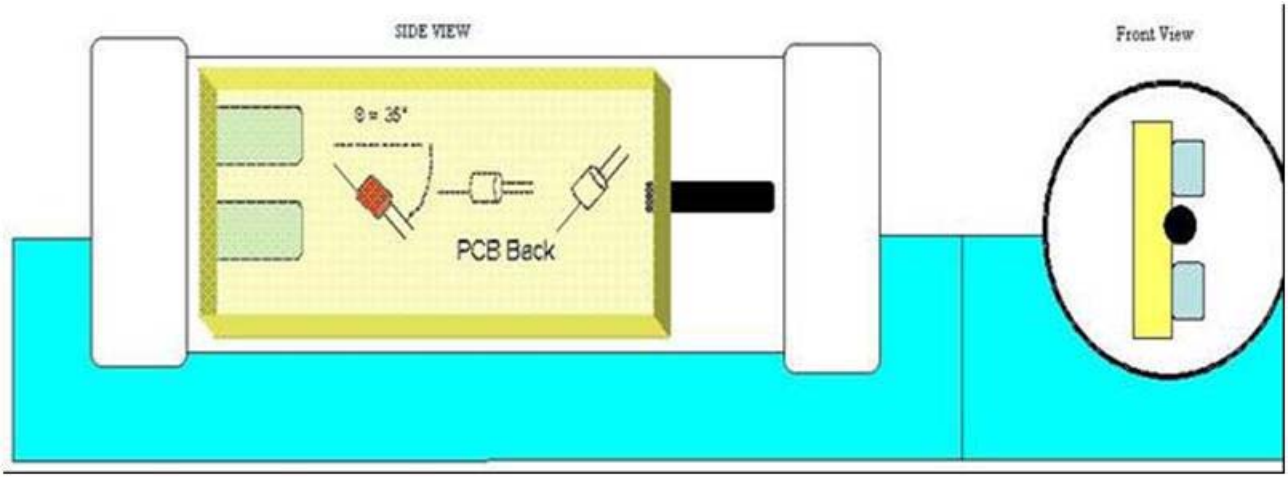


Figure 10: Activation position of second Tilt Switch

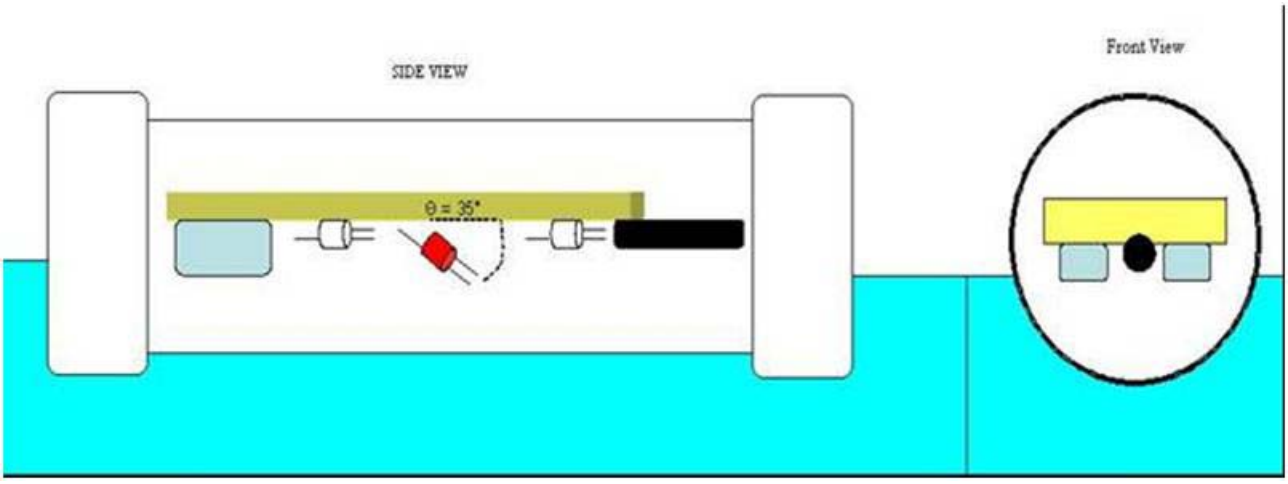


Figure 11: Activation position of third Tilt Switch (Naturally oriented position)

If a tilt switch becomes active, the relay switch will be polarized such that it connects the batteries to the power rail (V_{dd}). At this point, the microcontroller will start and the program stored in the on-board flash memory will be executed. The basic operation of the program is to transmit a preamble and the data unit containing all data pieces outlined in the description the Sensor Unit Data Block in a repeated sequence. The Sensor Unit will transmit this repeated sequence until the batteries are drained or the unit is retrieved. Calculations indicate that in most cases, the Sensor Unit will be well out of range when the batteries die.

4.1.2 Breakdown of Senor Unit into Functional Blocks

This section partitions the Sensor Unit into a set of basic functional blocks. Figure 12 shows the functional blocks making up the Sensor Unit. A detailed description of each block is presented in the following subsections.

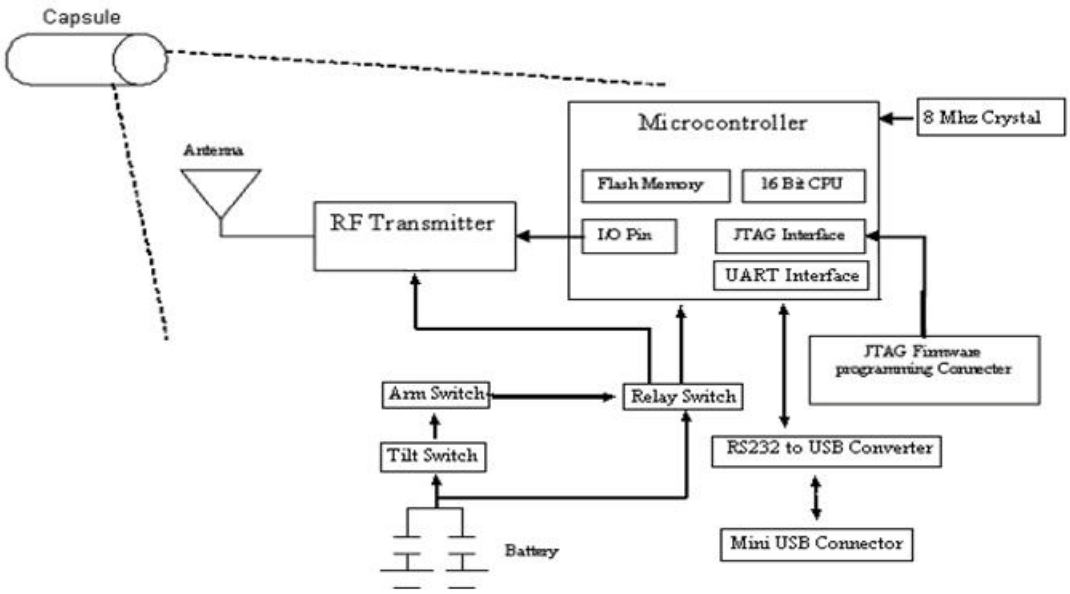


Figure 12: Sensor Block Diagram

Microcontroller

The microcontroller, with a block diagram shown in Figure 13, will control the operation of the Sensor Unit. The microcontroller to be used is the MSP430F2132 made by Texas Instruments [16]. This processor has a 28 pin, 16-bit RISC architecture CPU. The chip package is Thin-Shrink Small Outline Package, which is a four sided surface mount chip package. The Microcontroller is capable of operating at clock speeds up to 16 MHz. The clock speed used will be 8 MHz given the 3.0 V power supply. The processor memory is non-volatile flash memory. It contains 8 KB of flash program memory and 256 bytes of information (user) flash memory. The fundamental difference between the two flash memories is in how they are segmented and how they are erased. The information memory and main memories can be erased separately and the information memory can be erased in smaller segments. The main memory will contain the program code (software) and the information memory will contain the Sensor Unit Data Block. The microcontroller has 24 general input/output (I/O) pins. Of the 24 general I/O pins, only 1

will be used for I/O. Lastly, the microcontroller has a built in JTAG/BSL interface through which the microcontroller will be programmed. This will require 6 of the I/O pins.

The microcontroller must wake-up after being triggered by the tilt switch and transmit the Sensor Unit Data Block to the Receiver Unit. Thus, the microcontroller must read the Sensor Unit Data Block from non-volatile memory and interface with the RF transmitter to send the Sensor Unit Data Block to the Receiver Unit. The microcontroller will store the program code (software) and the Sensor Unit Data Block in on-board non-volatile memory. The software will be written in *C*. The microcontroller will be programmed through the JTAG/BSL interface.

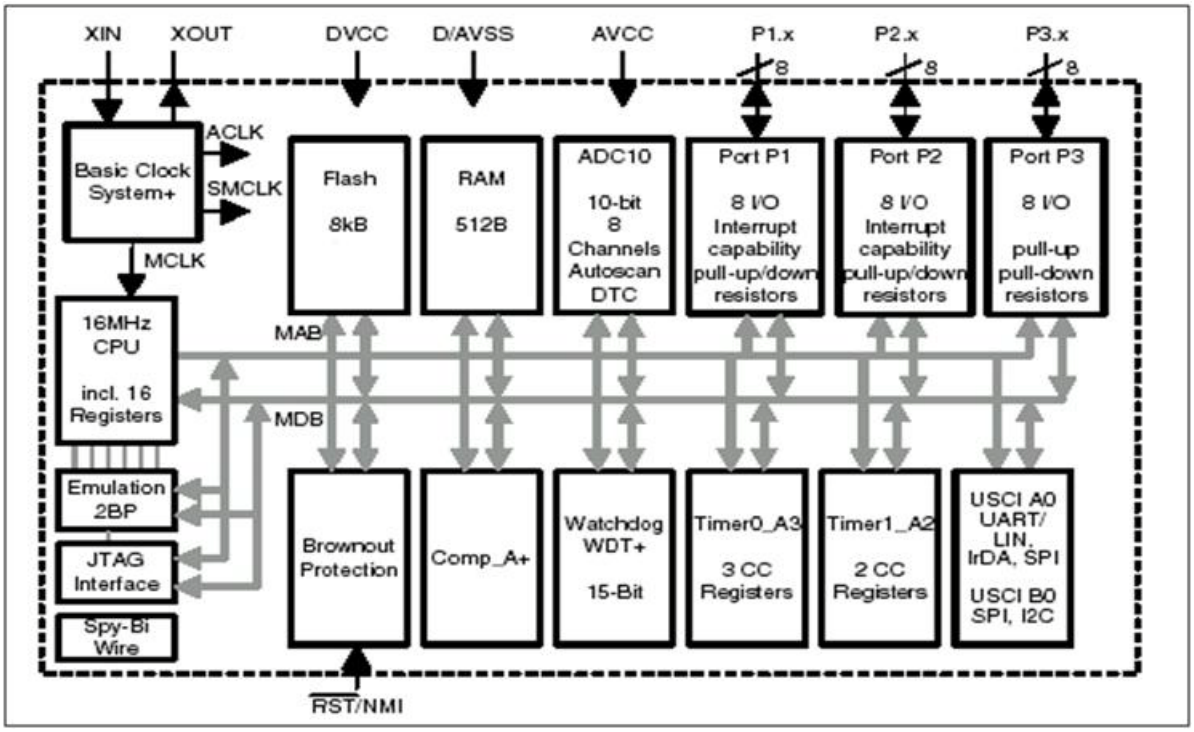


Figure 13: Functional Block Diagram of the MSP430F2132 [16]

RF Transmitter

A commercial off the shelf (COTS) RF transmitter transmitting at 433 MHz was chosen for this function. The RF transmitter, with a block diagram shown in Figure 14, is a Linx Technologies

TXM-433-LR Transmitter [17]. The microcontroller will interface with the RF transmitter and will control the operation of the RF transmitter. The RF transmitter will be capable of transmitting at a data-rate of ten (10) Kbps (kilobits per second) and have a range of 3000 feet. The RF transmitter will use on-off keying (OOK) to modulate the data stream from the microcontroller. The RF transmitter will connect to a COTS 50-Ohm Antenna.

The chip is composed of a frequency synthesizer phased locked loop (PLL) referenced to an internal high precision crystal. This connects to a voltage controlled oscillator (VCO) which outputs to a buffered power amp. The amplifier is switched by the incoming data, which produces a modulated carrier signal. This signal is filtered and output through the 50-Ohm port of the chip to be connected to an antenna.

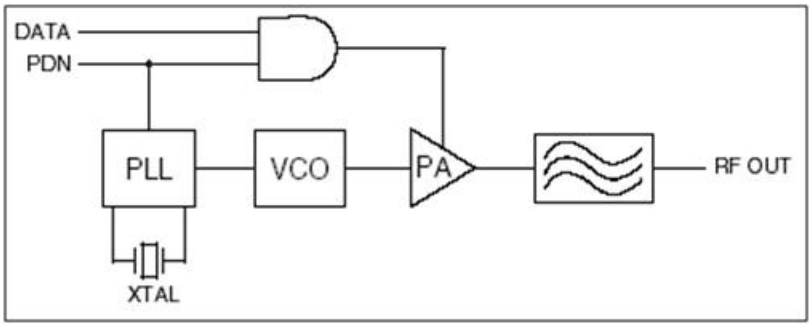


Figure 14: Functional Block Diagram of TXM-433-LR Transmitter

Relay Switch

The relay Switch, shown in Figure 15, provides a “one time” switching mechanism. Once the Sensor Unit is activated, it should stay activated until power (batteries) has been depleted. However, because the activation of the Sensor Unit is dependent on the tilt switch, the circuit would be turned on and off given the orientation of the Sensor Unit. In turbulent water, this would be problematic. Therefore, a single coil latching relay switch provides a solution. This

type of switch has two positions. In Figure 15, the switch is either in the reset position or in the set position. The switch will remain in its given state with no current passing through the coil. To force the relay switch to change states, current must flow through the coil in a specific direction. When current flows from pin 6 to pin 1, the relay connects the reset pin 10 to the common pin 7. When current flows from pin 1 to pin 6, the relay connects the set pin 12 to the common pin 7. For this application, the tilt switches will be used to force the relay into the set state. The circuitry will be such that when armed, current in the opposite direction will not be possible. A COTS relay switch that operates as described above is part DS1E-SL-DC3V from Panasonic Electric Works [18].

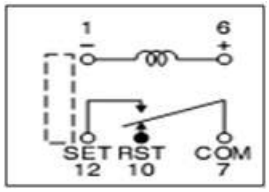


Figure 15: Single Coil Latching Relay

Arm Switch

The arm switch, shown in Figure 16, will connect the tilt switch output to the relay switch. For this design, a slide switch that has three positions will be used. In the middle position, the arm switch will act as an open circuit. In the right position, the arm switch will act as a short circuit connecting the tilt switch outputs to the relay switch. In this position, the arm switch will cause the relay switch to go to its reset state. This state should connect the circuit power (V_{DD}) to ground. In the left position, the arm switch will act as a short circuit, also connecting the tilt switch outputs to the relay switch. The configuration is different from the left position however, and this should cause the relay to switch to the set state and connect V_{DD} to power.

Switches that can achieve this function are Double Pole Double Throw (DPDT) type switches. For the purposes of switching a power supply in this circuit, this switch type is optimal. Also having a small package would be preferable. All of these characteristics were found in part AS23AP made by NKK Switches of America Inc [19].

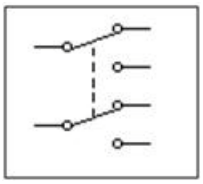


Figure 16: Diagram of the DPDT Switch

Tilt Switches

The tilt switch, shown in Figure 17, will connect the arm switch to the batteries at certain angles or open circuit at others. The switch chosen is the Comus Group S1234 tilt switch [20]. The tilt switch is composed of two leads that can connect to a common electrode based on the orientation of the tilt switch.

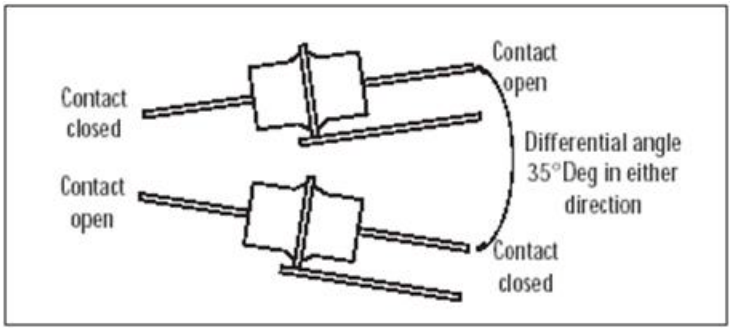


Figure 17: Tilt Switch Diagram

Figure 18 shows the setup of one of the tilt switch when the capsule is buried as well as when it is floating. This is only one of three tilts switches located on the Sensor Unit. The other tilt switches form the same connections but are triggered in different orientations as discussed in Section 4.1.1. Looking at the figure the common electrode, contact A, will be connected to the batteries, contact B will connect to the arm switch, and contact C be open. When buried, the switch will be positioned on the face of the printed circuit board such that contact B will have a 55 degree angle formed above the horizontal plane. In this state, contact B will be open and contact C will be open. The reasoning behind this design is because the Sensor Unit will rotate 90 degrees when it transitions to its floating position. In the floating position, contact B will form a 35-degree angle below the horizontal plane. In this position, contact B will be closed and power will be provided to the arm switch.

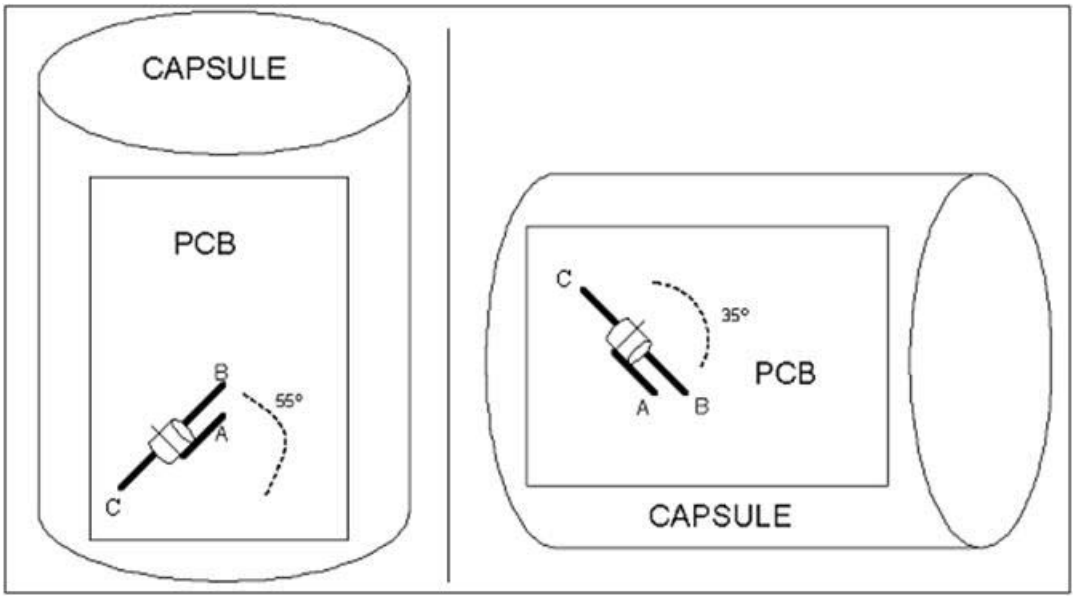


Figure 18: Tilt Switch Position Buried (Left) - Floating (Right)

JTAG/BSL Connection

The JTAG/BSL connector will provide the means to transfer (write) the software and data to the microprocessor memory. In all, the header connection will have 14 pins. The Joint Test Action Group (JTAG) connection will be the main means of programming the microcontroller. There are several simple software options available to program the microcontroller using the JTAG. It will be composed of a 4-pin connection of TDO, TDI, TMS, and TCK pins. An explanation of these pins will be provided in the interface description of the component. The Bootstrap Loader (BSL) is an alternative means of programming and interfacing with microcontroller. The JTAG has a security fuse that can be blown on the microcontroller, so the BSL could serve as the programming means. It can be used for programming and updating software on the processor. The BSL's downside is that there are not easy or readily available means to use the BSL. Lastly, the TEST pin of the microcontroller will be driven high to configure the microcontroller for JTAG and BSL operation. A common ground pin and VCC connection will also be provided.

Serial (RS-232) to USB Converter

The Serial (RS-232) to USB Converter by Future Technology Devices International, Ltd. FT232RL [21], is capable of converting serial (RS-232) data into the USB protocol and vice versa. The FT232RL is used as an intermediary between the microcontroller and the USB interface to the host machine.

Voltage Regulator

The voltage regulator must step down the voltage supplied by the USB power line to a voltage that can be used to supply the serial to USB chip. The voltage regulator being used is the LM3940 packaged as L52B by National Semiconductor [22]. This voltage regulator is a 1A low

dropout regulator which provides 3.3VDC output from a 5VDC supply input (a supply input as low as 4.5VDC will still yield 3.3VDC output).

Batteries

The batteries will provide power to the Sensor Unit. The batteries will need to provide 3 volts DC. The batteries capacity should be such that the unit can be run continuously after activation until it has traveled out of the transmission range. It should also be able to supply the necessary current needed by all of the active components. In this application, the main current drawing components are the microcontroller, the transmitter, and relay switch. Thus, the components together will total, at a maximum, around 39 mA of current. Most small COTS batteries cannot provide this current. Therefore, two 20 mA continuous load batteries will be used in parallel to power the circuit. Placing batteries in parallel simply adds the amount of current each can provide together. This results in 40 mA of continuous load capabilities. The batteries chosen which provide this, the CR-2/BE by Panasonic – BSG [23], combine for 1700 mA-hour of capacity. Given a current draw of 39 mA, it is estimated that the Sensor Unit will operate for around 43.6 hours once activated. The Sensor Unit will have sufficient lifetime after being activated to transmit multiple times.

Antenna

The antenna will be connected to the output of the RF transmitter. It will connect to the RF transmitter through a SMA connector and 50-Ohm micro strip. The antenna should also be designed for the 433 MHz frequency. The capsule length chosen is 6 inches. The current PCB

size has a length of 4 inches. This gives at around 3 inches of space for the antenna given the way the caps are screwed on. Therefore, the chosen antenna was part ANT-433-CW-RH by Linx Technologies Inc. This is a ¼ wave whip type antenna with a length of 2 inches. This size is more than sufficient for the current design and allows flexibility possible alterations to the PCB design down the line.

Capsule

The capsule will be of a cylinder shape and will be watertight. The capsule will be buoyant and will be made of sufficient material to prevent damage once activated (floating). The electronics will be contained within the capsule. The arm switch will be accessible from the outside of the capsule. The capsule size picked will be 6 inches with a diameter of 2 inches. Each end of the capsule shaft will be threaded for attachment to a threaded cap.

Printed Circuit Board (PCB)

The circuit board was designed with ExpressPCB version 6.1.4. The board is an FR-4 epoxy glass with two copper layers with a total of ¼ ounces of copper distributed over all the layers. Variable trace widths are used, but larger traces are used as the main power lines that are routed to each IC.

4.1.3 Design/Interface of Senor Unit Functional Blocks

This section describes the design of each of the functional blocks identified in the previous section. The design of each functional block will show the components needed for that functional block and the how that functional block should be interfaced.

Microcontroller

The microcontroller that will be used for the Sensor Unit will be the MSP430F2132. Figure 19 shows a schematic of the microcontroller that will be used in the Sensor Unit design. An explanation of the pin connections and configurations follow the schematic.

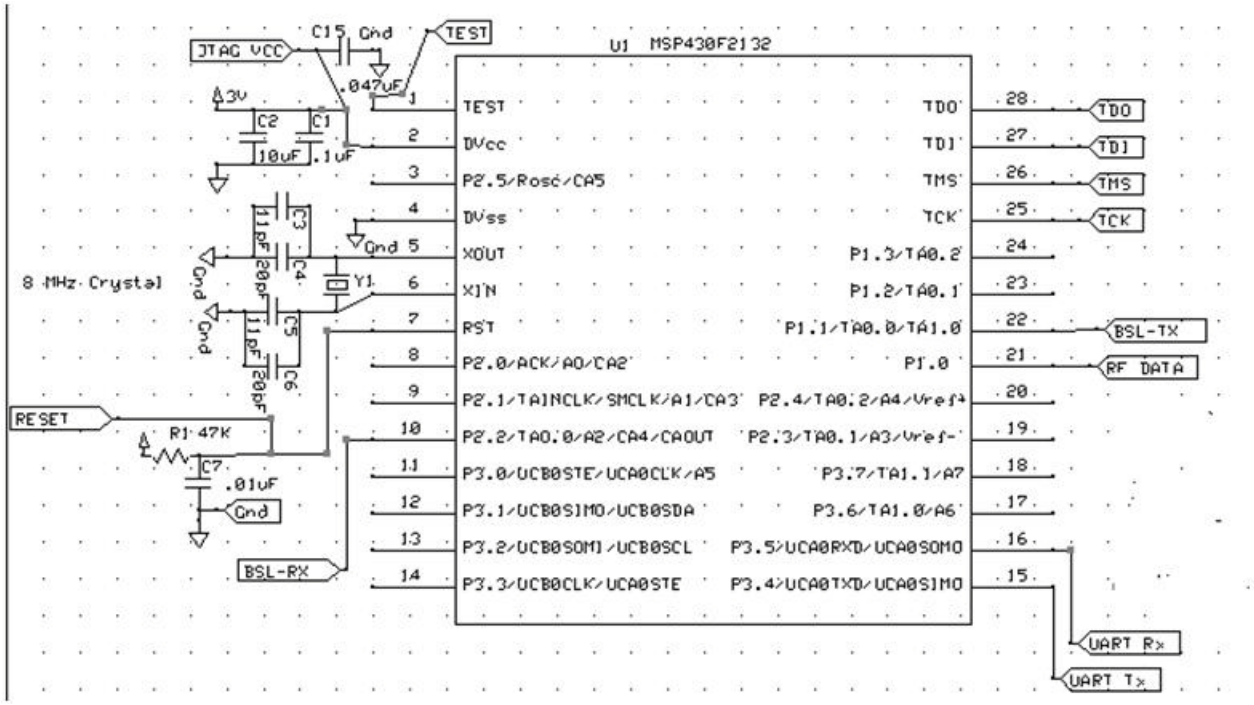


Figure 19: Schematic of the Microcontroller-MSP430F2132

Pin 1 – TEST: This pin is used to enable the standard 4-wire JTAG function. On the MSP430F2132 microcontroller, Port 1 is shared between an I/O function and a JTAG function. To enable the standard 4-wire JTAG function this pin must be given a logic level of 1. This pin will also be given a high-low sequence to initialize the BSL operation. This pin is connected to the 14 pin header.

Pin 2 - DV_{CC}: This is the digital voltage supply pin. This pin connects to the power rail supplying 3 volts. Capacitors will be used to eliminate noise. The capacitors C1 and C2 will have values of 10 µF (C2) and 0.1 µF (C1) based on the hardware setup of MSP430 Microcontrollers presented in Appendix A of [9]. This pin also connects to the 14 pin JTAG/BSL header for the purposes of powering the microcontroller directly from the header when trying to program. This also has a capacitor (C15) to eliminate noise.

Pin 4 – DV_{SS}: This is the digital voltage supply pin with a negative reference. This pin is connected to ground.

Pin 5 – XOUT and Pin 6 – XIN: These two pins connect to the ends of the 4 MHz crystal (Y1). This oscillator will be used to drive the clock of the Microcontroller at its highest frequency. Capacitors (C3, C4, C5, and C6) are placed between the ends of the oscillator and ground given the load capacitance described in the crystal datasheet. The values of the capacitors are determined using a common equation for determining the value of these capacitors. The equation is:

$$C_{crystal} = 2 * C_{load} - C_{stray} \quad (1)$$

In this equation $C_{crystal}$ is the value of each of the two capacitors that will be used with the crystal. C_{load} is load capacitance of the crystal specified in its datasheet as 18 Pico Fared. Finally, C_{stray} is the capacitance present from the traces and input capacitance of the Microcontroller. This is usually accepted to be around 5 Pico Fared. Given these values, $C_{crystal}$ = 2 * 18 pF – 5 pF = 31 pF. Due to the fact that 31 pF capacitors are hard to obtain commercially, 11 pF capacitors are put in parallel with 20 pF capacitors to equal the calculated 31 pF necessary.

Pin 7 - $\overline{\mathbf{RST}}$: This pin is used as a reset pin. It will be controlled by the JTAG/BSL header for programming purposes. If the BSL is not necessary it can be connected to power by adding the resistor and capacitor. It is active low, and therefore a pull-up 47 kilo Ohm resistor (R1) circuit with a 0.01 micro Fared capacitor (C7) will be used to keep this high. These values were based on a model hardware implementation provided from a MSP430 user guide [24].

Pin 10 – P2.2 RX: This pin is used as the data receive pin for the bootstrap loader within the microcontroller. This pin connects to pin 12 of the JTAG/BSL connection.

Pin 15 – UCA0RXD: This pin is used as the receiver pin for the serial UART of the microcontroller. This pin connects to pin 1 of the FT232RL chip.

Pin 16 – UCATXD: This pin is used as the transmit pin for the serial UART of the microcontroller. This pin connects to pin 5 of the FT232RL chip.

Pin 21 – P1.0: This is an I/O pin on port P1 of the chip. This pin will be configured to be an output pin. The output of this pin will be used as an interface to the RF Transmitter chip.

Pin 22 – P1.1 TX: This pin is used as the data transmit pin for the bootstrap loader within the microcontroller. This pin connects to pin 12 of the JTAG/BSL connection.

Pin 25 –TCK: This pin is the Test Clock of the JTAG connection. It synchronizes internal state machine operations. This pin connects to pin 7 of the JTAG/BSL connection.

Pin 26 – TMS: This pin is the Test Mode State of the JTAG connection. It is sampled at the rising edge of TCK to determine the next state. This pin connects to pin 5 of the JTAG/BSL connection.

Pin 27 – TDI: This pin is the Test Data In of the JTAG connection. It represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state. This pin connects to pin 3 of the JTAG/BSL connection.

Pin 28 – TDO: This pin is the Test Data Out of the JTAG connection. It represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK

when the internal state machine is in the correct state. This pin connects to pin 1 of the JTAG/BSL connection.

RF Transmitter

The RF transmitter that will be used in this design is the Linx Technologies TXM-433-LR Transmitter. Figure 20 shows a schematic of the design for the transmitter that will be used in the Sensor Unit.

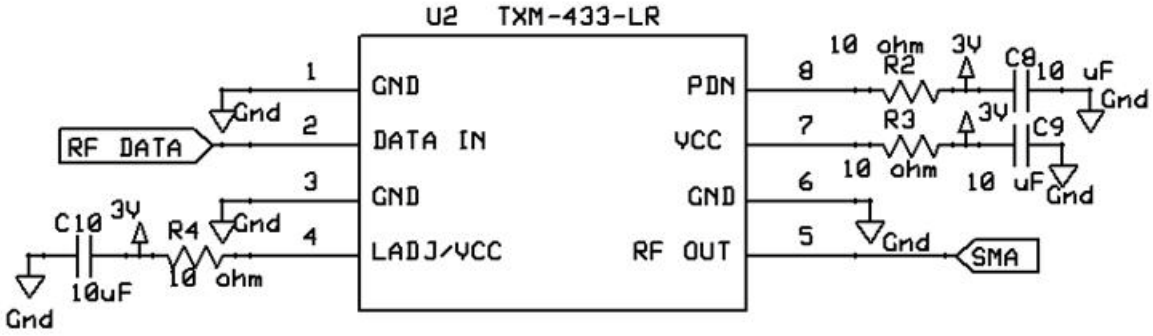


Figure 20: Schematic of the TXM-433-LR Transmitter

Pin 1, Pin 3, Pin 6 – GND: These pins are used to ground the chip. It is connected to the ground plane of the printed circuit board.

Pin 2 – DATA IN: This pin is used for input of digital data to the chip. The MSP430 microcontroller will drive this pin. The output of the transmitter will be modulated based on the input of this pin.

Pin 4 – LADJ/VCC: This pin is the Level Adjust line of the chip. It allows output power of the transmitter to be adjusted based on the voltage applied to this line. For our purposes, maximum range is desired. Therefore, to achieve maximum range, this pin should be connected to V_{DD}. As with the other lines connected to the power supply, a noise eliminating circuit will be placed between power and the pin. This circuit is composed of a 10 (Ohm) resistor (R4) placed between power and the pin as well as a 10 µF (micro-Farad) capacitor (C10) placed between power and ground at that connection.

Pin 5 – RF OUT: This pin produced the modulated RF output of the chip. It will be modulated between the carrier signal and the absence of the carrier signal. This will be connected to the Antenna through a 50-Ohm micro-strip that connects to a SMA connector.

Pin 7 – VCC: This pin is the Supply Voltage of the chip. It is connected to 3 volts DC of power from the Batteries. This pin will also have the noise eliminating circuitry as described for pin 4.

Pin 8 - PDN: This pin is the Power Down pin of the chip. It is used to disable the chip if desired. In the case of our application, the chip will not need to be powered down. This pin will power down the chip if the line is pulled low. Therefore, this pin will be kept high and connected to the 3-volt DC power supply through a noise eliminating circuit.

Relay Switch

The relay switch that will be used in this design is the Panasonic Electric Works' DS1E-SL-DC3V. Figure 21 shows a schematic of the design for the relay switch that will be used in the Sensor Unit. An explanation of the pin connections and configurations follow this figure.

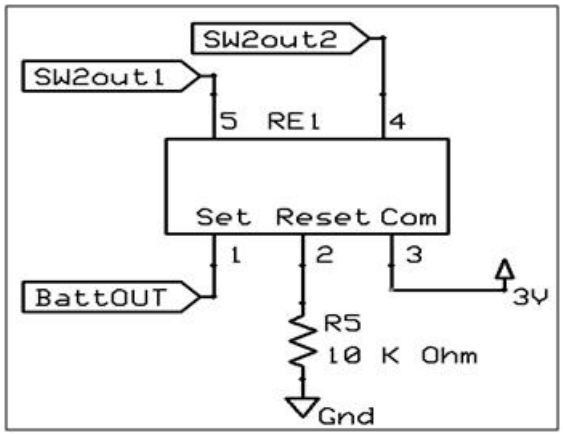


Figure 21: Schematic of the DS1E-SL-DC3V Relay Switch

Pin 1 – Set: This pin is used for a contact position that can be connected to common contact (pin 3). The relay switch will connect these two pins once a current with a voltage potential of no more than 2.4 volts DC is applied in the direction from pin 5 to pin 4. Once the switch has changed to this position, it will remain in this position until current is applied in the direction from pin 4 to pin 5. This pin is connected to the 3-volt DC potential from the Batteries.

Pin 2 – Reset: This pin is used for a contact position that can be connected to common contact (pin 3). The relay switch will connect these two pins once a current with a voltage potential of no more than 2.4 volts DC is applied in the direction from pin 4 to pin 5. Once the switch has

changed to this position, it will remain in this position until current is applied in the direction from pin 4 to pin 5. This pin is connected to ground through a 10 (kilo-Ohm) pull-down resistor.

Pin 3 – Com: This is the common contact pin of the relay switch. It can either be connected to pin 1 or pin 2. The method for which contact this pin is connected to was detailed in the pin 1 and pin 2 descriptions. This pin will serve as the power rail (V_{DD}) for the circuitry following the switches.

Pin 4: This pin is the right terminal of the coil within the relay. It is connected to pin 2 of the arm switch. Depending on the output of pin 2 of the arm switch, this will either be the negative or positive terminal for the coil.

Pin 5: This pin is the left terminal of the coil within the relay. It is connected to pin 5 of the arm switch. Depending on the output of pin 5 of the arm switch, this will either be the negative or positive terminal for the coil.

Arm Switch

The arm switch that will be used in this design is the NKK Switches of America Inc's AS23AP. Figure 22 shows a schematic of the design for the arm switch that will be used in the Sensor Unit. An explanation of the pin connections and configurations follow this figure.

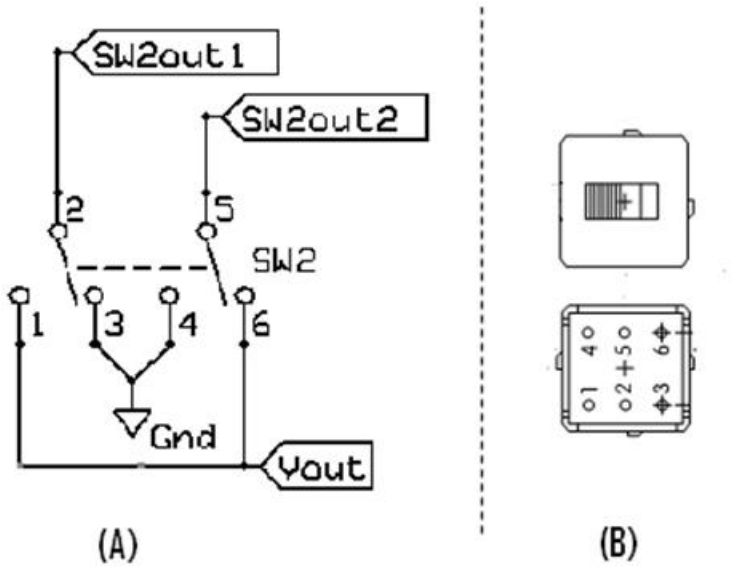


Figure 22: (A) Arm Switch Schematic and (B) Arm Switch Pin View

Pin 1: This is one of the two contacts that pin 2 can connect to. It will be connected when the actuator of the switch is in the left position. The left position is when the actuator is over pins 1 and 4 of the component as shown in **Figure 22**. When connected, this will connect pin 5 of the relay switch to 3.0 volts DC output from the tilt switches. If not connected, if serves as an open circuit.

Pin 2: This is the common contact for pins 1 and 3. In the left position (actuator over pins 1 and 4), this pin connects to pin 1 and will pass 3.0 volts DC from the tilt switches to pin 5 of the relay switch. In the middle position (actuator over pins 2 and 5), this pin does not connect to any pin and is open. In the right position (actuator over pins 3 and 6), this pin connects to pin 3 and will connect pin 5 of the relay switch to ground.

Pin 3: This is one of the two contacts that pin 2 can connect to. It will be connected when the actuator of the switch is in the right position. The right position is when the actuator is over pins 3 and 6 of the component as shown in Figure 22. When connected, this will connect pin 5 of the relay switch to ground. If not connected, it serves as an open circuit.

Pin 4: This is one of the two contacts that pin 5 can connect to. It will be connected when the actuator of the switch is in the left position. The left position is when the actuator is over pins 1 and 4 of the component as shown in Figure 22. When connected, this will connect pin 4 of the relay switch to ground. If not connected, it serves as an open circuit.

Pin 5: This is common contact for pins 4 and 6. In the left position (actuator over pins 1 and 4), this pin connects to pin 4 and will connect pin 4 of the relay switch to ground. In the middle position (actuator over pins 2 and 5), this pin does not connect to any pin and is open. In the right position (actuator over pins 3 and 6), this pin connects to pin 6 and will pass (3.0) volts DC from the tilt switches to pin 4 of the relay switch.

Tilt Switches

The tilt switch that will be used in this design is the Comus Group of Companies S1234. Figure 23 shows a schematic of the design for the arm switch that will be used in the Sensor Unit. An explanation of the pin connections and configurations follow this figure.

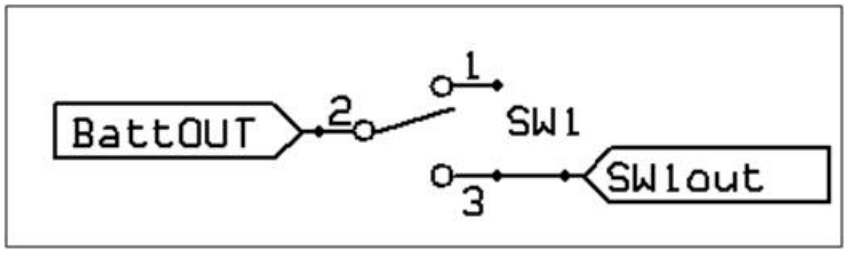


Figure 23: Schematic of the S1234 Tilt Switch

The pins of the tilt switch and their descriptions are provided below:

Pin 1: This pin will connect to pin 2 when the Sensor Unit is oriented in a vertical position. In this case, the positive terminal of the batteries will be connected to an open circuit.

Pin 2: This is the common contact of the tilt switch. When the Sensor Unit is in a vertical position it will connect to pin 1 and when the Sensor Unit is in a horizontal position it will connect to pin 3. If connected to pin 1 the positive terminal of the batteries will be connected to an open circuit. If connected to pin 3, it will connect the 3 volt DC potential of the batteries to pins 1 and 6 of the arm switch.

Pin 3: This pin will connect to pin 2 when the Sensor Unit is oriented in a horizontal position. In this case, the positive terminal of the batteries will supply 3 volts DC potential to pins 1 and 6 of the arm switch.

Voltage Regulator

The voltage regulator that will be used in this design is the LM3940. Figure 24 shows a schematic of the design for the voltage regulator that will be used in the Sensor Unit. An explanation of the pin connections and configurations follow this figure.

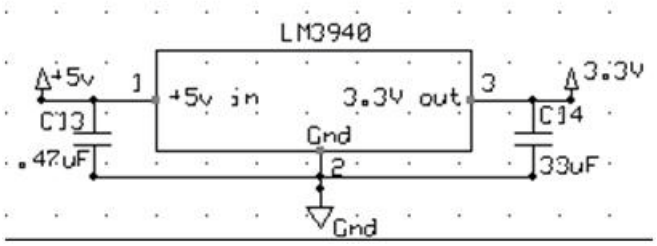


Figure 24: Schematic of the LM3940 Voltage Regulator

Pin 1 – +5Vin: This pin requires a 5V source input. Inputs as low as 4.5V can be used. A .47 uF capacitor is connected between this pin and ground as per the datasheet for the voltage regulator [25]

Pin 2 – GND: This pin is the connected to a common ground.

Pin 3 – +3.3Vout: This pin provides a 3.3V output. It requires a 33 uF capacitor connected between this pin and ground as per the datasheet for the voltage regulator [25].

JTAG/BSL Connector

The JTAG/BSL connection used is through 0.1-inch pitch header pins provided by Molex/Waldom Electronics Corp. Figure 25 below shows a schematic of the design for the JTAG/BSL header that will be used in the Receiver Unit. An explanation of the pin connections and configurations follow this figure.

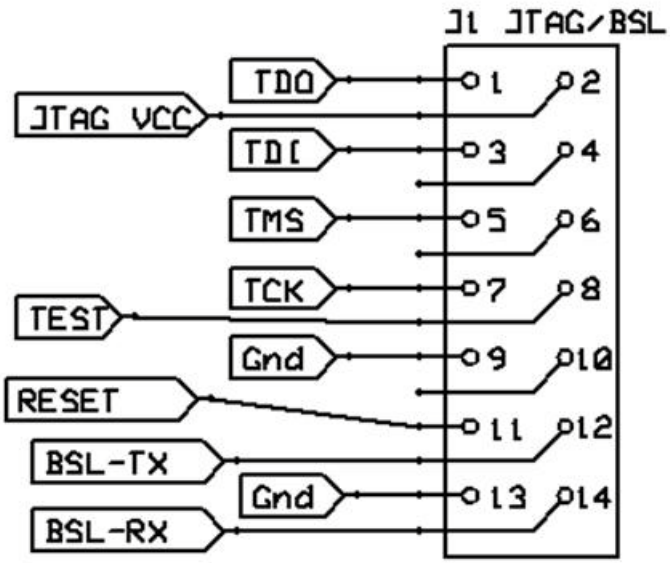


Figure 25: Schematic of JTAG/BSL header

The pins of the JTAG/BSL header and their descriptions are provided below:

Pin 1: This pin is the Test Data Out of the JTAG connection. It represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state. This is connected to pin 28 of the Microcontroller.

Pin 2: This pin is the VCC pin of the JTAG connection. It will be used to power the processor. The voltage of 3 V must be set in the IDE for proper operation.

Pin 3: This pin is the Test Data In of the JTAG connection. It represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state. This is connected to pin 27 of the Microcontroller.

Pin 5: This pin is the Test Mode State of the JTAG connection. It is sampled at the rising edge of TCK to determine the next state. This is connected to pin 26 of the Microcontroller.

Pin 7: This pin is the Test Clock of the JTAG connection. It synchronizes internal state machine operations. This is connected to pin 25 of the Microcontroller.

Pin 8: This will be used to control the Test pin of the Microcontroller. This is pin 1 of the Microcontroller

Pin 9, 13: This pin is available to provide a common ground between the host computer/laptop and the board.

Pin 11: This pin will be used to control the Reset pin of the Microcontroller. This is pin 7 of the Microcontroller.

Pin 12: This is the transmit data output pin from the bootstrap loader of the Microcontroller. It is connected to pin 22 of the Microcontroller.

Pin 14: This is the receive data input pin to the bootstrap loader of the Microcontroller. It is connected to pin 10 of the Microcontroller.

-All other pins are unconnected and unused.

Mini USB Port

The mini USB port used on the Sensor Unit is part H2961CT-ND by Hirose Electric Co. Ltd. Figure 26 below shows a schematic of the mini USB port that will be used. An explanation of the pin connections follow this figure.

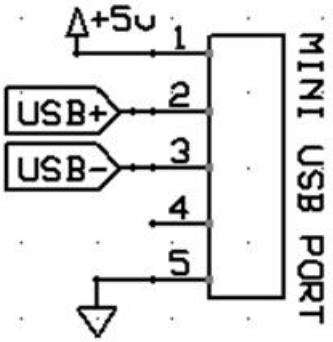


Figure 26: Mini USB Port Schematic

The pins of the mini USB port and their descriptions are described below.

Pin 1: This pin supplies 5 volts DC from the host device. It connects to pin 1 of the voltage regulator.

Pin 2: This pin is connected to the USB positive terminal of the host machine. It also connects to pin 15 of the serial to USB chip (FT232RL).

Pin 3: This pin is connected to the USB negative terminal of the host machine. It also connects to pin 16 of the serial to USB chip (FT232RL).

Pin 4: This pin is not used

Pin 5: This pin connects the host machine to the ground of the PCB.

Batteries

The batteries that will be used in this design are Panasonic –BSG's CR-2/BE batteries. Figure 27 shows a schematic of the design for the batteries that will be used in the Sensor Unit. An explanation of the pin connections and configurations follow this figure.

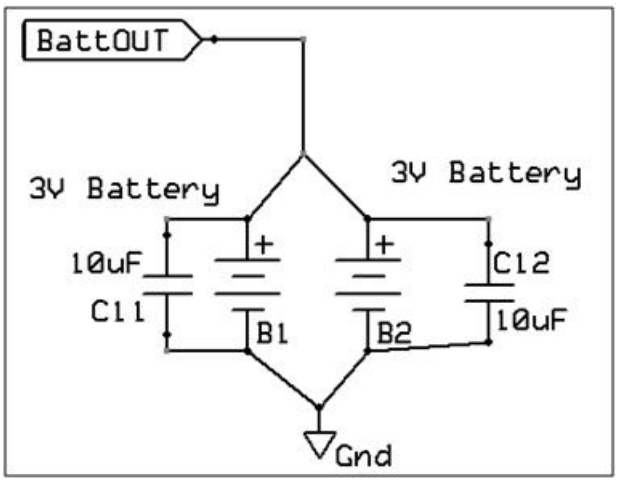


Figure 27: Schematic of the CR-2/BE Batteries

Displayed in Figure 27 there are two instances of the CR-2/BE batteries. They are placed in parallel in order to increase the capacity of the batteries. Each battery has a positive and negative terminal. The positive terminals have 3 volts DC potential and are connected to a line that connects to pin 2 of the tilt switches. The negative terminals serve as the negative reference

and are connected to ground. Capacitors are also placed in parallel with the batteries to increase the stability of the batteries power output.

4.2 RECEIVER UNIT DESIGN

This section contains the design of the Receiver Unit. The first subsection provides a description of the operational functionality required by the receiver system. The second subsection partitions the Receiver Unit into a set of functional blocks. The third subsection provides a detailed description of the interfaces between the functional blocks presented in the second subsection.

4.2.1 Receiver Unit Description of Operation

This section presents a detailed description of the operation of the Receiver Unit. First, this section provides a written description of the Receiver Unit operation. The components mentioned in this section will be discussed in the context of their role within the operation of the Receiver Unit. More specific details on these components can be found in the following sections.

The first operational requirement of the Receiver Unit is that it monitors for released Sensor Units. To do this, the Receiver Unit will be powered continually in order to have it circuitry active at all times. The Receiver Unit will use a RF Receiver designed for the reception of 433 MHz frequency transmissions. The RF Receiver will produce a digital output based on

the presence or absence of the 433 MHz carrier wave. This presence or absence of the carrier wave (known as On-Off Keying or OOK) will be sourced from the Sensor Unit described in the preceding sections of this document. Once a synchronizing period and preamble has been detected (a digital output of 10101010 from the RF Receiver), a Sensor Unit has been detected and the following 23 bytes of data should be processed by the microcontroller.

The second operational requirement of the Receiver Unit is that it should take the 23 bytes of data following the preamble and use the information to determine the state of light indicators driven by the microcontroller. To do this, the data must first be parsed according to the data fields laid out in the Sensor Unit Block shown below in Table 5.

Table 5: Sensor Unit Data Block

Byte	1	2	3	4	5	6	7	8
Offset	0	1	2	3	4	5	6	7
Field	Bridge Identification Number							
Sub-Field	N/A							
Byte	9	10	11	12	13	14	15	16
Offset	8	9	10	11	12	13	14	15
Field	Bridge Identification Number						Serial Number	
Sub-Field	N/A						N/A	
Byte	17	18	19	20	21	22	23	
Offset	16	17	18	19	20	21	22	
Field	Location of Sensor					CRC-16		
Sub-Field	Nearest Structure Unit ID				Color Code	N/A		

For example, the microcontroller will take the first 14 bytes of data received after the preamble to be stored and interpret that as the Bridge Identification Number. The 20th byte of the Sensor Unit Block will determine which light indicators are activated and which are

deactivated. Lastly, this storage of this data and action based off this data will be dependent on the confirmation of the 16-bit CRC, which is the last two bytes received.

4.2.2 Breakdown of Receiver Unit into Functional Blocks

This section partitions the Receiver Unit into a set of basic functional blocks. Figure 28 shows the functional blocks making up the Receiver Unit. A description of each block is presented in this section.

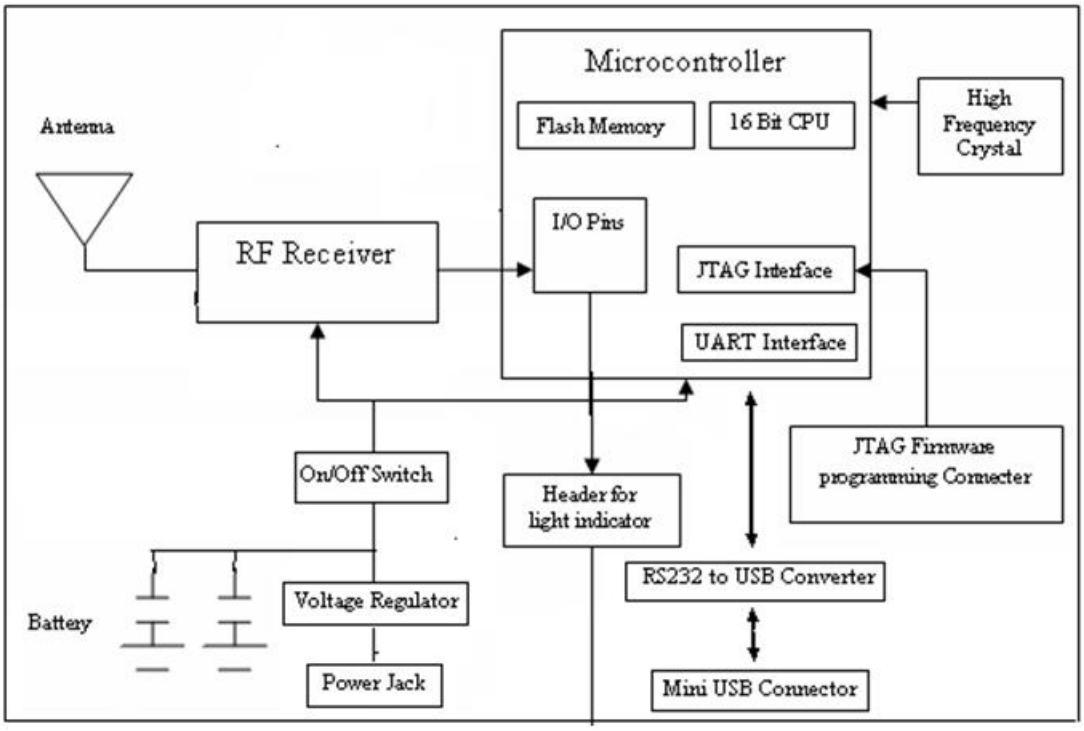


Figure 28: Receiver Block Diagram

Microcontroller

The microcontroller shown in Figure 29 will control the operation of the Receiver Unit. This is the same microcontroller used by the Sensor Unit, the MSP430F2132 made by Texas Instruments [16]. This processor has a 28 pin, 16-bit RISC architecture CPU. The chip package is Thin-Shrink Small Outline Package, which is a four sided surface mount chip package. The Microcontroller is capable of operating at clock speeds up to 16 MHz. The processor will operate from an 8 MHz clock. Due to the incoming data rate of 10 Kbps, 8 MHz is sufficient and cost efficient compared to a 16 MHz clock. The clock will be supplied by an 8 MHz crystal. The processor memory is non-volatile flash memory. It contains 8 KB of flash program memory and 256 bytes of information (user) flash memory. The main fundamental difference between the two flash memories is in how they are segmented and how they are erased. The information memory and main memories can be erased separately and the information memory can be erased in smaller segments. The main memory will contain the program code and the Sensor Unit Data Blocks received. Normally, the Sensor Unit Data Blocks would be stored in the information memory, but small size (256 bytes) of the information memory prevents the Receiver Unit from storing more than a few Sensor Unit Data Blocks. Using the main memory to store this information enables many more Sensor Unit Data Blocks to be stored.

The microcontroller has 24 general input/output (I/O) pins. Of the 24 general I/O pins, five will be used for I/O. The microcontroller must interface with the RF Receiver through one of the I/O pins. This pin will be used along with a compare/capture register to input the incoming data stream. The other four I/O pins will be used to drive the Light Indicator. Another six I/O pins will be used on the microcontroller but in either a programming or interfacing capacity. The microcontroller will be programmed through the JTAG interface. The

microcontroller has a built in JTAG interface through which the microcontroller will be programmed. This requires five pins connected to a JTAG/BSL Header. Four of these are I/O pins while the other is a Test pin used to change the function of these I/O pins. Two I/O pins will also be used to for a Bootstrap Loader (BSL) Connection. These will connect to two pins of the JTAG/BSL Header.

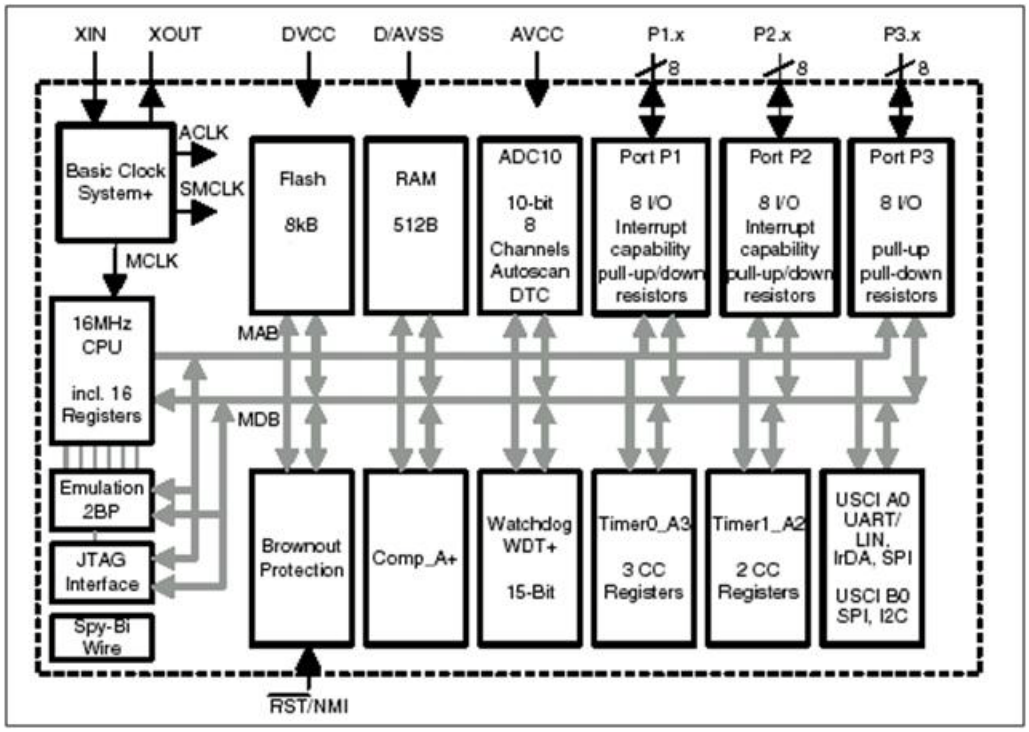


Figure 29: Functional Block Diagram of MSP430F2132

RF Receiver

A commercial off the shelf (COTS) RF Receiver at 433 MHz was chosen for this function. The RF Receiver (Figure 30) chosen was the Linx Technologies RXM-433-LR Receiver [26]. The microcontroller will interface with the RF Receiver and will process the data received from it. The RF Receiver will be capable of transferring data at a rate of ten (10) Kbps (kilobits per

second) and have a range of 3000 feet. The RF Receiver is set up to recover the On-Off Keying (OOK) modulated data stream from the RF Transmitter of the Sensor Unit. The RF Receiver will connect to a COTS 50-Ohm antenna.

The chip is based on a super heterodyne structure that uses several stages to produce the digital out. The input will be connected to the receiver antenna, which is a 50 Ohm antenna. A detailed explanation of the operation of this component can be found in [26].

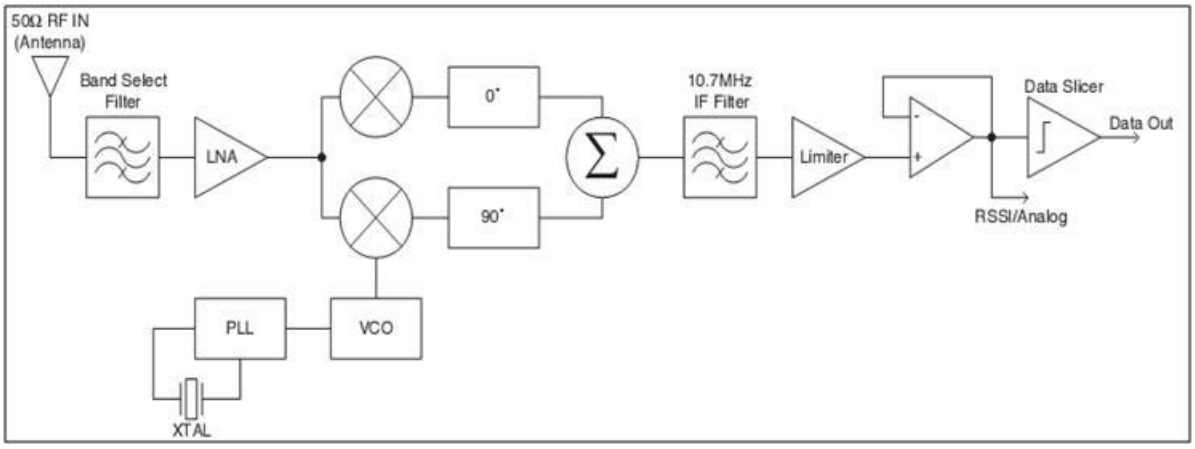


Figure 30: Functional Block Diagram of RXM-433-LR Receiver

JTAG/BSL Connector

The JTAG/BSL Connector will provide the means to transfer (write) the software and data to the microprocessor memory. The header connection will have 14 pins. The Joint Test Action Group (JTAG) connection will be the primary means of programming the microcontroller. There are several simple software options available to program the microcontroller using the JTAG. It will be composed of a 4-pin connection of TDO, TDI, TMS, and TCK pins. An explanation of these pins will be provided in the interface description of the component. The Bootstrap Loader (BSL) is alternative means of programming and interfacing with microcontroller. The JTAG has a

security fuse that can be blown on the microcontroller, so the BSL could serve as the programming means. It can be used for programming and updating software on the processor as well as adding data. It also has the advantage of being able to interface with the processor while it is active. Lastly, the TEST pin of the microcontroller will be driven high to configure the microcontroller for JTAG and BSL operation. A common ground pin and VCC connection are also provided.

Receiver Antenna

The receiver antenna will be connected to the input of the RF Receiver. It will connect to the RF Receiver through a SMA connector and 50-ohm micro strip. The receiver antenna should also be designed for the 433 MHz frequency. The chosen Receiver Antenna was part ANT-433-CW-QW-ND by Linx Technologies Inc. This is a quarter wave whip type Antenna with a length of 6.81 inches. Unlike the sensor antenna, the receiver antenna does not have a length restriction, which persuades the use of a longer antenna.

Light Indicator Header

The header for the Light Indicator will be used to connect I/O pins of the microcontroller to OR gates on the Light Indicator PCB. These OR gates directly drive LEDs indicating the state of bridge scour. The header will be composed of six pins. Four of these pins will be connected from the I/O pins of the microcontroller. Each of these pins will be used to drive a different color LED. The other two pins will be used to connect VCC to the Light Indicator Board and to

provide a common ground. The prototype will have four different LEDs, most likely red, orange, yellow, and green.

Power Switch

A Single Pole Double Throw (SPDT) Switch will be used to power the Receiver Unit. The double throw switch was chosen because of the two different power source options on the board. The switch has three positions to service this. In the middle position (P2), the switch is an open circuit. In the left position (P1), the switch connects the power jack to the VCC of the board. In the right position (P3), the switch connects the batteries to the VCC of the board. The switch part chosen to do this is the MHS12304 SPDT switch by Tyco Electronics Alco. Figure 31, below, illustrates a diagram of SPDT switches.

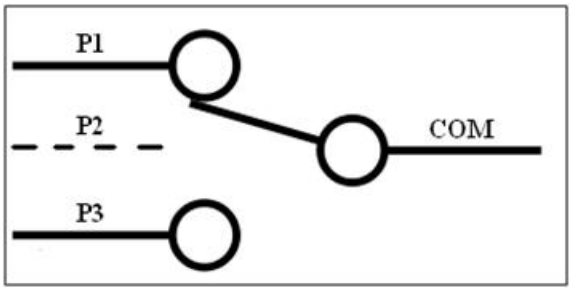


Figure 31: SPDT Switch

Power Jack

The power jack will be used to connect the receiver PCB to power. It is intended to be used with most AC/DC power adapters that connect to a standard outlet. The power jack is a male barrel type connector, which corresponds with most power adapters. The part number is PJ-202A and is manufactured CUI Inc.

Voltage Regulator

The voltage regulator will be used to condition any DC voltage between 3 V DC and 13 V DC to the 3 V DC that is required by the Receiver Unit circuitry. This regulator will take this voltage provided by from the power jack to output it to the circuit through a switch. The voltage regulator to be used is TPS78930DBVR from Texas Instruments.

AC/DC Plug-in Adapter

The AC/DC Plug-in Adapter will allow the Receiver Unit to source power from a standard 120 VAC outlet. The adapter will convert the 120 volts AC to 9 volts DC power. Other transformers with output between 3 V DC and 13 V DC can be substituted. The part chosen at this time is the 212AS09012 by Tamura.

Receiver Case

The case chosen to house the receiver unit is the OD36-2.0 by Pactec. The OD series is a family of IP67 rated outdoor enclosures. These plastic weatherproof enclosures are suitable for harsh outdoor environments or indoor industrial applications. The OD series is described for uses in outdoor wireless communication systems, wall mount lighting controls, security systems, test equipment and other wall mountable systems. This is suitable for this application. The dimensions of the case are 6.1 by 3.6 by 2.1 inches.

4.2.3 Design/Interface of Receiver Unit Functional Blocks

This section will take the functional blocks described in section 4.2.2 and display the design of each. The design of each functional block will show the components needed for that functional block and the how that functional block should be interfaced.

Microcontroller

The microcontroller that will be used for the Sensor Unit will be the MSP430F2132. Figure 32 shows a schematic of the microcontroller design that will be used in the Receiver Unit design. An explanation of the pin connections and configurations follows the schematic. Notice that the pins in use have one label while the used pins have multiple descriptions. This is because each pin may have more than one optional function so only the used pins will have a decided function.

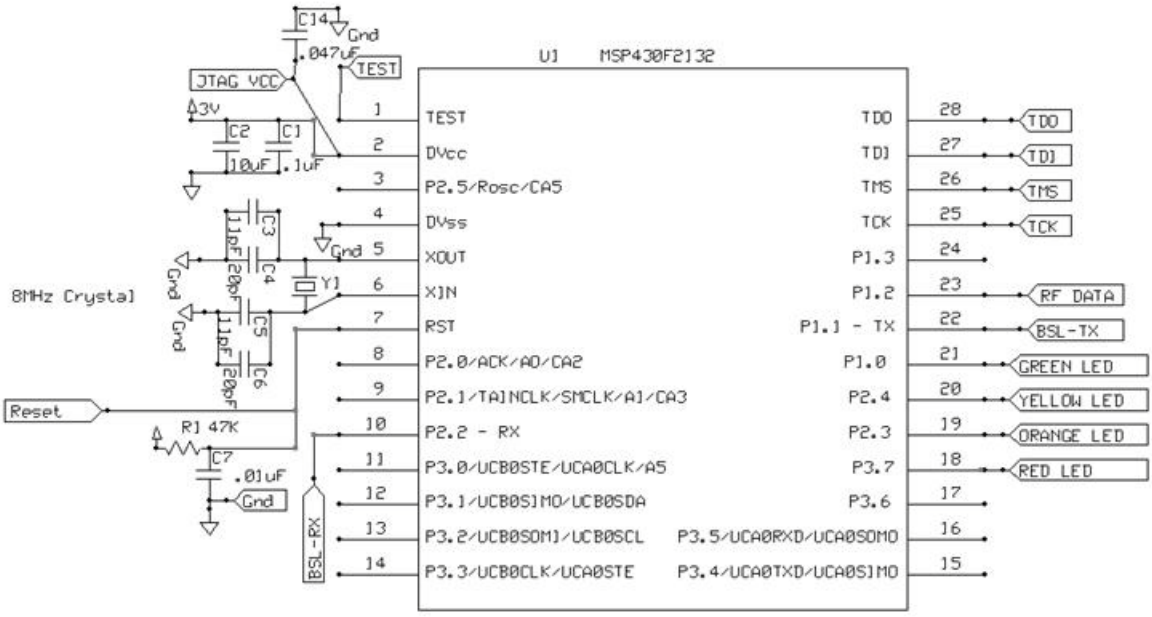


Figure 32: Schematic of Microcontroller-MSP430F2132

The pins that are used and their descriptions are described below:

Pin 1 – TEST: This pin is used to enable the standard 4-wire JTAG function. On the MSP430F2132 microcontroller, Port 1 is shared between an I/O function and a JTAG function. To enable the standard 4-wire JTAG function this pin must be given a logic level of 1. This pin will also be given a high-low sequence to initialize the BSL operation. This pin is connected to the 14-pin header.

Pin 2 - DV_{CC}: This is the digital voltage supply pin. This pin connects to the power rail supplying 3 volts. Capacitors will be used to eliminate noise. The capacitors C1 and C2 will have values of 0.1 µF and 10 µF respectively, based on the recommended hardware setup for MSP430 microcontrollers presented in Appendix A of [27]. This pin also connects to the 14-pin JTAG/BSL header for the purposes of powering the microcontroller directly from the header when trying to program. This also has a capacitor (C15) to eliminate noise.

Pin 4 – DV_{SS}: This is the digital voltage supply pin with a negative reference. This pin is connected to ground.

Pin 5 – XOUT and Pin 6 – XIN: These two pins connect to the ends of the 8 MHz crystal (Y1). This oscillator will be used to drive the clock of the Microcontroller at its highest frequency. Capacitors (C3, C4, C5, and C6) are placed between the ends of the oscillator and ground given the load capacitance described in the crystal's datasheet. The values of the capacitors are

determined using a common equation for determining the value of these capacitors. The equation is:

$$C_{crystal} = 2 * C_{load} - C_{stray} \quad (5)$$

where $C_{crystal}$ is the value of each of the two capacitors that will be used with the crystal. C_{load} is the load capacitance of the crystal specified in its datasheet as 18 Pico-Fared (pF). Finally, C_{stray} is the capacitance present from the traces and input capacitance of the Microcontroller. This is usually accepted to be around 5 Pico-Fared (pF). Given these values, $C_{crystal}$ = 2 * 18 pF – 5 pF = 31 pF. Due to the fact that 31 pF capacitors are difficult to obtain commercially, 11 pF capacitors are put in parallel with 20 pF capacitors to equal the calculated 31 pF necessary.

Pin 7 - $\overline{\mathbf{RST}}$: This pin is used as a reset pin. It will be controlled by the JTAG/BSL header for programming purposes. If the BSL is not necessary, it can be connected to power by adding the resistor and capacitor. It is active low and therefore a pull-up 47 kilo Ohm resistor (R1) circuit with a 0.01 micro Fared capacitor (C7) will be used to keep this high. These values were based on a model hardware implementation provided from a MSP430 user guide [24].

Pin 10 – P2.2 RX: This pin is used as the data receive pin for the bootstrap loader within the microcontroller. This pin connects to pin 12 of the JTAG/BSL connection.

Pin 18 – P3.7: This is an I/O pin on port P3 of the chip. This pin will be configured to be an output pin. The output of this pin will be used to drive the state of the red LED on the Light Indicator Component.

Pin 19 – P2.3: This is an I/O pin on port P2 of the chip. This pin will be configured to be an output pin. The output of this pin will be used to drive the state of the orange LED on the Light Indicator Component.

Pin 20 – P2.4: This is an I/O pin on port P2 of the chip. This pin will be configured to be an output pin. The output of this pin will be used to drive the state of the yellow LED on the Light Indicator Component.

Pin 21 – P1.0: This is an I/O pin on port P1 of the chip. This pin will be configured to be an output pin. The output of this pin will be used to drive the state of the green LED on the Light Indicator Component.

Pin 22 – P1.1 TX: This pin is used as the data transmit pin for the bootstrap loader within the microcontroller. This pin connects to pin 12 of the JTAG/BSL connection.

Pin 23 – P1.2: This is an I/O pin on port P1 of the chip. This pin will be configured to an input pin. The input on this pin comes from the RF Receiver chip. The data will be processed to find Sensor Unit Data Blocks when they are transmitted.

Pin 25 –TCK: This pin is the Test Clock of the JTAG connection. It synchronizes internal state machine operations. This pin connects to pin 7 of the JTAG/BSL connection.

Pin 26 – TMS: This pin is the Test Mode State of the JTAG connection. It is sampled at the rising edge of TCK to determine the next state. This pin connects to pin 5 of the JTAG/BSL connection.

Pin 27 – TDI: This pin is the Test Data In of the JTAG connection. It represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state. This pin connects to pin 3 of the JTAG/BSL connection.

Pin 28 – TDO: This pin is the Test Data Out of the JTAG connection. It represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state. This pin connects to pin 1 of the JTAG/BSL connection.

All other pins will be unconnected and unused.

RF Receiver

The RF Receiver that will be used in this design is the Linx Technologies RXM-433-LR Receiver. **Figure 33** below shows a schematic of the design for the RF Receiver that will be

used in the Receiver Unit. An explanation of the pin connections and configurations follow this figure.

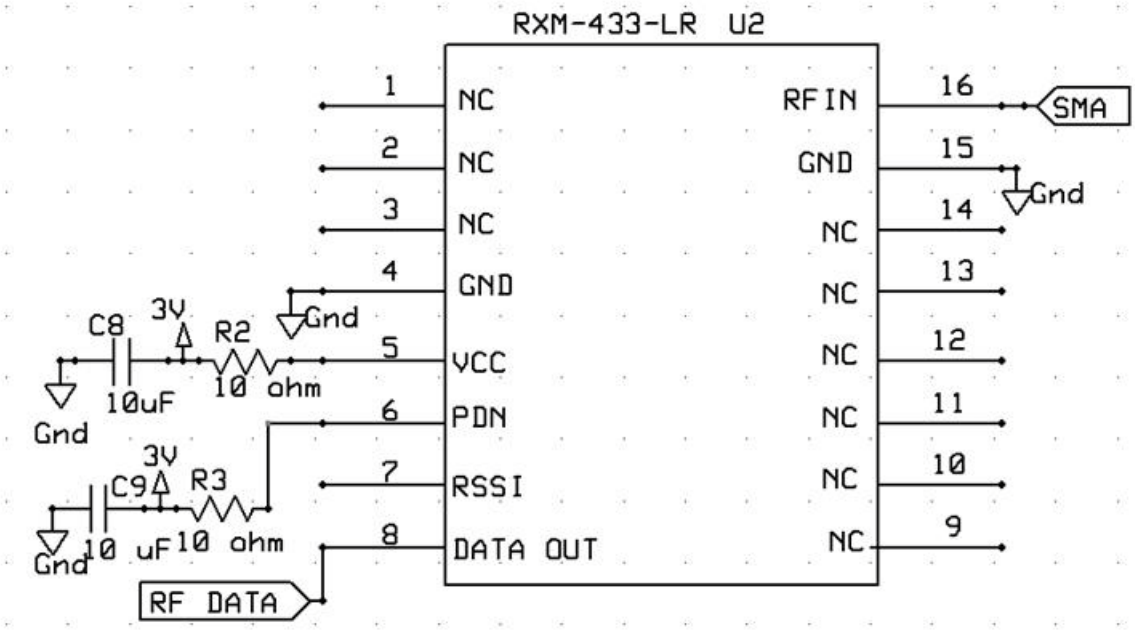


Figure 33: Schematic of RXM-433-LR Receiver

The pins of the Receiver chip and their descriptions are provided below:

Pin 1, 2, 3, 9, 10, 11, 12, 13, 14 – NC: These pins are have no connection and no function specified.

Pin 4, 15 – GND: These pins are used to ground the chip. They will be connected to the ground of the PCB.

Pin 5 –VCC: This pin is the Supply Voltage of the chip. It is connected to 3 volts DC of power from the Power Jack or Batteries. As with the other lines connected to the power supply, a noise eliminating circuit will be placed between power and the pin. This circuit is composed of a 10 Ohm resistor (R8) placed between power and the pin as well as a 10 µF capacitor (C8) placed between power and ground at that connection.

Pin 6 – PDN: This pin is the Power Down pin of the chip. It is used to disable the chip if desired. In the case of our application, the chip will not need to be powered down. This pin will power down the chip if the line is pulled low. Therefore, this pin will be kept high and connected to the 3 volts DC power supply through a noise eliminating circuit.

Pin 7 – RSSI: This is the Received Signal Strength Indicator pin. It provides an analog voltage representing the strength of the received signal. It will not be used in this design.

Pin 8 – DATA OUT: This pin provides the demodulated digital output of the received signal. This will be connected to the Microcontroller.

Pin 16 – RFIN: This pin takes the RF signal input. The input should be in the form of On-Off Keying (OOK). This pin will be connected to the SMA connector of the Receiver Antenna through a 50-ohm micro strip.

JTAG/BSL Connector

The JTAG/BSL connection used is through 0.1-inch pitch header pins provided by Molex/Waldom Electronics Corp. Figure 34 below shows a schematic of the design for the

JTAG/BSL header that will be used in the Receiver Unit. An explanation of the pin connections and configurations follow this figure.

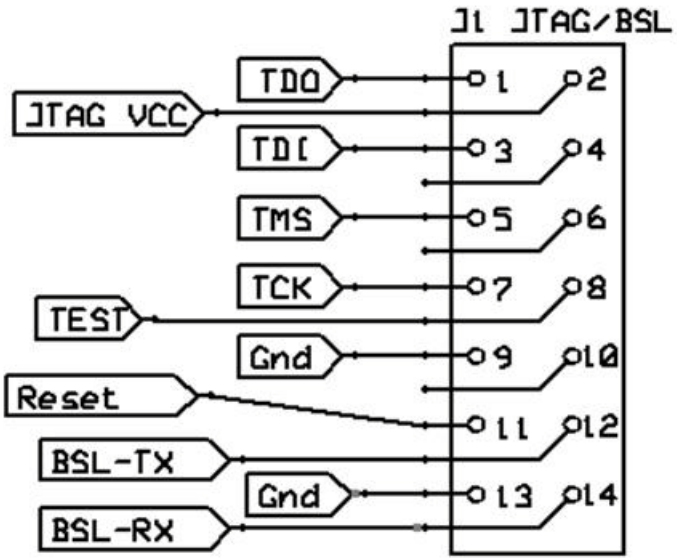


Figure 34: Schematic of JTAG/BSL header

The pins of the JTAG/BSL header and their descriptions are provided below:

Pin 1: This pin is the Test Data Out of the JTAG connection. It represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state. This is connected to pin 28 of the microcontroller.

Pin 2: This pin is the VCC pin of the JTAG connection. It will be used to power the processor. The voltage of 3 V must be set in the IDE for proper operation.

Pin 3: This pin is the Test Data In of the JTAG connection. It represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state. This is connected to pin 27 of the microcontroller.

Pin 5: This pin is the Test Mode State of the JTAG connection. It is sampled at the rising edge of TCK to determine the next state. This is connected to pin 26 of the Microcontroller.

Pin 7: This pin is the Test Clock of the JTAG connection. It synchronizes internal state machine operations. This is connected to pin 25 of the Microcontroller.

Pin 9, 13: This pin is available to provide a common ground between the host computer/laptop and the board.

Pin 11: This pin is used to control the reset (pin 7) of the microcontroller.

Pin 12: This is the transmit data output pin from the bootstrap loader of the microcontroller. It is connected to pin 22 of the microcontroller.

Pin 14: This is the receive data input pin to the bootstrap loader of the microcontroller. It is connected to pin 10 of the microcontroller.

All other pins are unconnected and unused.

Light Indicator Header

The Light Indicator Header uses a 0.1-inch pitch 6 pin header by Samtec Inc. The header will be connected to the Light Indicator component by a ribbon cable. **Figure 35** below shows a schematic of the design for the Light Indicator header that will be used in the Receiver Unit. An explanation of the pin connections and configurations follow this figure.

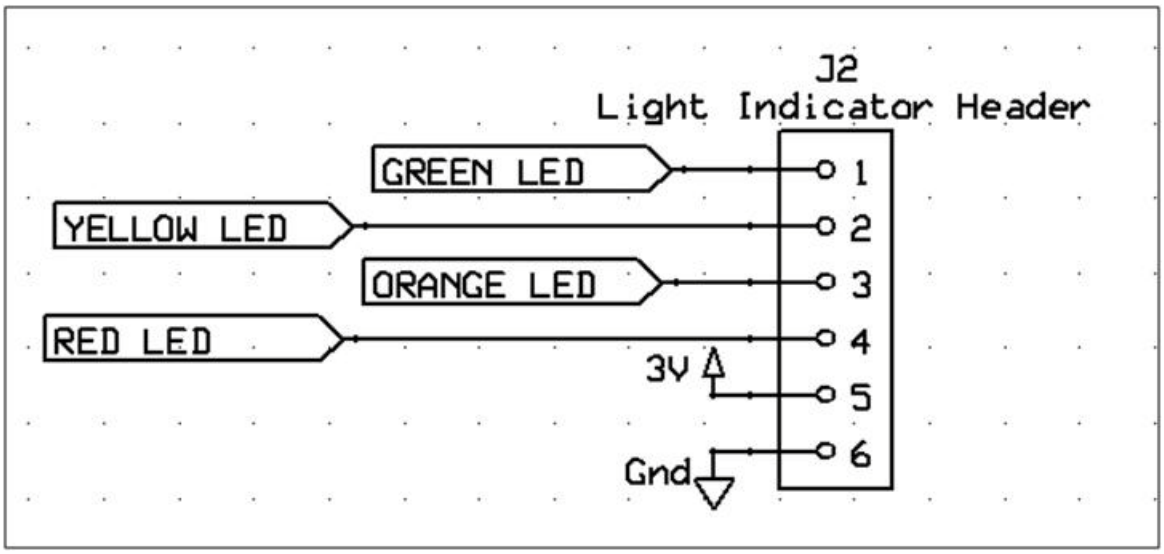


Figure 35: Schematic of Light Indicator Component

The pins of the Light Indicator header and their descriptions are provided below:

Pin 1: This pin connects to pin 21 of the microcontroller. The line driven by this pin is used to light the green LED of the Light Indicator Component.

Pin 2: This pin connects to pin 20 of the microcontroller. The line driven by this pin is used to light the yellow LED of the Light Indicator Component.

Pin 3: This pin connects to pin 19 of the microcontroller. The line driven by this pin is used to light the orange LED of the Light Indicator Component.

Pin 4: This pin connects to pin 18 of the microcontroller. The line driven by this pin is used to light the red LED of the Light Indicator Component.

Pin 5: This pin connects to the 3V power rail of the Receiver Unit PCB. It will be used to provide power to the Light Indicator Component.

Pin 6: This pin connect to the ground plane of the Receiver Unit PCB. It will be used to provide a common ground with the Light Indicator Component.

Power Switch

The Power Switch that will be used in this design is the Tyco Electronics Alcoswitch MHS12304. Figure 36 shows a schematic of the design for the Power Switch that will be used in the Receiver Unit. An explanation of the pin connections and configurations follow this figure.

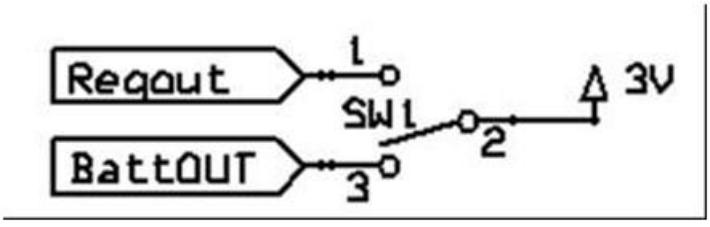


Figure 36: Schematic of SPDT Switch

The pins of the Power Switch and their descriptions are provided below:

Pin 1: This pin will connect to pin 2 when the SPDT Power Switch is in its left position. In this case, the voltage potential pin of the Power Jack will be connected to the 3 V DC power rail of the PCB.

Pin 2: This is the common contact of the power switch. When the power switch is in its left position, it will connect to pin 1. When the Power Switch is in its right position, it will connect to pin 3. If connected to pin 1 the voltage potential pin of the power jack will be connected to the 3 V power rail. If connected to pin 3, it will connect the batteries to the 3 V power rail.

Pin 3: This pin will connect to pin 3 when the SPDT power switch is its right position. In this case, the voltage potential pin of the Batteries will be connected to the 3 V power rail of the PCB**.**

Power Jack

The power jack that will be used in the Receiver Unit is a barrel type male pin power jack by Tamura. Figure 37 shows a schematic of the design for the power jack. An explanation of the pin connections and configurations follow this figure.

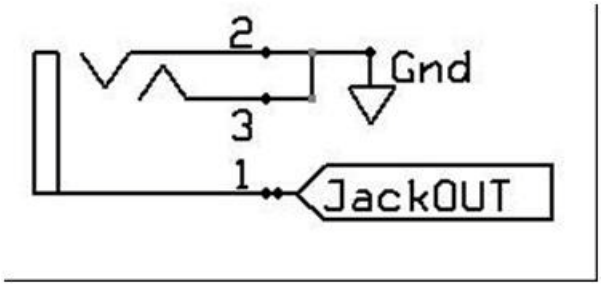


Figure 37: Schematic of Power Jack

The pins of the power jack and their descriptions are provided below:

Pin 1: This pin is the positive terminal of the power jack. This pin has a 3 volts DC potential. It is connected to pin 1 of the power switch.

Pin 2, 3: These pins are the negative terminals of the power jack. Pin 3 is only connected when the adapter is plugged in and pin 2 represents the outer shield of the power jack. Both pins are connected to ground for correct operation.

Voltage Regulator

The Voltage Regulator used is from Texas Instruments. It regulates voltages up to 13 V DC down to 3 V DC. Figure 38 below shows a schematic of the Voltage Regulator. An explanation of pin connections and configurations follow this figure.

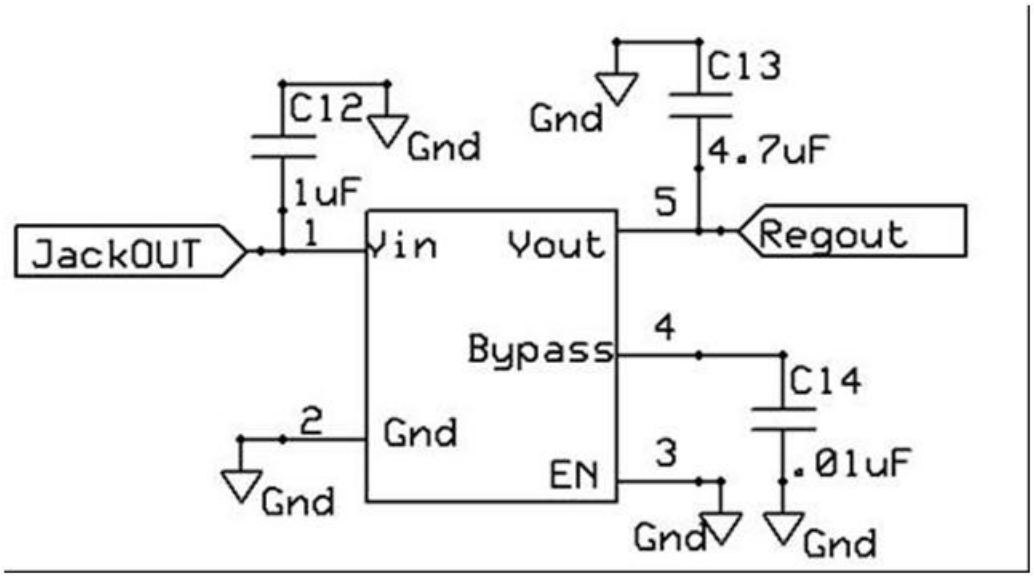


Figure 38: Voltage Regulator

The pins of the voltage regulator and their descriptions are provided below:

Pin 1 –Vin: This is the voltage input pin of the chip. The voltage range here is anywhere between 0 and 13 V DC. For this design, it should be between 3 V DC and 13 V DC. It will have a noise reducing capacitor connected between the voltage input and ground. This input will come from the power jack.

Pin 2 – Gnd: This pin is the ground reference of the chip. This will be connected to the ground plane of the board.

Pin 3 – EN: This is the chip enable pin. It is active low and this is will be connected to ground.

Pin 4 – Bypass: This is the external bypass capacitor connection pin. This pin is used to connect to an external resistor, which further reduces noise by creating a low pass filter.

Pin 5 – Vout: This is the output pin of the regulator. It will produce 3 V DC output. It has a capacitor connected between the output and ground to reduce noise. This output will be used by the circuit and is directed connected to the power switch.

SMA Connector

The SMA connector used is of the reverse polarity type to connect to the reverse polarity antenna chosen. The connector is provided by Linx Technologies Inc. **Figure 39** shows a schematic of the connector. An explanation of the pin connections and configurations follow this figure.

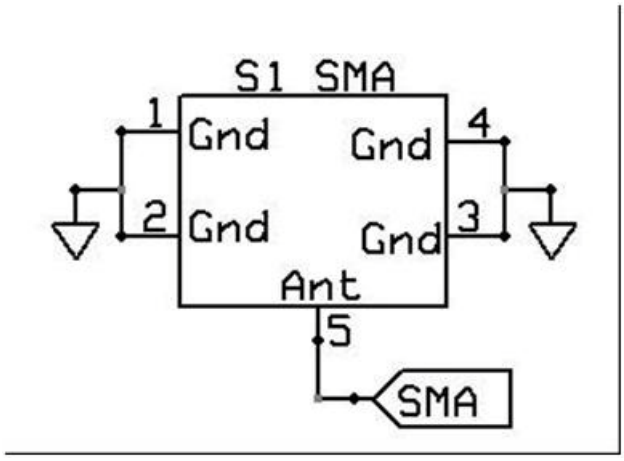


Figure 39: Schematic of SMA connector

The pins of the SMA connector and their descriptions are provided below:

Pin 1, 2, 3, 4 – GND: These pins are used to ground the RP-SMA connector. They are all connected to the ground plane of the PCB.

Pin 5 – ANT: This pin connects to the antenna on the Receiver Unit.

Batteries

The batteries that will be used in this design are Panasonic –BSG's CR-2/BE batteries. Figure 40 shows a schematic of the design for the batteries that will be used in the Receiver Unit. An explanation of the pin connections and configurations follow this figure.

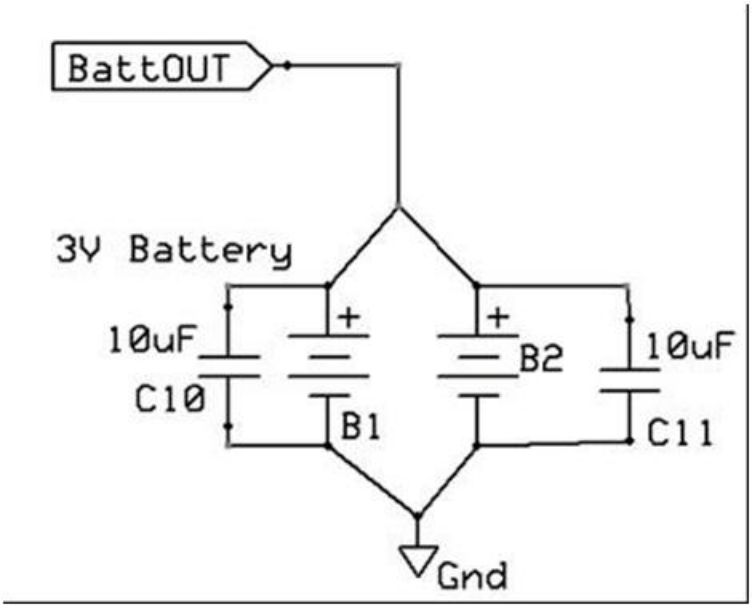


Figure 40: Schematic of the CR-2/BE Batteries

Displayed in Figure 40 there are two instances of the CR-2/BE batteries. They are placed in parallel in order to increase the capacity of the batteries. Each battery has a positive and negative terminal. The positive terminals have 3 V DC potential and are connected to a line that connects to pin 2 of the tilt switches. The negative terminals serve as the negative reference and are connected to ground. Capacitors are also placed in parallel with the batteries to increase the stability of the batteries power output.

4.3 LIGHT INDICATOR DESIGN

4.3.1 Breakdown of Light Indicator Component into Functional Blocks

This Section partitions the Light Indicator Component into a set of basic functional blocks. Figure 41 shows the functional blocks making up the Light Indicator Component. The Light Indicator Component is designed to accept input from up to four independent Receiver Units. This addresses the case when multiple (up to four) Receiver Units are placed at a bridge site to monitor for Sensor Units. A description of each block is presented in this Section.

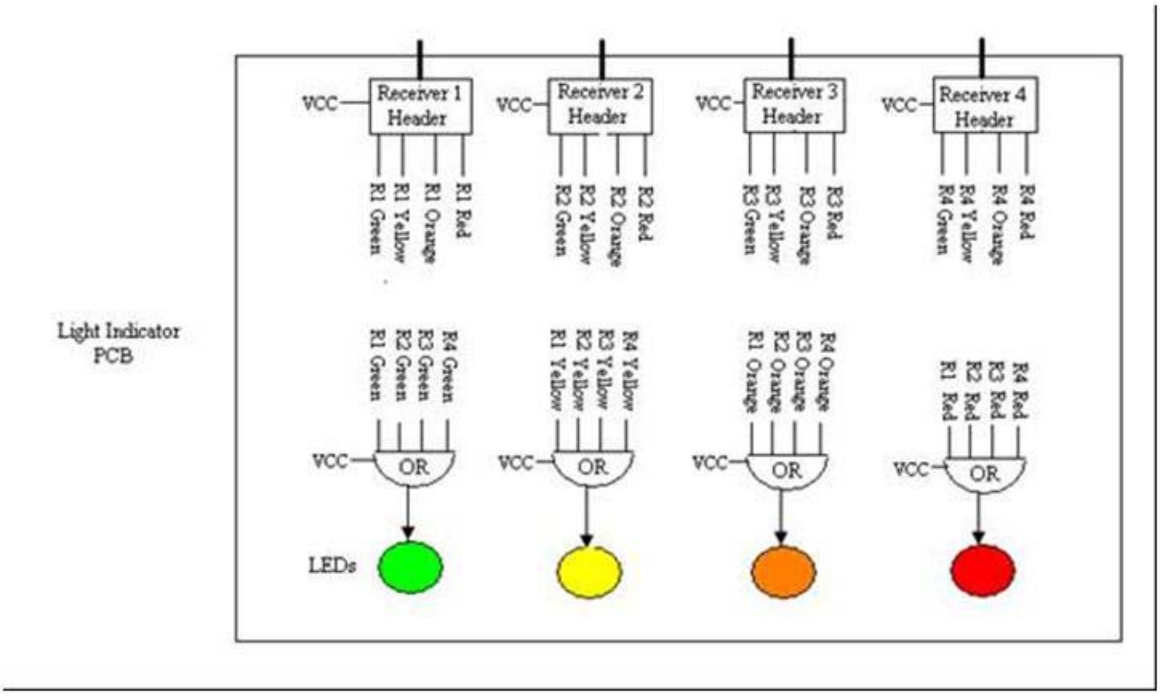


Figure 41: Diagram of Light Indicator

Receiver Headers

These headers are used to connect the Light Indicator Component to the different Receiver Units. In Figure 41, there are four headers. This could be expanded to more headers in the future if necessary. Each Receiver Header can connect to one Receiver Unit. The header pins correspond to the pins of the Light Indicator Header of the Receiver Units. Four of the pins on each of the Receiver Headers correspond to the four different LEDs (green, yellow, orange, and red). The other two pins will be used to connect the Light Indicator Component to the VCCs and grounds of the Receiver Units. The VCC connection will be used to drive the OR gates and as a consequence, the LEDs. The part that provides this six-position header is part TSW-106-07-G-S by Samtec Inc.

OR Gates

The logical OR-gates on the Light Indicator Component will be used to detect if any of the four Receiver Units has a light state corresponding to the LED the OR-gate connects with. Each LED will have a four input OR gate driving it. The OR-gate will take the logical OR of the four inputs and propagate the result to the LED (output). The corresponding LED will be lit (asserted) if at least one of the four inputs is asserted. The part chosen to do this is CD4072BNSR from Texas Instruments and contains two, 4-input OR-gates in one chip.

LEDs

The Light Indicator Component will have four differently colored Light Emitting Diodes (LEDs) connected. Each of these LEDs represents a certain possible severity of scour as denoted by the *Color Code* field of the Sensor Unit Data Block. When a Receiver Unit detects a Sensor Unit and determines the color status of the released Sensor Unit represents, it drives an I/O pin

that will cause the connected OR gate to light its connected LED. The four colors for the prototype of the LEDs are green, yellow (a blue LED may be used depending on availability), orange, and red. What each color represents is presented in section 3.1.2. The LEDs chosen were three LTI series LEDs by Lite-On Inc for the green, yellow, and red LEDs. For the orange LED, part LO 3360-JM-24-0-10-BULK by Osram Opto Semiconductors Inc.

4.3.2 Design/Interface of Light Indicator Component

This section will take the functional blocks described in Section 4.3.1 and display the design of each. The design of each functional block will show the components needed for that functional block and the how that functional block should be interfaced.

Receiver Headers

The Receiver Headers will use a 0.1-inch pitch 6 pin header by Samtec Inc. The headers will be connected to the Receiver Unit by a ribbon cable. Figure 42 below shows a schematic of the design for one of the Receiver Headers that will be used in the Light Indicator Component. There will be four identical instances of this header on the component. An explanation of the pin connections and configurations follow this figure.

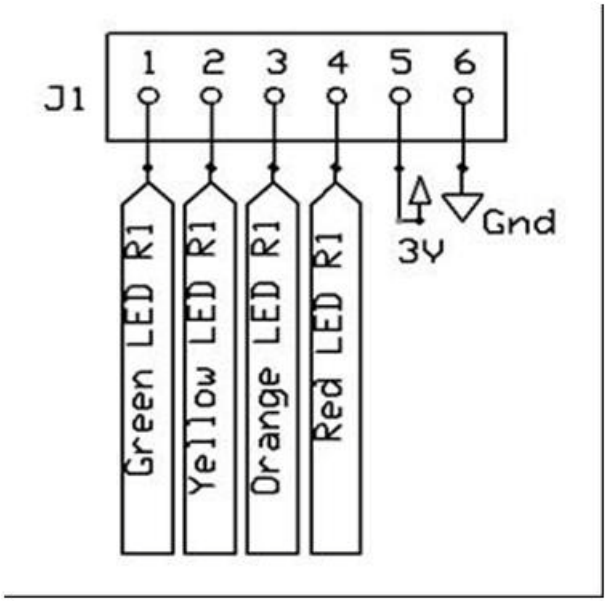


Figure 42: Schematic of Receiver Header

The pins of the Receiver Headers and their descriptions are provided below:

Pin 1: This pin takes input from the Receiver Unit for the operation of the green LED. This pin will be connected to pin (9, 10, 11, 12)[3] of OR gate instance L1. If this pin is high, the green LED connected to OR gate (L1) should be lit.

Pin 2: This pin takes input from the Receiver Unit for the operation of the yellow LED. This pin will be connected to pin (5, 4, 3, 2)[3] of OR gate instance L1. If this pin is high, the yellow LED connected to OR gate (L1) should be lit.

[3] Corresponds to the pin connection for the different Receiver Header instances. First number corresponds to instance J1. Second number corresponds to instance J2. Third number corresponds to instance J3. Fourth number corresponds to instance J4.

Pin 3: This pin takes input from the Receiver Unit for the operation of the orange LED. This pin will be connected to pin (9, 10, 11, 12)[3] of OR gate instance L2. If this pin is high, the orange LED connected to OR gate (L2) should be lit.

Pin 4: This pin takes input from the Receiver Unit for the operation of the red LED. This pin will be connected to pin (5, 4, 3, 2)[3] of OR gate instance L2. If this pin is high, the red LED connected to OR gate (L2) should be lit.

Pin 5: This pin connects to the 3 Volts DC potential of the Receiver Unit. It will be used to supply the OR gates.

Pin 6: Provides a common ground between the Receiver Unit and the Light Indicator component.

OR Gates

The 4-input OR gates (Figure 43) will be provided by Texas Instruments. Part CD4072BNSR contains two, 4 input OR gates on the chip. Two instances of this chip will be used in the Light Indicator Component. An explanation of the pin connections and configurations follow this figure.

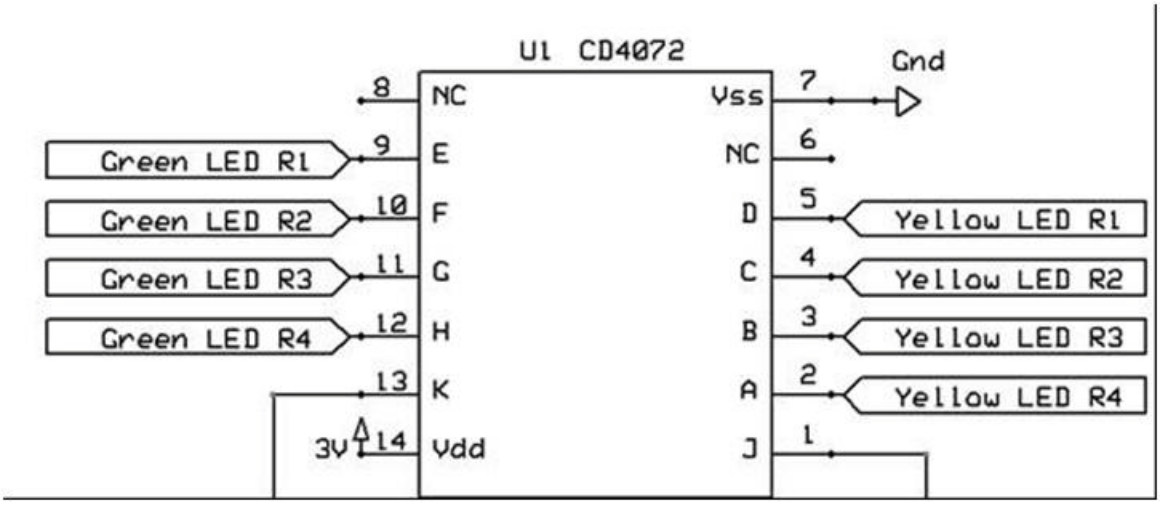


Figure 43: Schematic of OR Gates

The pins of the OR gates and their descriptions are provided below:

*Corresponds to the two instances of the OR gate chips. With the order being (instance U1/ instance U2).

Pin 1 – J: This is the output for the logical OR of pins 2, 3, 4, and 5 (J = A + B + C + D, where '+' denotes logical OR). This pin will connect to the yellow/red)* LED through a current limiting resistor. If the result is true, this pin will drive the (yellow/red)* LED high causing the LED to light. If the result is false, this pin will drive the (yellow/red)* LED low causing the LED not to light.

Pin 2 – A: This is one of 4-input pins that will be logically ORed together to produce output on pin 1. The input to this pin comes from pin (2/4)* of receiver header 4 (R4).

Pin 3 – B: This is one of 4-input pins that will be logically ORed together to produce output on pin 1. The input to this pin comes from pin (2/4)* of receiver header 3 (R3).

Pin 4 – C: This is one of 4-input pins that will be logically ORed together to produce output on pin 1. The input to this pin comes from pin (2/4)* of receiver header 2 (R2).

Pin 5 – D: This is one of 4-input pins that will be logically ORed together to produce output on pin 1. The input to this pin comes from pin (2/4)* of receiver header 1 (R1).

Pin 6 – NC: This pin has no connection and will not be used.

Pin 7 – V_{ss}: This is negative reference for the voltage of the chip. This will be connected to ground.

Pin 8 – NC: This pin has no connection and will not be used.

Pin 9 – E: This is one of 4-input pins that will be logically ORed together to produce output on pin 13. The input to this pin comes from pin (1/3)* of receiver header 1 (R1).

Pin 10 – F: This is one of 4-input pins that will be logically ORed together to produce output on pin 13. The input to this pin comes from pin (1/3)* of receiver header 2 (R2).

Pin 11 – G: This is one of 4-input pins that will be logically ORed together to produce output on pin 13. The input to this pin comes from pin (1/3)* of receiver header 3 (R3).

Pin 12 – H: This is one of 4-input pins that will be logically ORed together to produce output on pin 13. The input to this pin comes from pin (1/3)* of receiver header 4 (R4).

Pin 13 – K: This is the output for the logical OR of pins 9, 10, 11, and 13 (K = E + F + G + H, where '+' denotes logical OR). This pin will connect to the (green/orange)* LED through a current limiting resistor. If the result is true, this pin will drive the (green/orange)* LED high. If the result is false, this pin will drive the (green/orange)* LED low.

Pin 14 – V_{dd}: This is the positive reference for the voltage of the chip. This will be connected to the 3 V DC supplied by the receiver headers.

LEDs

Four LEDs will be used on the Light Indicator Component. These LEDs will be driven based on the output of the OR gates. Three of the four LEDs are made by Lite-on Inc, while the orange LED is made by Osram Opto Semiconductors Inc. Figure 44 shows the setup of one of the LEDs. This setup will differ for each LED for the resistor value and input source.

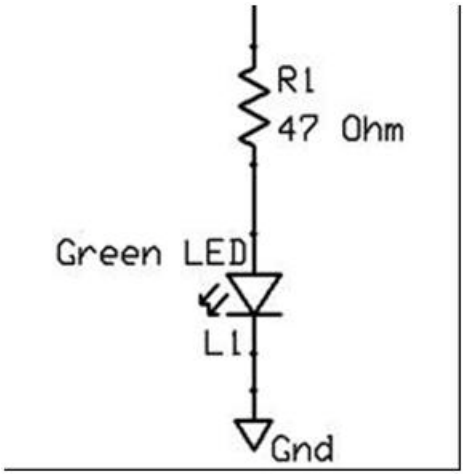


Figure 44: Schematic of LED Circuit

LED setup: The LED circuit is simple. A current limiting resistor will be placed be the LED and the voltage source. The value of this resistor is determined through Ohm's Law. In this case, it would be in the form:

$$R = \frac{V}{I} \qquad (6)$$

The voltage supply, *V*, comes from the output of the 4-input OR Gate. The LED voltage drop and LED current rating are specified by each LED's datasheet. Plugging these values in and solving gives us the minimum value of the resistor. The resistor will be connected to the anode of the LED. The cathode of the LED will be connected to ground.

Green LED resistor value: 47 Ohm

Yellow LED resistor value: 45 Ohm

Orange LED resistor value: 100 Ohm

Red LED resistor value: 50 Ohm

5.0 SOFTWARE DESIGN

5.1 TRANSMITTER SOFTWARE

As mentioned above, the software has two main tasks. It must take input that will comprise the Sensor Unit Data Block and store this to information flash memory. Then the software must take this data and send it to the RF Transmitter through output pin P1.0. The next two subsections explain each of these tasks in more detail with flow charts to illustrate the concept.

5.1.1 Initialization

To begin, the microcontroller must be setup and initialized. This includes initializing variables, setting the necessary microcontroller registers, and disabling the watchdog timer. Initializing the variables within this program consists of setting all counters to zero, setting pointers to the proper address, setting up a dummy Sensor Unit Data Block, and setting the priority of the Sensor Unit. Next, the program disables the Watchdog Timer of the MSP430. The purpose of the Watchdog Timer is to perform a controlled system restart after a software problem has occurred. This is based on a specified time interval set in the Watchdog Timer registers. On initial power up, this interval is 32768 cycles [24]. If the system is restarted during flash operations, unexpected results could occur. For this reason, the Watchdog Time is disabled.

MSP430 devices come with calibrated Basic Clock Module settings for specific frequencies stored in information memory segment A [16][24]. The next part of the program will check to ensure this memory is intact within information memory. Unless the processor has been corrupted or the information memory has been purposely erased, this should not be the case. These stored settings are then used in the next step to setup the registers of the Basic Clock Module. The other register that is set up is the Flash Timing Generator. The Flash Timing Generator controls erase and write operations. It has a particular operating frequency that must be used. For our processor this is 333 kHz. To achieve this operating frequency, clock dividers settings in a flash register must be specifically set.

Another timer must also be setup to control the timing of the output to the RF transmitter chip. Given the 10 kbps data rate constraint given by the RF transmitter chip, bits can only be reliably changes every 0.0001 seconds (1/10 kbps). In this system the processor is running at 8 MHz. This means a clock cycle is produced every 0.000000125 seconds. Thus if an action is to occur every 0.0001 seconds, 800 clocks cycles should elapse.

The last initial steps are to set up the pins of the microcontroller needed. First, the I/O line of the microcontroller is then setup. For the Sensor Unit this is pin 21 or P1.0. This pin is set to the I/O function as an output pin. Next the UART pins need to be setup. On this microcontroller P3.4 and P3.5 need to be set to a UART function. A baud rate also needs to be set for the UART. In this system, 19200 kbps has been arbitrarily selected. Recommended settings for each baud rate given a crystal frequency can be found in [24]. These settings were applied.

5.1.2 Sensor Unit Data Block Storage

The data to be stored in the flash information memory of the MSP430F2132 is based on the Sensor Unit Data Block presented in section 3.1.2. Although a dummy set of data will be initialized within the program, the Sensor Unit Data Block specific to the site must be input through use of the UART communication. Details on the host side of UART communication will be provided in

An interrupt is set up for the UART communication. Each time a character is sent from the host pc, it is placed in a buffer and the interrupt is entered. Here the input can be processed. For the Sensor Unit, this interrupt is only responsible for either storing to the Sensor Unit Data Block to a temporary array or sending the stored block back to host pc to be displayed.

To display the Sensor Unit Data Block stored in memory the UART will look for a (#) character. If this character is received by the program it will enter a function that will output the Sensor Unit Data Block. This print function sets a pointer to the flash memory location where the block is located. This location is the beginning of information memory segment B (0x1080). The function will take each of the 23 bytes one at a time and parse them into nibbles. These nibbles are converted to their ASCII equivalent for the hex representation of each nibble. Printed to the host machine on their Hyper Terminal display will be a sequence of 46 characters (0-9,A-F) representing the 23 bytes stored in flash memory.

Otherwise, the interrupt has two modes dictated by the state of a variable titled (UART_mode). This variable is initialized to 0. In this state, the interrupt is only looking for two characters. These characters are the aforementioned (#) character or a (*) character. Once the interrupt has seen two consecutive (*) characters in a row it will enter the second mode where UART_mode is equal to 1. Once in this mode, the next 42 characters will be taken and

stored as the Sensor Unit Data Block. Every two characters received, which must be 0-9 or A-F, will be concatenated to form a byte. After the 21rd byte is formed, a function takes the bytes and forms the last two bytes of the Sensor Unit Data Block.

The last part of the program places the Sensor Unit Data Block in information memory. To do this, a pointer is first set to the address of information memory segment B. As mentioned earlier, segment A is used for calibration information. Next, the flash segment is unlocked and set to be erased. This is done using a flash settings register. Following this is a write, termed a *dummy write* that will erase the segment [13]. Then the segment can be written with Sensor Unit Data Block. The flash register is first set to write. Next, a loop is entered that writes one byte of the 23 total data byes. Lastly, the flash write bit is cleared and lock bit is reset within the flash register. Once the data has been placed in memory, a "data_txed" string will be displayed on the host pc's Hyper Terminal window.

Figure 45 shows the flowchart for the process of storing the Sensor Unit Data Block to information memory.

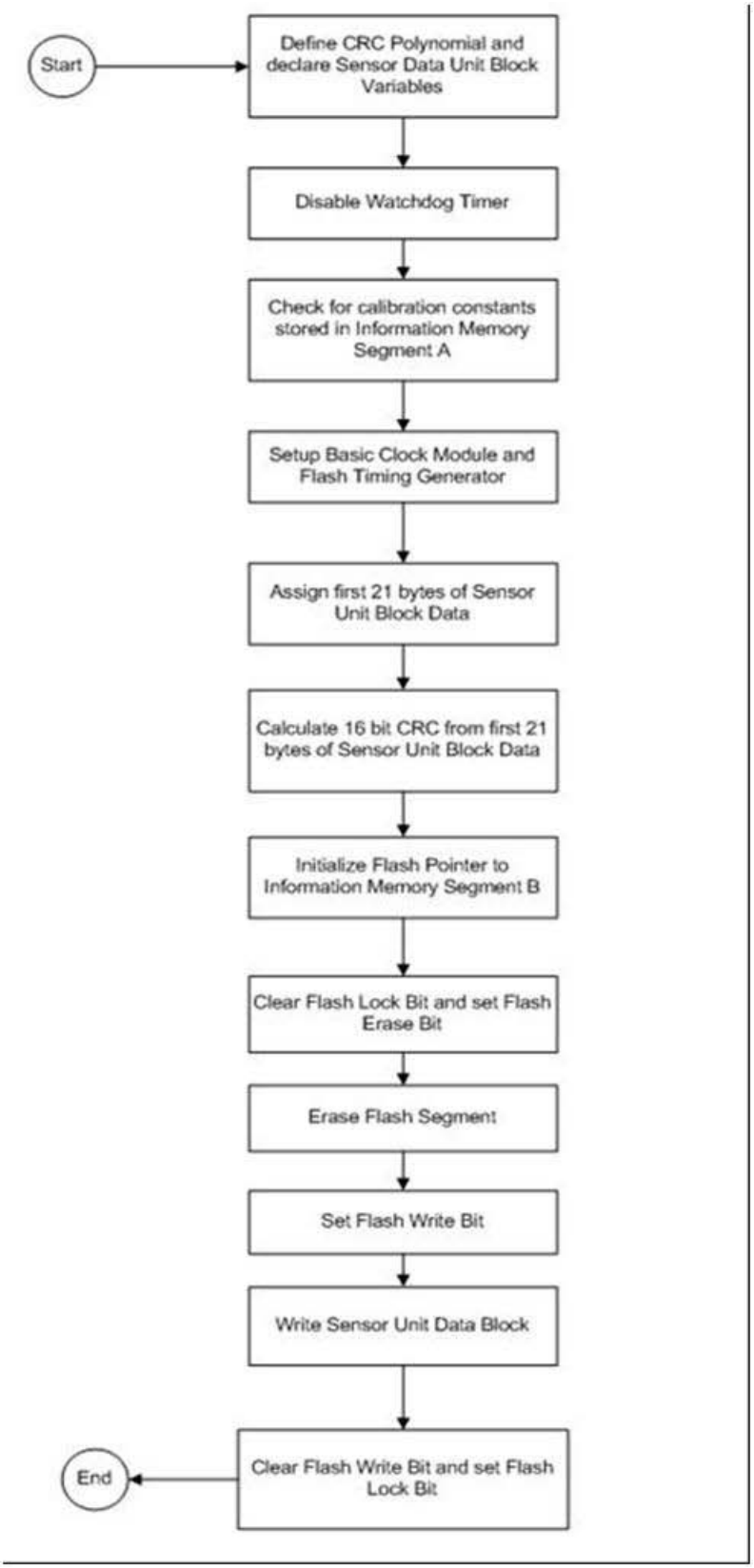


Figure 45: Flowchart for Data Storage Program

5.1.3 Data Transmission

5.1.3.1 Manchester Encoding

The Sensor Unit Data Block will be transmitted using Manchester Encoding. Although there are various interpretations on the encoding, the principle is based on the transitions made in the mid bit period. The bit period will be 0.4 milliseconds. This system defines a logical '0' being transmitted as a mid bit transition from low to high. It defines a logical '1' being transmitted as a mid bit transition from high to low. This is shown in Figure 46 below.

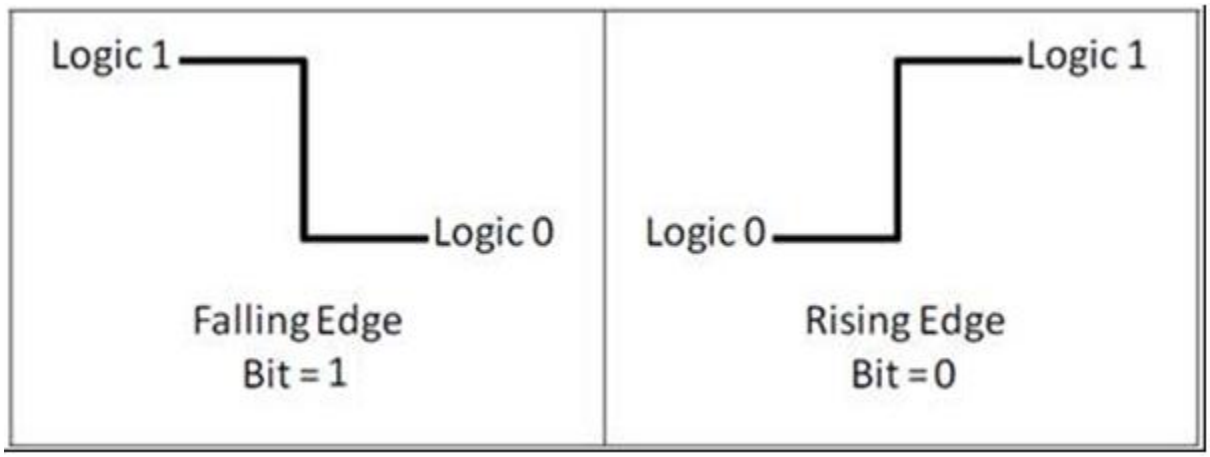


Figure 46: Manchester Encoding

5.1.3.2 Synchronization

A reference point must be set by the transmitter in order for the Receiver Unit to properly decode the data. Normally, in a synchronized system, just a preamble can be sent to signify the beginning of the data packet. However there are a few constraints which make this insufficient for this system. First, the systems are not synchronized. This means a set sampling of the

transmitted bit stream cannot be used to determine the mid bit transition. Instead, the capability of Receiver Unit to detect edges and the time between these edges will be used. Secondly, using edge detection brings forth the need to establish whether each edge is rising or falling. If the direction of the first transition is known, the direction of each succeeding edge can be kept track of.

The solution to these problems is a synchronization period. Immediately before data is sent, the line will be pulled high for a known period. This period must be larger than the biggest time difference in edges. For this system a period of 18000 clocks or 2.25 milliseconds will be used. This also establishes the direction of edges. Following the synchronization period, the Receiver Unit knows the last edge is a falling edge. Using this reference the following data can be decoded by the Receiver Unit.

5.1.3.3 Transmission

The Sensor Unit Data Block is stored at the beginning of information memory segment B (address 0x1080). When data are to be transmitted a pointer is set to this address and a bytes are retrieved one at time for a total of 23 bytes. For each byte, the bits are determined one at a time. This is done using a mask byte containing seven 0's and one 1 digit. The position the 1 digit is within the byte is the bit being tested in the data byte. For example, a byte with the value of 128 or (10000000 binary) will test the most significant digit, a byte with the value of 64 (01000000) will test the second most significant bit, etc. Performing an AND operation between the mask and the data byte will yield the value of the tested digit. If the result of the operation is 1 (true),

then the value of the digit is a 1. If the result of the operation is 0 (false), then the value of the digit is a 0.

Given the value of a current bit and the last bit, a certain sequence of transitions will be output. This sequence is based on Manchester Encoding as mentioned above. Figure 47 illustrates what each of these sequences would look like. All four possible combinations for two consecutive bits are shown.

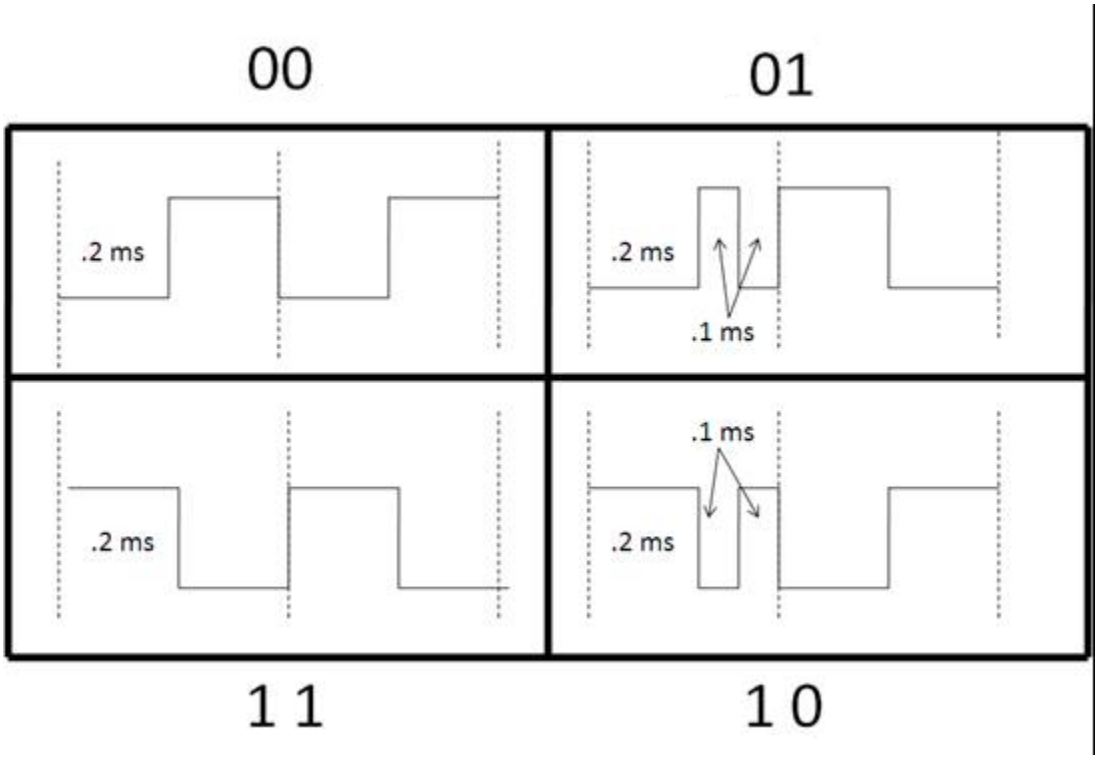


Figure 47: Bit Transition Sequences for Consecutive Bits

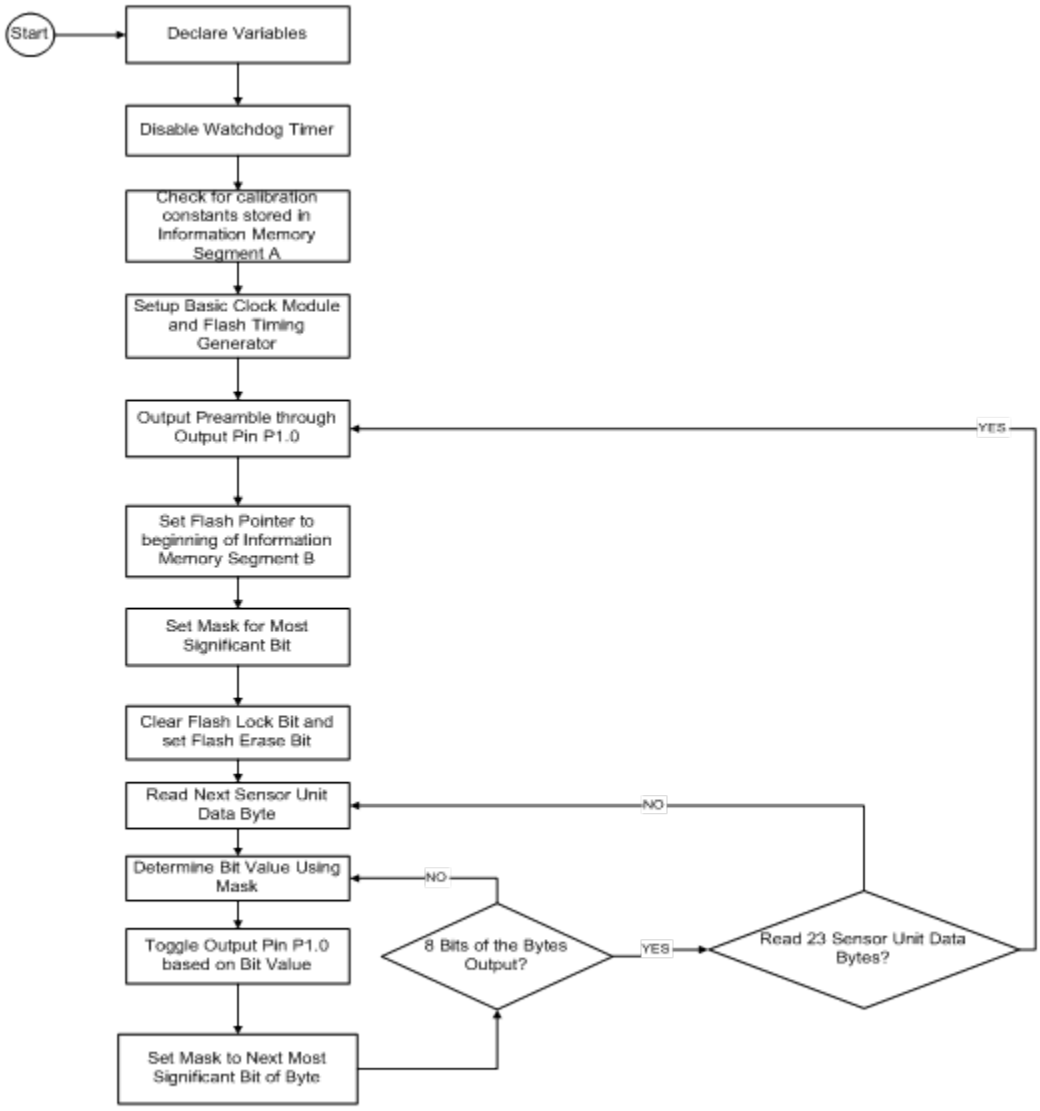


Figure 48: Flowchart for Data Transmission Program

5.1.4 Delay Addition for Collision Avoidance

The use of multiple independent Sensor Units within the system introduces a chance of collision. A collision is when two packets are sent from separate transmitters on the same channel to the same receiver causing the packets to overlap or *collide*. If this occurs, the overlapping data will be unintelligible and neither packet will reach the receiver without error. In this system, a collision would occur when two or more of the Sensor Units are transmitting to the same Receiver Unit simultaneously. This would mean two or more Sensor Units were released, rose to the surface while both were within range of the Receiver Unit at the same time. Given any significant difference in depth of the different Sensor Units, this should not be an ordinary situation. However, a solution was developed to account for this situation.

To give each Sensor Unit a strong probability of having its message accepted at the Receiver Unit (the Receiver Unit will ignore a message containing an error) the transmission algorithm was altered. For a given transmission to succeed, a time slot of at least twice the message length is needed. This message length will be denoted by T as shown in Figure 49 below. If Sensor Unit 2's transmission starts T seconds before Sensor Unit 1's transmission, the end of Sensor Unit 2's transmission will collide with the start of Sensor Unit 1's transmission. Also, if Sensor Unit 2's transmission starts anytime during the T seconds that Sensor 1 is transmitting the two transmissions will collide.

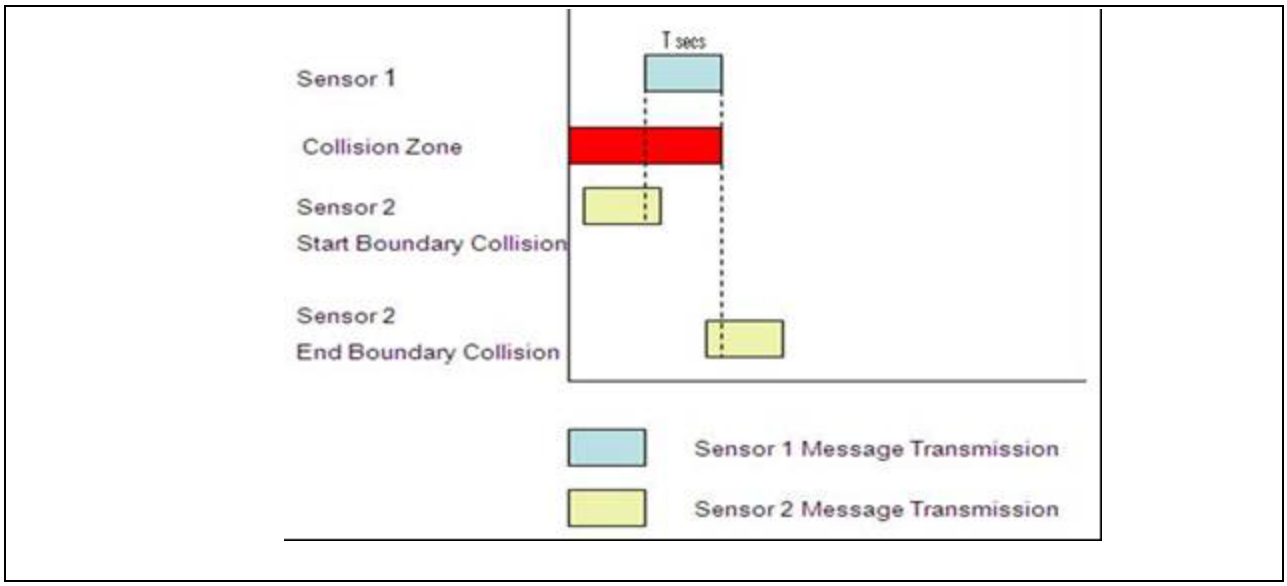


Figure 49: Collision Scenarios of Two Transmissions

If the Sensors were able to be scheduled to transmit at certain times, a delay of *T* could be added in between each Sensor Unit's transmissions and collisions could be avoided. A centralized synchronizing agent is required to schedule the transmissions. However, for this system, the time at which a Sensor Unit will begin transmitting cannot be scheduled as the unit will be released by natural events. The centralized synchronizing agent is not an option for this application. Even with a larger delay added between each Sensor Unit's transmissions, the collisions would reoccur if both Sensor Units have the same delay. Therefore, several delay lengths were added to the transmission algorithm. Each delay is a multiple of the *2T* transmission length of the message (*2T*, *4T*, *8T*, *16T*). The delay length chosen is based on a random number generated by the Sensor Unit. Figure 50 shows an example of a possible scenario using this algorithm. The figure shows an initial collision and then the two Sensor Units select on of the four delay options based on the random number they have generated. If the random numbers are different then each Sensor Unit will select a different delay and both

transmissions will get through to the Receiver Unit. In the example below the two Sensor Units would have a 25% chance of a collision occurring on their next transmission.

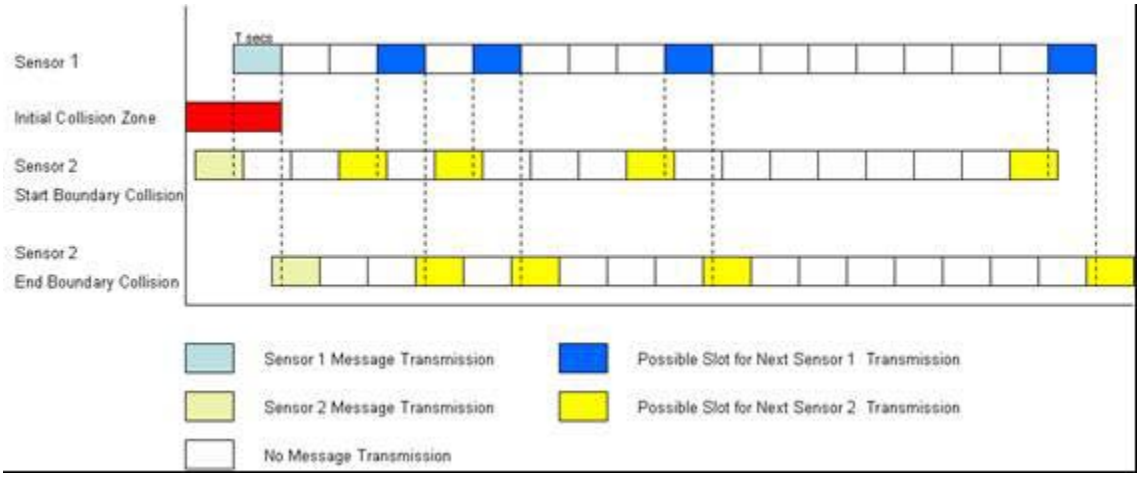


Figure 50: Collision Scenario with Delays Added

Additionally, the algorithm was altered so that if the Sensor Unit is programmed with a higher priority Color Code, it would transmit more frequently. For example, a Sensor Unit with a red Color Code would transmit the most frequently. In the following table lists percentages for the frequency each color would have of selecting a certain delay time.

Table 6: Delay Percentages for Each Color Code, T ≈ 100 ms

	Red Sensor Unit	Orange Sensor Unit	Yellow Sensor Unit	Green Sensor Unit
2T Delay	50.00%	37.50%	33.33%	25.00%
4T Delay	25.00%	37.50%	25.00%	25.00%
8T Delay	25.00%	12.50%	25.00%	25.00%
16T Delay	0.00%	12.50%	16.66%	25.00%

Figure 51, below, shows a Sensor Unit programmed with the red Color Code and a Sensor Unit programmed with a yellow Color Code. This oscilloscope screen shot shows how the red Color Code random delays are on average much shorter than that of the yellow Color Code random delays.

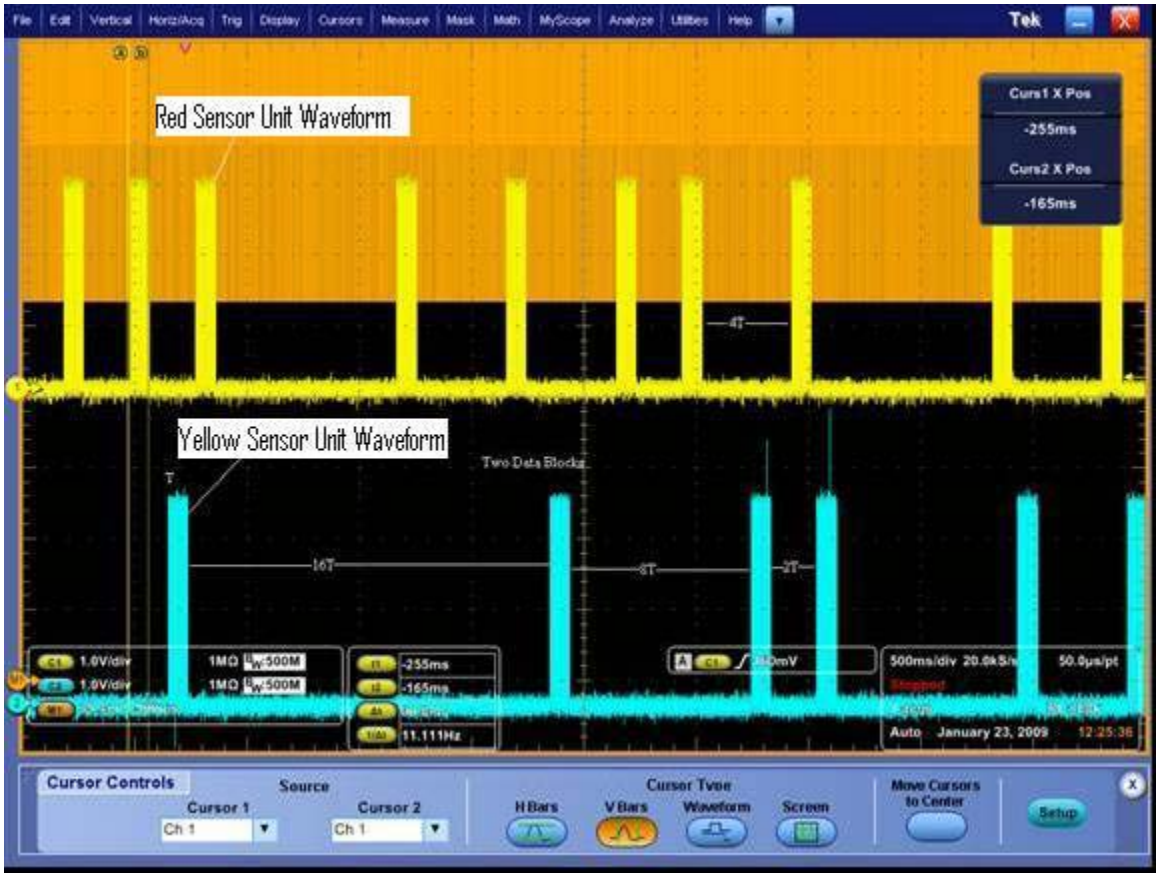


Figure 51: Oscilloscope shot of Red Priority Delays and Yellow Priority Delays

5.2 RECEIVER SOFTWARE

This Section will present the design of the software that will be run on the Receiver Unit's MSP430F2132 processor. This software has the tasks of receiving the data from the Sensor Units and driving the LEDs.

5.2.1 Receiver Software Description

Figure 52 shows the flowchart for the program that receives the Sensor Unit transmission as well as controlling the LED I/O lines. The program begins by defining the CRC polynomial that will be used to check if the received Sensor Unit Data Block is correct. It also defines the preamble, which is 0xAA. The preamble will precede the Sensor Unit Data Block. The program then defines several global variables that will be shared by the function and the interrupt service routines (ISRs).

Next, the program disables the Watchdog Timer of the MSP430. The purpose of the Watchdog Timer is to perform a controlled system restart after a software problem has occurred. This is based on a specified time interval set in the Watchdog Timer registers. On initial power up, this interval is 32768 cycles [16]. If the system is restarted during flash operations, unexpected results could occur. For this reason, the Watchdog Time is disabled.

MSP430 devices come with calibrated Basic Clock Module settings for specific frequencies stored in information memory segment A [16], [24]. The next part of the program will check to ensure this memory is intact within information memory. Unless the processor has been corrupted or the information memory has been purposely erased, this should not be the case. These stored settings are then used in the next step to setup the registers of the Basic Clock

Module. The other register that is set up is the Flash Timing Generator. The Flash Timing Generator controls erase and write operations. It has a particular operating frequency that must be used. For this processor the frequency is 333 kHz. To achieve this operating frequency, clock dividers settings in a Flash register must be specifically set

At this stage, I/O port P1, P2, and P3 are setup for correct operation. Specifically, P1.2 must be an input and P2.3, P2.4, and P3.7 must be outputs. Timer A is then set up to run off the sub-main clock in up mode. This means that the timer will produce an interrupt every specified number of clock cycles. The number of clock cycles is stored in register TA0CCR0. The incoming data will arrive at 10000 Hz. Given the 8 MHz clock, this means an interrupt should be created every 800 clock cycles. The flash pointer is then initialized to the starting address for the storage of the data. The other variables are initialized to 0 except the mask, which is initialized to 4 in order to operate with third least significant bit (LSB) of P1IN.

The program now enters a continuous while loop. The loop will do nothing until mode 2 is reached. The loop is entered in mode 0 and mode 2 is only achieved through the interrupt service routine (ISR). The ISR will be entered every 800 cycles. Within the ISR, the logic level of the signal on pin P1.2 is determined using the mask. The value is then shifted into the variable holding the last 8 data bits received. Following this, there is a check to see if the program is in mode 1. Mode 1 means the program has received a correct synchronization pulse and will store the next 25 bytes of data to an array. After the 25 bytes of data have been received, the program will be placed in mode 2.

Once in mode 2, a conditional is entered within the main function's while loop. Within this conditional *CRC_check()* called first. *CRC_check()* will perform the same methodology as the function *CRC_calculation()* from the Sensor_Data_Write.c. The only

difference between this and the version used for the Sensor Unit is that instead of using only the first 21 bytes of the Sensor Unit Data Block to calculate a remainder, the entire 23 bytes of the Sensor Unit Data Block will be used. Using the same polynomial, if the message is error free, the remainder produced by dividing the Sensor Unit Data Block by the polynomial will be zero. Following this, a function *BridgeID_Repeat()* is called to check two factors of the message transmitted. This function checks if the bridge ID of the of the message matches the bridge ID stored in the Receiver Unit. This bridge ID must be input through use of the UART. The function also checks to ensure the message is not simply a repeat. If these two checks pass, the *store_and_process()* function will be called within the main function. If the CRC, bridge ID, or repeat check come back incorrect the message will be discarded. In either case, the mode is restored to mode 0 and the conditional is then left.

The function *store_and_process()* has the task of storing the 23 byte Sensor Unit Data Block to flash memory and driving the LED lines based off the value of byte 20 of the Sensor Unit Data Block. The function will take the array of the current 23 bytes of data and begin storing it at the next available flash location. When the 20^{th} byte is encountered, it will check its value. If the value is 0, 1, 2, or 3 it will drive the corresponding green, yellow, orange, or red I/O line high respectively. The other three lines that are not driven high will be driven low. Additionally, once a state of higher priority has been reached, a lower state cannot be activated.

Once the conditional is left, the program will repeat this process infinitely.

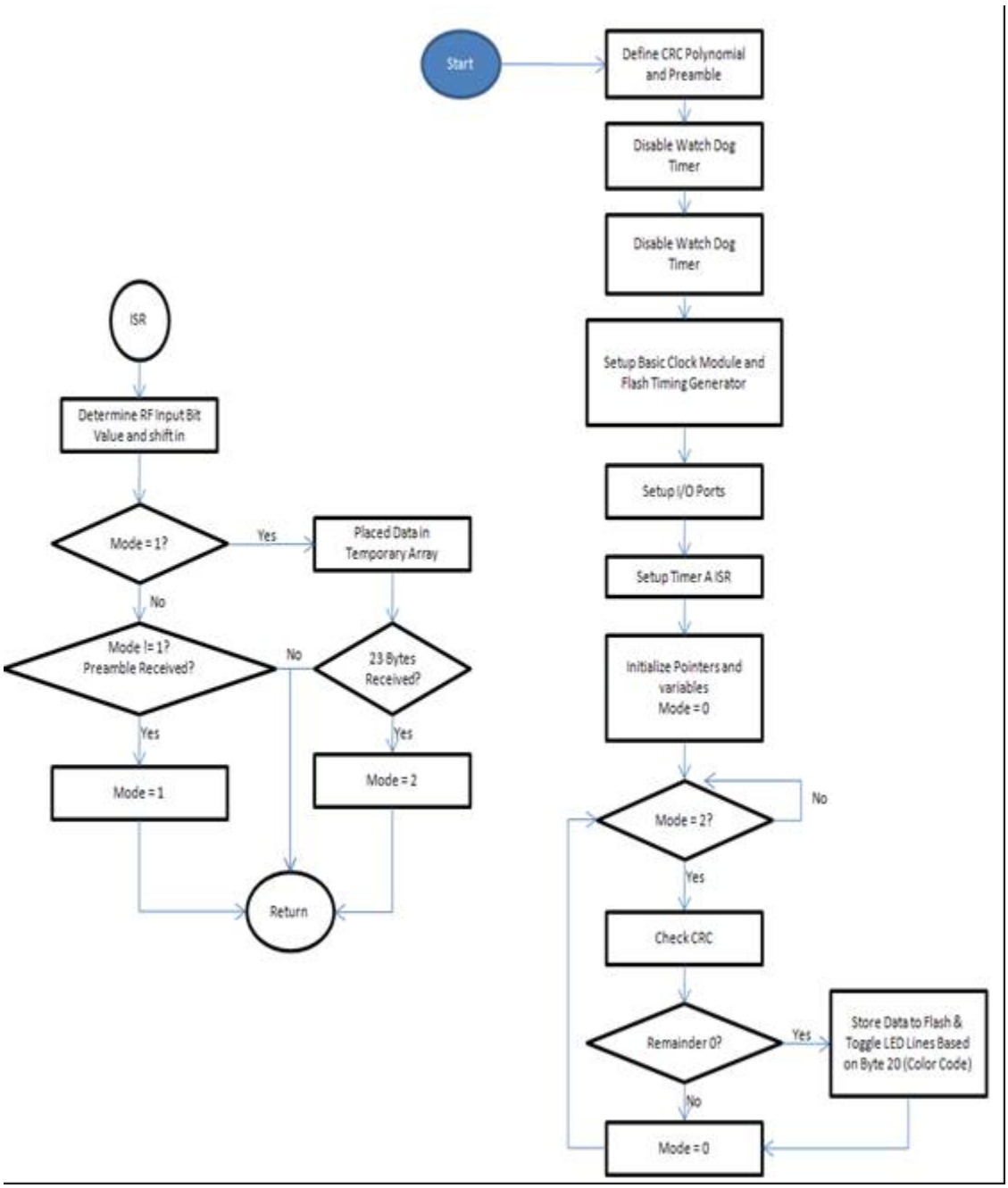


Figure 52: Flowchart for Receiver Program

5.3 SOFTWARE UTILITY SETUP AND USE

Three software utilities are used with the system. The first is Code Composer Essentials v3 Core Edition. This is used for the medication of program code as well as debugging of the system. Another tool used is a flash programmer FET-Pro430. This tool simply takes the final output file created from the program code and downloads it to the microcontrollers. The other is HyperTerminal™. This is used for the UART communication. The three utilities will be discussed in the following subsections

5.3.1 Code Composer Essentials v3.1 Core Edition

The Code Composer Essentials v3.1 Core Edition utility is made by Texas Instruments for the programming and debugging of MSP430 microcontrollers. It utilizes a user-friendly eclipse style interface. The use of this tool within this remote scour detection system is to modify program code and debug the microcontrollers on the Sensor and Receiver Units. This utility uses the USB/JTAG connector MSP-FET430UIF to interface with the 14 pin JTAG header on the Sensor and Receiver boards.

The program can be obtained at no cost directly from the Texas Instruments website at the following link: http://focus.ti.com/docs/toolsw/folders/print/msp-cce430.html. The program is to be ran on either a Windows XP or Windows Vista Operating System. The tool has up to 16 KB of code space available for a given download. The bridge scour detection system does not approach this limit. Once downloaded, follow the setup steps presented by the installer to have the utility set up on the host PC or laptop.

5.3.2 FET-Pro430

The software tool FET-Pro430 is a flash programmer made for the MSP430 devices. The programmer is made by Elprotonic Inc. and can be freely downloaded at http://www.elprotronic.com/download.html. Although Code Composer Essentials is also able to download program code to the microcontroller, this application would simplify the necessary action of a technician once the system is in a production phase. This program has a simple GUI interface and can take a .hex output file of the source code and download it to the flash memory of the microcontroller without any of the debugging features of a IDE like Code Composer Essentials.

Once the FET-Pro430 is installed on the machine the executable can be run. The interface is as shown in Figure 53 below. A couple of settings must be established the first time the tool is run. First, the **Group** pull down in the Microcontroller Type section of the GUI must be changed to **MSP430F2xx** . Under this, the **MSP430F-** pull down must be changed to **2132**. Then, under the **Power Device from Adapter** section, the pull down should be change to **3.0 V**. Lastly, a setting should be changed in the Connection / Device Reset menu. This menu can be reached by clicking on Settings at the top of the GUI and clicking on Connection / Device Reset. A window will appear as shown in Figure 54. Make sure the JTAG is selected in the Communication with Target Device section. Also ensure USB is selected under the COM port section and click OK. Once these settings are established, the file to be downloaded should be selected. This can be done by clicking the Open Source File button near the top left of the interface. Once a .hex file has been selected, the microcontroller can be programmed by clicking the AUTOPROGRAM button. The result should look as shown in Figure 53.

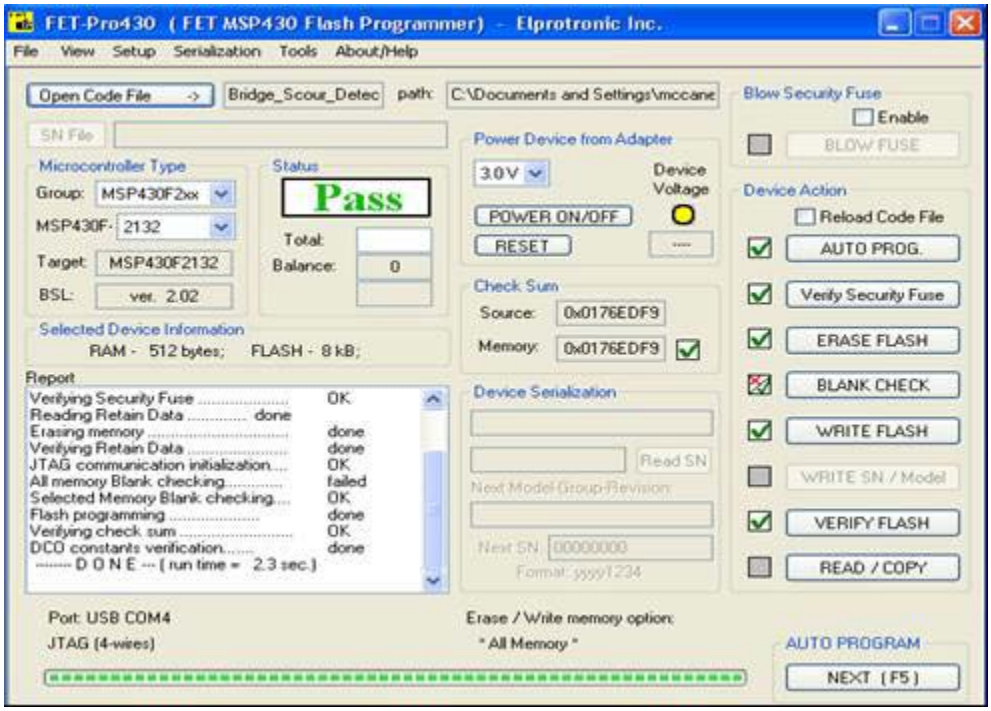


Figure 53: FET – 430 Interface

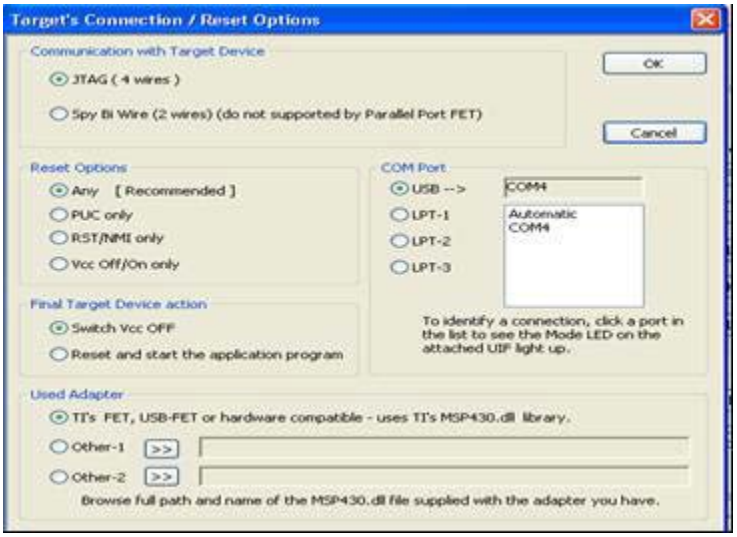


Figure 54: FET - 430 Settings

5.3.3 HyperTerminal

HyperTerminal™ is a utility that comes pre-installed on all Windows Operating Systems. It is a communications application. The current system will utilize it for communication with the UART of the microcontroller. For convenience, a USB connection will be used as an intermediary between the host computer and the UART of the microcontroller. A Serial to USB chip will convert between the USB connection and the UART connection. Using the USB within the HyperTerminal requires installation of a Virtual Com Port (VCP) made to interact with the Serial to USB chip. Using a VCP, the USB will show up within HyperTerminal as regular COM port. The VCP drivers can be found freely available at: http://www.ftdichip.com/Drivers/VCP.htm. Download the latest version of the VCP driver and install.

Once the VCP drivers are installed the HyperTerminal™ can be setup. The HyperTerminal™ can be run by going to **Start** within the Windows Operating , clicking on **Run..**, and typing 'hypertrm'. Any name and icon will suffice. On the next prompt the COM port will be chosen. The COM port used should correspond to the VCP installed. Choose this COM port next to the Connect Using line while leaving all other setting as is. Once the Hyper Terminal is setup, follow the UART protocol described in the next section

5.4 SERIAL INTERFACE PROTOCOL

A protocol is necessary to invoke the specific functionality desired by the technician when using UART communication. The protocol for interaction with the Sensor Unit and Receiver Unit is described here.

5.4.1 Sensor Unit UART Protocol

This section details the protocol used for interface with the Sensor Unit microcontroller. Below the available functions are listed along with how to invoke them.

Send Sensor Unit Data Block Entry- The Sensor Unit Data Block is composed of 21 bytes of data. This is 168 bits of data. To transfer this data, the data string will be represented 42 hexadecimal (hex) characters (base 16, 0-9 A-F). Each hex character represents four bits of information. The first two hex characters represent the first byte (8 bits make up one byte) of the Sensor Unit Data Block, the next two hex characters represent the second byte, and so forth. Once the 42 hex characters have been formed they can be transferred to the Sensor Unit by two means.

1. Text file transfer (Shown in Figure 56)
    a. Place the 42 hex characters proceeded by two asterisks (**) at the beginning of a file.

b. Save the file as a .txt file type (text file). A simple text editor such as Notepad or WordPad can be used.

c. Within HyperTerminal™ click Transfer→Send Text File.

d. Choose the .txt file containing the Sensor Unit Data Block, this is illustrated in Figure 56.

e. The file will be transferred to the Sensor Unit and "data_txed" will be displayed in the HyperTerminal™ window.

2. Character by Character Entry

a. Enter the asterisk character ('*') two times in a row within the HyperTerminal™ interface. This notifies the microcontroller that the next 42 hex characters are the Sensor Unit Data Block and should be stored as such.

b. Enter the Sensor Unit Data Block one hex character at a time.

Sensor Unit Data Block Read Out - Verification of the Sensor Unit Data Block entered can be done simply by entering # at any time during the interface. The data entered concatenated with the two bytes of CRC will be displayed. The data block will be displayed as a sequence of hex characters. This is shown in Figure 55.

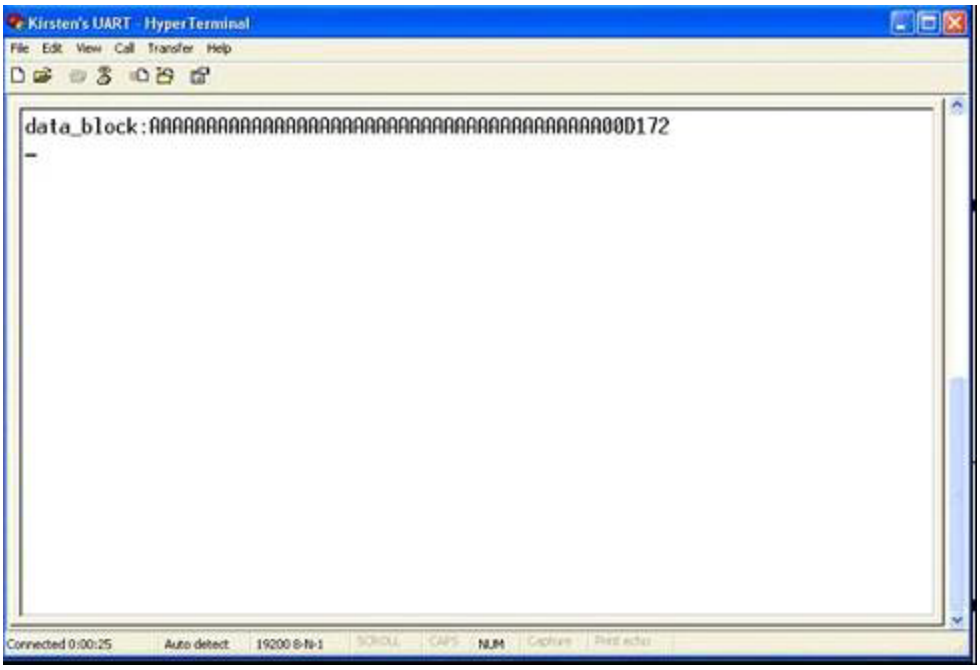


Figure 55: Sensor Unit Data Block Read Out

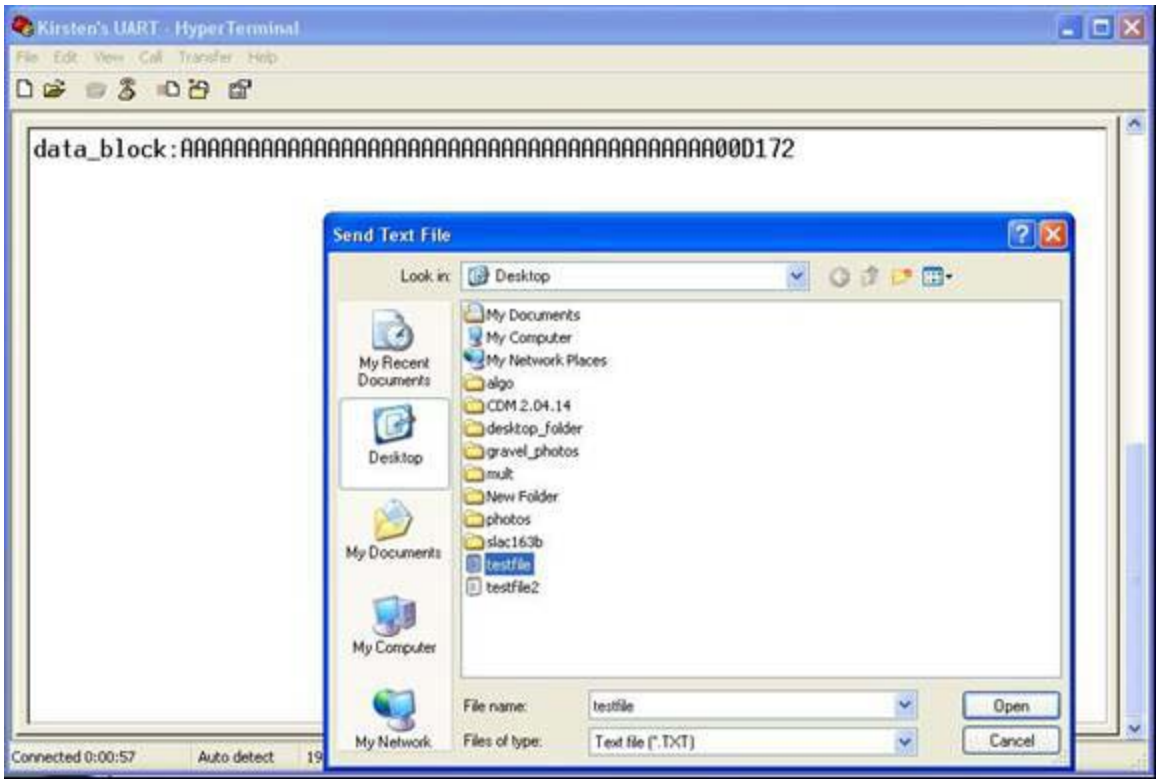


Figure 56: Transmitter UART Communication: Transfer of Data Block from txt file

5.4.2 Receiver Unit UART Protocol

This section details the protocol used for interface with the Receiver Unit microcontroller. Below the available functions are listed along with how to invoke them.

Bridge ID Entry – The Receiver Unit will have a stored Bridge ID to compare with the Bridge ID contained within the Sensor Unit Data Block transferred. The Bridge ID portion of the Sensor

Unit Data block is composed of 14 bytes of data. This is 112 bits of data. To transfer this Bridge ID to the microcontroller, the data string will be represented 28 hex characters (base 16, 0-9 A-F). The first two hex characters represent the first byte of the Bridge ID; the next two hex characters represent the second byte, and so forth. Once the 28 hex characters have been formed they can be transferred by two means.

1. Text file transfer
    a. Place the 28 hex characters proceeded by two asterisks (**) at the beginning of file.
    b. Save the file as a .txt file type (text file). A simple text editor such as Notepad or WordPad can be used.
    c. Within HyperTerminal™ click Transfer→Send Text File. This is the same procedure as step 1.c for the Sensor Unit serial interface.
    d. Choose the .txt file (text file) containing the Bridge ID. This is the same procedure as step 1.d for the Sensor Unit serial interface.
    e. The file will be transferred to the Receiver Unit and "data_txed" will be displayed in the HyperTerminal™ window. This is the same procedure as step 1.e for the Sensor Unit serial interface.
2. Character by Character Entry
    a. Enter the asterisk character ('*') two times in a row within the HyperTerminal™ interface. This notifies the microcontroller that the next 28 hex characters are the Bridge ID and should be stored as such.
    b. Enter the Bridge ID one hex character at a time.

Bridge ID Read Out - Verification of the Bridge ID entered can be done simply by entering # followed by a 1 at any time during the interface. The Bridge ID will be displayed as a sequence of hex characters.

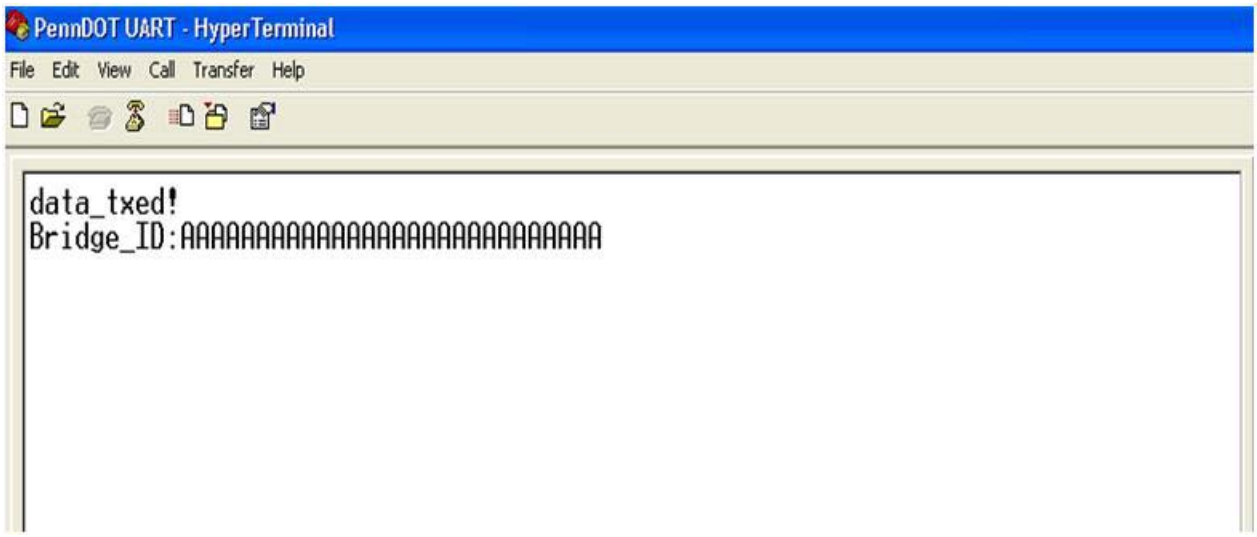


Figure 57: Bridge ID Transfer and Read Out

Sensor Unit Data Blocks Received Read Out - When a Sensor Unit Data Block is accepted by the Receiver, it is stored into the flash memory of the Receiver Unit microcontroller. Each block received is stored consecutively after the last. Entering # followed by a 2 at any time during the interface will result in a read out of all Sensor Unit data blocks received. The blocks will each be placed on a new line with the first block read out corresponding to the first block received by the Receiver Unit.

Receiver Unit Reset - The state of the Receiver Unit is based on the color code of the Sensor Unit Data Blocks received. To reset this state to the default starting state, two ampersand

characters (&&) should be entered consecutively. A “system reset” statement will be displayed in the HyperTerminal™ window upon doing this.

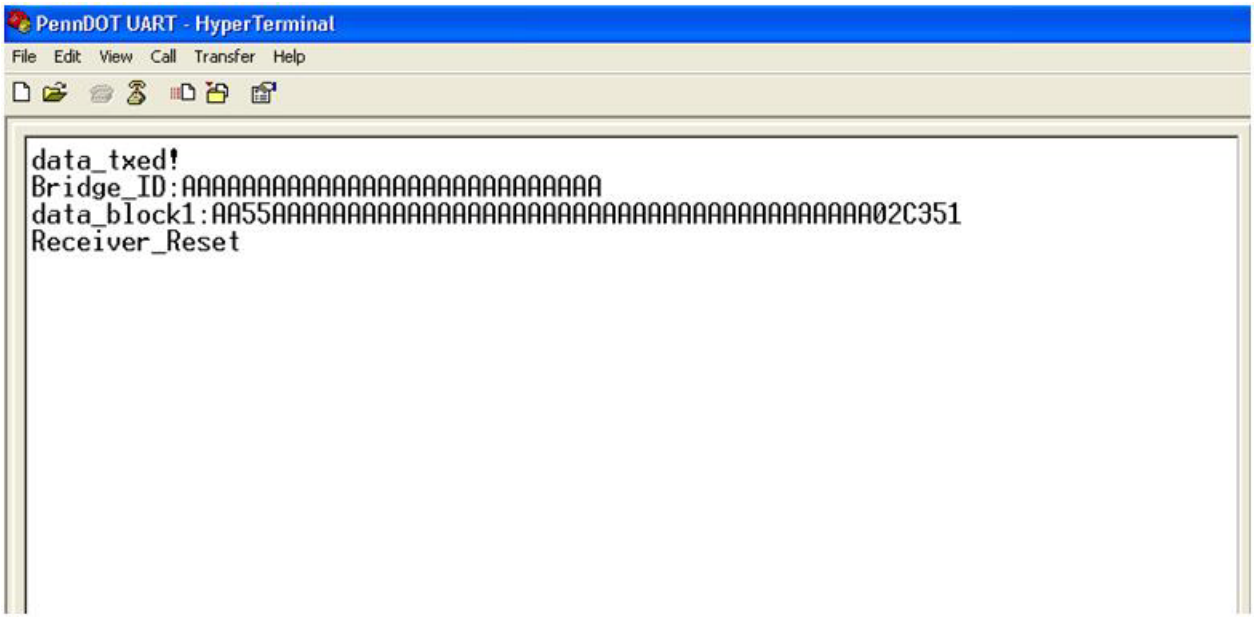


Figure 58: Transferred Data Block Read Out and System Reset

6.0 TETHER OPTION ANALYSIS

This section describes the tethered and un-tethered options for the Sensor Unit. The decision to tether or un-tether the Sensor Unit will be based primarily on the ability of the Sensor Unit to rise to the surface and transmit the message before going out of range of the Receiver Unit on the bridge. The time required for the Sensor Unit to reach the surface is determined by the buoyancy force of the Sensor Unit, the drag on the Sensor Unit as it rises to the surface, and the distance (depth) from the senor unit to the surface of the water (river/stream).

The following section details how the Sensor Unit rise time was computed. The following sub-sections detail the pros and cons of the tethered and un-tethered options respectively. Based on the results the un-tethered option will be used.

6.1 TRANSMITTER –RECEIVER RANGE

The Linx 433 MHz transmitter-receiver pair have a free space range of 3000 feet [17], [26]. This value will be used as an upper limit for the range in which a message can be transmitted and received. However, the conditions in which the system operates may differ greatly from those that produced the 3000 feet limit. These conditions include less than maximum transmitting power, as well as antenna and orientation factors.

The Friis equation is used to determine the distance between the transmitter and receiver (range),

$$P_r = \frac{G_t G_r P_t \lambda^2}{(4\pi)^2 d^2} \quad (1)$$

where P_r is the received power, P_t is the transmit power, G_r is the receiver antenna gain, G_t is the transmitter antenna gain, d is the distance between the transmitter and the receiver, and λ is the wavelength of carrier frequency. To simplify, both receiver and transmitter antenna gain are set to one. The transmit and receive power ranges specified by the radio frequency (RF) transmitter and receiver datasheets are used for P_t and P_r [17], [26]. The wavelength for a 433 MHz carrier frequency is 0.7 meters. Using these parameters equation (1) is solved for the distance, d.

Given the range of output power, and receive power, the resulting range of distances was calculated. The worst case distance calculated was 731 feet.

6.2 SENSOR UNIT RISE TIME ANALYSIS

The calculated rise-time of the Sensor Unit from a given depth is presented in this sub-section. The rise-time calculations and assumptions are presented, followed by a description of the simulation process used to determine the rise-time. A simulation was used to accurately model the drag force on the Sensor Unit. This sub-section concludes with a presentation of the calculated rise-times for a variety of depths, water velocities, and Sensor Unit dimensions (the radius of cylinder and the length of cylinder).

6.2.1 Buoyancy Calculation

The rise-time calculation is based on a combination of buoyancy and drag forces. The buoyant force must be greater than the force of gravity on the object in order for the object to rise. This is expressed as follows where a positive force exerts a downward (in the direction of gravity) force on the object,

$$F_{net-buoyant} = mg - \rho Vg \tag{2}$$

where, m is the mass of the object, g is the acceleration due to gravity, V is the volume of the object and p is the density of the fluid the object is in. The first term in (2) is the force due to gravity and the second term is the buoyant force of the object in a fluid having density p.

The capsule will be constructed out of Poly-Vinyl-Chloride (PVC) pipes. In order to compute the mass of the capsule the density of PVC will be multiplied by the volume of PVC pipe used to make the capsule; the mass of the electronics is neglected here. The capsule will have a thickness of 0.00256 m (0.1 inches) which is the minimum thickness of PVC [28]. The density of PVC is 1380 kg/m^3 [29]. The surface area, A_S, of the capsule is found using Equation (3),

$$A_S = 2\pi r^2 + 2\pi rh \tag{3}$$

where *r* is the radius of the cylinder, and *h* is the height (or length) of the cylinder. The volume of the capsule is computed as follows,

$$V = 2\pi r^2 h \tag{4}$$

where *r* is the radius of the cylinder, and *h* is the height (or length) of the cylinder. Lastly, the acceleration due to gravity has a value of g = 9.78045 m/s^2, and the density of water is 1000 Kg/m^3. Together, these values can be inserted into the buoyancy formula presented above to produce a net force.

6.2.2 Drag Calculation

While buoyancy produces a net force for a given object within a fluid (water in this case), the net force will change depending on the drag force incurred by the object while moving through the water. There are two types of drag forces that can be encountered by the object. At faster velocities, the drag force can be determined using the Quadratic Drag equation [29], [30]. At slower velocities, viscous forces dominate and drag force takes the form of viscous resistance [29], [30]. To determine which drag force is appropriate for a given set of circumstances, a value known as the Reynolds number is used.

6.2.2.1 Quadratic Drag

Quadratic drag is typical for objects moving through a fluid at a high velocity. The quadratic drag depends on a number of factors and is calculated using Equation (5),

$$F_d = \frac{AC_d \rho V^2}{2} \quad (5)$$

where F_d is the force due to drag, C_d is the drag coefficient, p is the density of the fluid (water in this case), and A is the cross-sectional area of the surface moving through the fluid. The drag coefficient is a unit-less number found by experimentation and a value of 1.17 is used for this study [29]. The cross-sectional area is the area of the top of the cylinder, or a circle of radius r. The velocity of the object changes as the object rises to the surface, requiring F_d to be recomputed. To effectively model the velocity of the capsule a simulation was developed which steps through time in small discrete units (1 milli-second) updating the velocity and computing F_d for each unit of time. Using smaller discrete units of time i.e., 1 micro-second showed changes only in the fifth or more significant digit, thus, a 1 milli-second time-step was used.

6.2.2.2 Linear Drag

Linear drag occurs when the viscous force of the fluid is the dominate opposing force and is used for slow moving objects. Linear drag is computed using Equation (6),

$$F_d = 6\pi\eta r v \tag{6}$$

where, η is the viscosity of the fluid (water in this case), r is the Stokes radius (radius of the capsule in this case), and v is the velocity of the object. The viscosity of water is 1.12 g/(m*s) [29]. The Stokes radius will be set to the radius of the cylinder. Linear drag, just as quadratic drag, depends on the velocity of the object. Again, a simulation model was developed.

6.2.2.3 Reynolds Number

The Reynolds number (Re) is used in the calculation of the drag coefficient and to characterize fluid flow conditions. Generally, larger Reynolds numbers indicate that the quadratic drag equation should be used whereas lower numbers indicate that the linear drag equation should be used. Reynolds numbers up to 100 have the characteristics appropriate for the linear equation with Re < 0.1, being the more common boundary [29], [31]. The Reynolds number can be computed for a cylindrical object using the following equation,

$$\mathrm{Re} = \frac{v_s d}{\nu} \tag{7}$$

where v_s is the velocity of the object, d is the diameter of the cylinder, and v is the absolute kinetic viscosity of the fluid the object moves through.

The Reynolds number for diameters of 2 inches and 4 inches were computed for cylinders of the sizes and water velocities being investigated. The results show that the Reynolds numbers are larger than the boundary point defined in [29] and [31], hence the quadratic drag equation should be used. The simulations will be run using both the linear and quadratic drag equations to compute the rise time and the larger of the two times will be used. **Table 7** shows the Reynolds numbers for a representative set of velocities and capsule sizes.

Table 7: Reynolds number for diameters of 2 and 4 inches.

Velocity	Diameter = 0.05 meter (2 inches)	Diameter = 0.1 meter (4 inches)
10 m/s (33 ft/s)	**4.98E+05**	**9.96E+05**
1 m/s (3.3 ft/s)	**4.98E+04**	**9.96E+04**
0.1 m/s (4 inch/s)	**4.98E+03**	**9.96E+03**
0.01 m/s (0.4 inch/s)	**4.98E+02**	**996.0159**

6.2.3 Simulation Method

A simulation was developed in MATLAB to calculate the velocity and then the drag force of the Sensor Unit as it rises to the surface. As the velocity of object increases, the drag force will also increase.

The simulation divides time into discrete units (1 milli-second units) and computes the velocity and drag force for that unit of time. Then the simulation moves onto the next unit of time, until the Sensor Unit reaches the surface. The net force is computed as follows,

$$F_{net} = F_{net-buoyant} - F_d \tag{8}$$

where $F_{net\text{-}buoyant}$ is the net force due to buoyancy (upward) and F_d is the force due to drag (downward).

The first program, constants_for_rise_time_calcs.m, is used to setup the variables needed for the calculations. The inputs to this program are the radius and length of the Sensor Unit capsule. The variables for density water, gravity, the volume displaced by the object, the drag coefficient, the cross-sectional area of the object, and a starting depth and initialized or calculated within this program.

The second program, calc_rise_time.m, takes these variables as inputs along with a time interval, delta_t, for its computations. For this experiment a delta_t of 1 milli-second was used. First the initial F_{net} is computed with the velocity set to zero. If F_{net} is less than or equal to zero, then the Sensor Unit is not buoyant and will not float to the surface, and the program will report an error and exit. However, if F_{net} is greater than zero the Sensor Unit is buoyant and will float to

surface, and the program begins to loop (step) through time in steps of length delta_t (1 milli-second).

Within this loop, acceleration is first calculated from the net force calculated during the last iteration of the loop or the initial net for in the initial case. Given this acceleration, and the velocity of the last iteration, a new velocity is calculated for the next time interval. Using this velocity and the position of the Sensor Unit, the program then calculates the position of the Sensor Unit at the start of the next time interval. Next, the drag force is calculated from the velocity of the object at that time. Either the linear or quadratic drag equation can be used to compute F_d. Two programs one for the linear drag and the second for quadratic drag were developed and used. The version of the program that performs linear drag is calc_rise_time_viscous.m. This simply replaces the quadratic equation with the linear one. The net force on the Sensor Unit, F_{net}, is computed as shown in Equation (8). Finally, the total time, current position, and current velocity variables are updated for the next iteration. Once the current position (vertical distance traveled) is larger than the depth the Sensor Unit was buried, the total time calculated is considered rise time.

6.3 RISE TIME RESULTS

The rise time results presented in Section 6 show rise times for a large variety of Sensor Unit sizes and depths. The results computed using the quadratic drag equations are consistently larger than those computed using the linear drag equations. Because the critical factor is to verify if the

Sensor Unit will surface within range of the Receiver Unit and based on the computed Reynolds numbers the results generated using the quadratic drag equation will be used.

6.3.1 Distance Traveled While Transmitting

The distance traveled by the Sensor Unit downstream is the primary concern for the tether/un-tethered decision. Given the time it takes the Sensor Unit to rise and the velocity of the stream, this distance can be determined. However, it will also take the Sensor Unit some amount of time to transmit the message once it has surfaced.

The river currents that will be used will be based off stream flow predictions made in 2002, estimating a maximum water velocity of approximately 23 ft/s [31]. A water velocity of 33 ft/s will be used in this evaluation to allow for a conservative safety factor.

The time required for the Sensor Unit to transmit the message depends on the rate at which the data is transmitted, *b*, and the number of bytes (length) of the message, *L*,

$$t_{msg} = 8bL \quad [4] \tag{9}$$

The amount of data in the Sensor Unit data block is 23 bytes. The maximum data transfer rate, *b*, is 10 Kbps (kilo-bits per second). With these two values, the time it takes the transmitter to complete the transfer of one message can be determined. Table 8 shows the time it takes for this transfer at the maximum 10 Kbps and a slower 1 Kbps for the slow water velocity of 3 ft/s and the extreme water velocity of 33 ft/s.

[4] 1 byte contains 8 bits

Table 8: Message Transmission Time and Distance Traveled

Data Rate, *b*	10 Kbps	10 Kbps	1 Kbps	1 Kbps
Water Velocity	3 ft/s	33 ft/s	3 ft/s	33 ft/s
Time to Transmit (seconds)	0.0184	0.0184	0.184	0.184
Distance Traveled While Transmitting (feet)	0.0552	0.6072	0.552	6.072

Table 8 shows that the distance traveled while transmitting a message is (0.0184 feet to 6.072 feet). Therefore, the distance traveled by the Sensor Unit will be primarily based on the distance traveled while the sensor rises. The following three figures show the rise times for different capsule (cylinder) sizes and water velocities; 3 ft/s and 33 ft/s. The following figures show the distance away from the bridge that the Sensor Unit surfaces under different depths and different water velocities. Three different Sensor Unit capsule sizes are investigated. The minimum range is a rough approximation of the transmitter-receiver range under the worst-case conditions, the actual range in practice is at least 3000 feet.

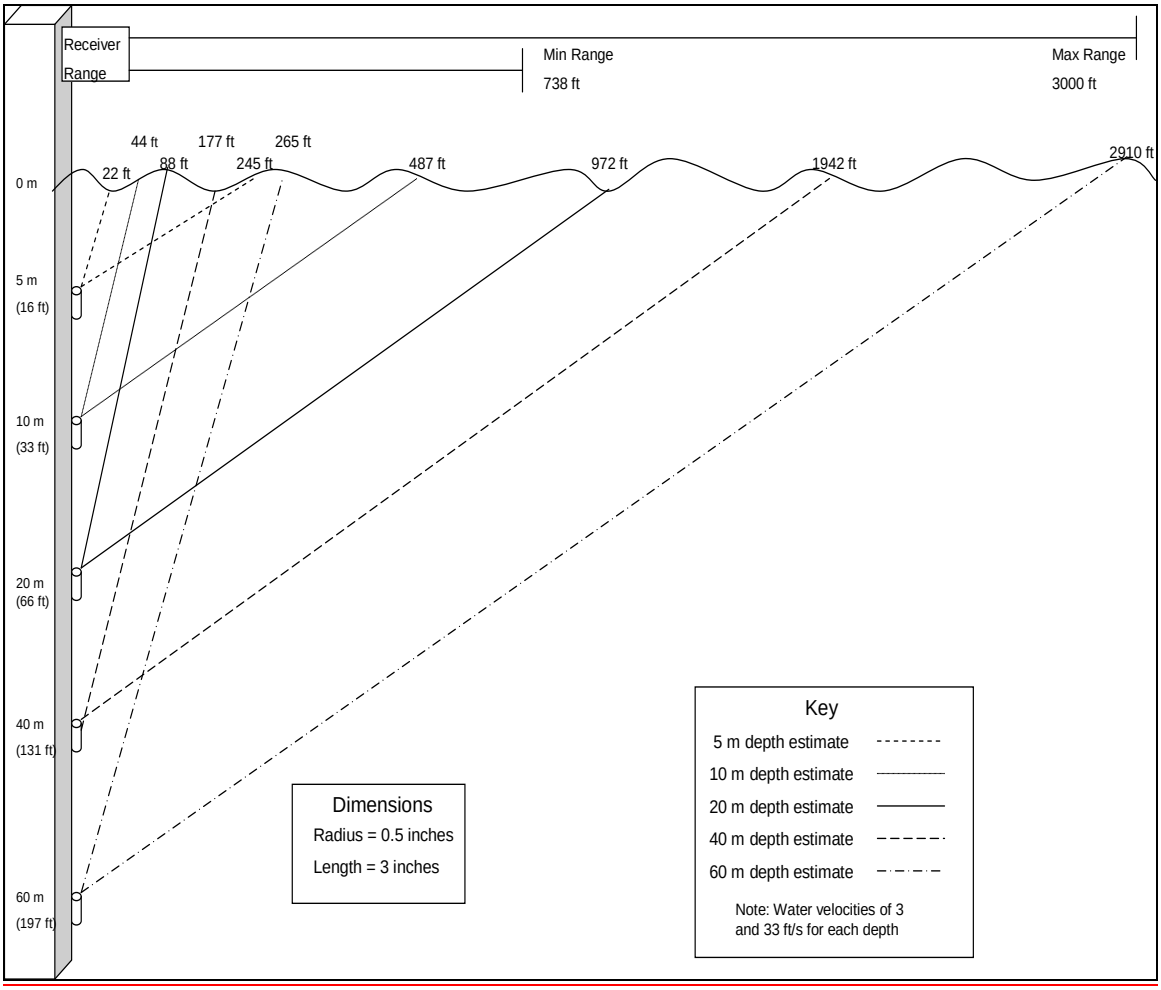


Figure 59: Travel distances for capsule with a radius of 0.5 inches and a length of 3.0 inches.

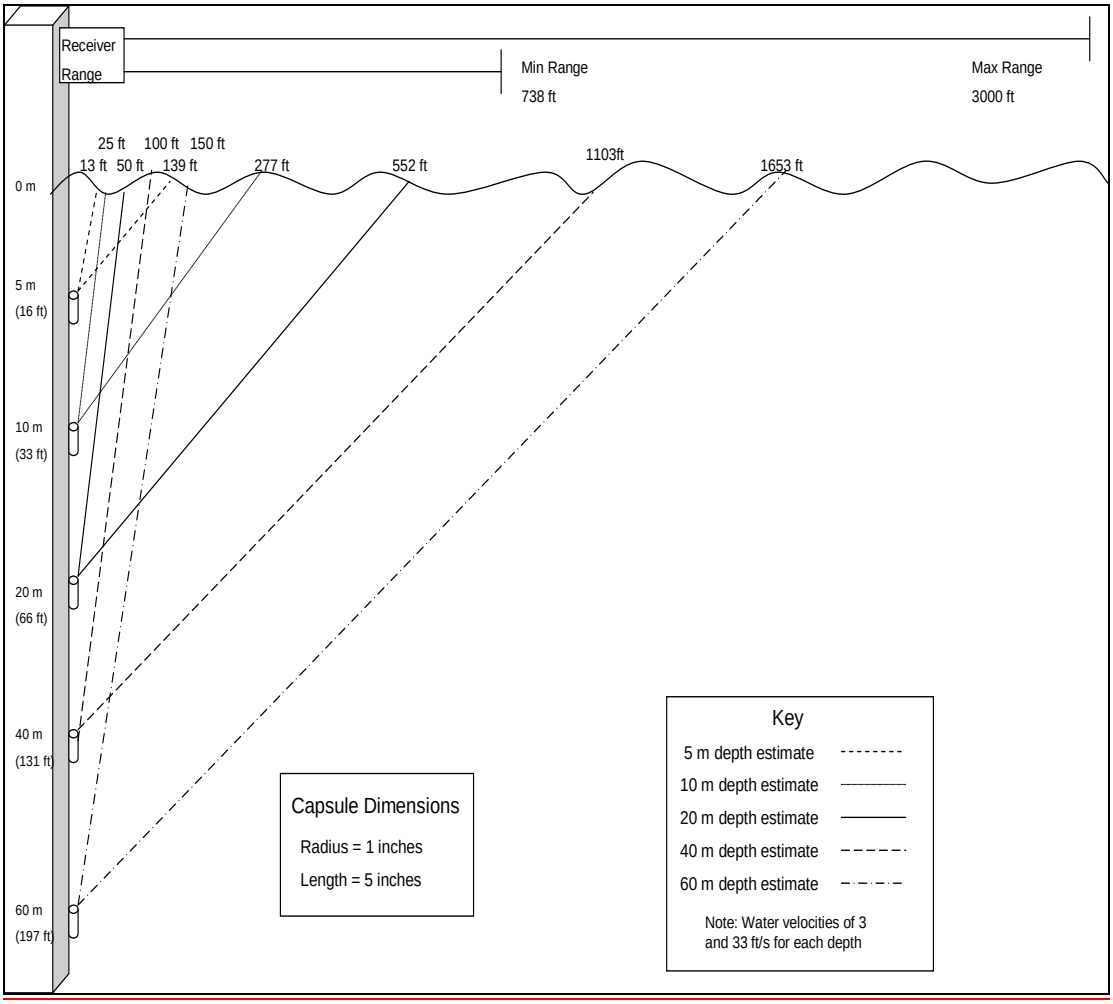


Figure 60: Travel distances for capsule with a radius of 1.0 inch and a length of 5.0 inches.

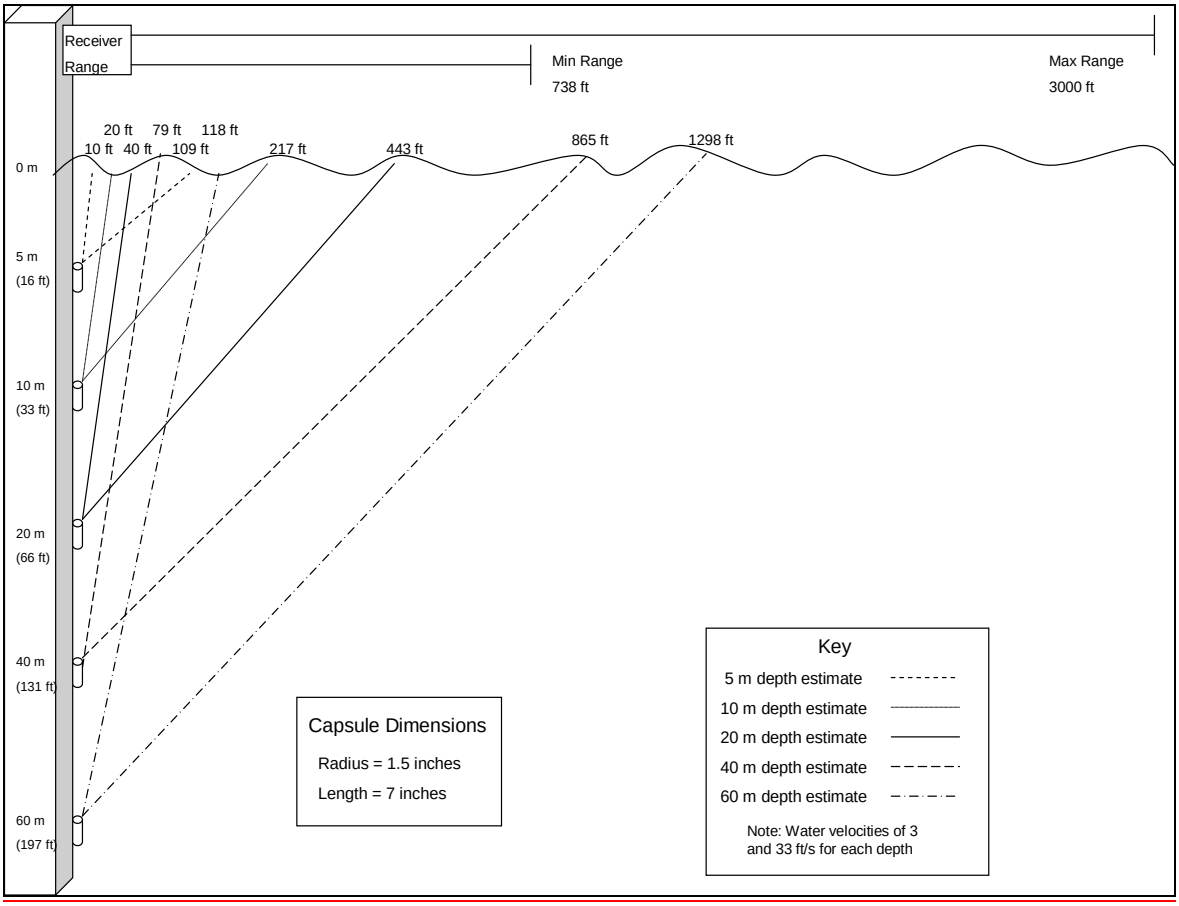


Figure 61: Travel distances for capsule with a radius of 1.5 inches and a length of 7.0 inches.

6.4 TETHERED OPTION

The tethered option for the system calls for all Sensor Units to be anchored to the foundation area in order to limit their distance from the Receiver Unit. The reason for this implementation is to ensure transmission of the Sensor Unit data. However, the tethered option introduces complexity for the installation and release of the Sensor Unit from the foundation. A possible

implementation of this option is discussed in the first subsection. Additionally, the full range of pro and cons are summarized in the second subsection. For this option, the primary issue is substantiating that the added complexity of system is warranted and necessary for proper system operation.

6.4.1 Description of Tethered Option Solution

Implementation of the tethered option requires two additional requirements to be met. First, the anchor must be placed in the foundation of the bridge in such a way and at a depth that it will remain in that position permanently. Second, the anchor must be connected to the Sensor Unit in such a way that the Sensor Unit can rise and activate properly while being restricted to a given area. Given these requirements, the solution deals with the installation and deployment of the Sensor Unit under this option.

For installation, the Sensor Unit must be installed through the use of standard NX (3-3/16 inch ID) hollow stem augers. Due to this size restraint, the anchor must be thin and simple. Therefore, to ensure the anchor will not be moved is to make it sufficiently heavy and buried deeply. If the anchor is buried below depths susceptible to scour, the only force pulling on the anchor should be the deployed Sensor Unit. Using the 33 ft/s maximum water velocity, the floating deployed Sensor Unit would produce maximum possible accelerations of 33 ft/s^2. This is only slightly larger gravity. Therefore, given a buried anchor with weight significantly larger than the Sensor Unit, we can assume this the anchor will be sufficiently unmoved by the Sensor Unit.

The installation and attachment of the tether to the Sensor Unit is another issue. Given the fact that using the steel auger would require placement of the anchor followed by soil and

then placement of the Sensor Unit followed by soil, the length of the tether must be available at the depth the Sensor Unit is buried. For a solution, the tether be wound up around a point on the Sensor Unit. This would allow for the Sensor Unit to simply unwind the tether as it rises. A visual representation of this idea is presented in Section 4.

6.4.2 Pros and Cons of Tethered Option

The tether option fixes the Sensor Unit to the vicinity of the bridge. Provided that the tether does not break during a flood event the Sensor Unit will not go out of range of the Receiver Unit. Hence, the tether enables operation in events with extremely high water velocity. In addition, the Sensor Units would be easy to recover because they remain within the vicinity of the bridge.

The tether option introduces an additional potential failure point into the system, namely the tether getting hung up during installation or on debris during a flood event. Also, the tether could break during the flood, or the tether's anchor could be washed way. Finally, the tether introduces difficulty in the installation process because an anchor point and tether must be installed. This limits the potential locations for the Sensor Units because of the necessity of the anchor point.

6.5 UN-TETHERED OPTION

In the un-tethered mode of operation, the Sensor Unit will float to the surface of the river/stream (water) and will transmit the message to the Receiver Unit. Here the critical factor is that the senor unit is within range of the Receiver Unit long enough to transmit at least one message.

6.5.1 Description of Un-Tethered Option Solution

The un-tethered option would simply place the Sensor Unit at a particular depth below the surface of the streambed/riverbed. As the stream/river bed is removed during a scouring event the Sensor Unit will be released, float to the surface, and be carried downstream. It is improbable that every Sensor Unit could be economically recovered in the un-tethered solution because the search area (downstream and the floodplain) is too large to search.

6.5.2 Pros and Cons of Un-Tethered Option

Lack of a tether provides for added freedom and flexibility in deployment of the Sensor Units. No anchor point on an existing structures or the placement of a dedicated anchor point for the Sensor Units is needed. Thus, deploying Sensor Units away from existing structures does not have the added cost of placing or connecting the Sensor Units to an anchor point. Further, removing the tether removes a potential failure point in the system. For example, if the tether gets tangled or hung up during installation, or gets hung up on some debris during a flood event the Sensor Unit may never reach the surface to communicate with the Receiver Unit.

Without the tether option the Sensor Unit may move out of range of the Receiver Unit (too far downstream) before the complete message is transmitted. Calculations show that this should not be the case, even for stream velocities of 33 ft/s and vertical distance that the Sensor Unit must travel 2910 ft from a depth of approximately 195 feet. However, for normal flood event stream velocities and depths this would be a problem. Further, once released, the Sensor Units will float downstream indefinitely or be deposited on a floodplain after the flood event. Hence, it is improbable that the Sensor Units could be economically recovered once they are released.

7.0 ASSEMBLY AND INSTALLATION

7.1 SYSTEM PCB ASSEMBLIES

There are three main components that must be assembled for the system. These components are the Sensor, Receiver, and Light Indicator Units. Each of these components require that a printed circuit board or PCB have components added to its surface. Components are added to the printed circuit board through soldering. For the surface mount chips, surface mount machines can be used. However, all chips are sufficiently large in size that hand soldering is also reliable. The .pcb file or printed circuit board file representing the printed circuit board of each unit provides a means to correspond the correct orientation and placement of each component. Within ExpressPCB, each component and its pads can be clicked on so that information regarding the name of the component and the pin clicked will come up. The .pcb file directly corresponds to the .sch or schematic file such that the schematic file can also be used for help. Failure to follow the placement specified by these files will result in malfunction of the unit.

7.2 SENSOR UNIT ENCAPSULATION AND INSTALLATION

The encapsulation and installation of the Sensor Unit has several particular steps that must be followed. This section will go through each step in detail.

7.2.1 Step 1 - Sensor Unit PCB Preparation

The Sensor Unit PCB must be assembled as described in section 3.0. Once all components have been added to the circuit board, the board should be programmed. The most up to date Sensor Unit source code output file should be downloaded to the microcontroller using the MSP-FET430UIFconnection and the FET –Pro430 utility described in section 2.2.1.2.

7.2.2 Sensor Unit PCB – Capsule Attachment

The printed circuit board will be attached to the inner face of the PVC cap. It is attached such that the length of the board will be perpendicular to the inner face of the PVC cap. To do this a slot .0634 inches wide by 1.5 inches long by .1 inches deep should be etched. This slot should be centered. Since the PVC inner cap face is slightly concave, the depth should be considered starting at the surface point furthest from the center of the cap. Once this slot has been machined, the bottom .1 inch of the PCB should be placed in the slot and junction adhesive should be applied.

7.2.3 Sealing the Sensor Unit Capsule

The Sensor Unit capsule should be as watertight as possible. Both the PVC capsule pipe and caps are threaded such that they can be screwed together. However this alone is not sufficient to keep the inside of the capsule completely dry. To further seal the capsule teflon tape should be wrapped around the threads on both ends of the PVC pipe prior to the caps being screwed on. Additionally, silicon sealant should be applied around of the junction of the cap and the pipe once they have been screwed together. These steps, along with the silica gel desiccants packets being placed inside the capsule, will keep the inside of the capsule completely dry. Remember to maintain the verticality of the PCB while sealing the capsule. The cap connected to the PCB should remain the lowest point of the Sensor Unit with the length of the PCB, antenna, and PVC pipe all such that they form a 90 degree angle with level ground. The Sensor Unit should then be placed in a container of water large enough to completely submerge the unit. Signs of any leakage should be observed such as bubbles being released from the unit. If this is the case, the Sensor Unit should be immediately removed from the water. The unit should then be unsealed and inspected. If undamaged, the unit should be re-tested and re-sealed.

7.2.4 Sensor Unit Installation/Deployment

The steps to properly deploy and properly assembled and programmed Sensor Unit are described here.

1. Drill hole in riverbed using hollow stem auger. Drill about 6 inches deeper than depth Sensor Unit should be buried.
2. Slowly add material to the hollow stem while slowly retracting auger about 6 inches. This material should fill in the bottom of the hole and provide a base for the Sensor Unit. Compacting this base might help to fill the hole.

3. Arm the Sensor Unit once in the field. It is important to keep the Sensor Unit in a vertical position to prevent it from being triggered.
4. Sensor Unit can be tested (by tilting it) to verify functionality and can be reset using the arm switch. A test Receiver Unit will be needed to determine if the Sensor Unit is functioning (transmitting). Any Receiver Unit on the bridge will need to be reset because it will detect the Sensor Unit being tested.
5. Place Sensor Unit in hollow stem auger and push down to bottom using push rod. The push rod may need to be attached to the Sensor Unit to keep it in a vertical orientation during filling stage.
6. Slowly fill the auger with material while slowly retracting the auger until entire hole is filled.

8.0 TESTING

8.1 PRELIMINARY COMPONENT TESTING

This section contains a review of preliminary tests done on the early RF receiver, Transmitter, and Light Indicator. This work had the goal of ensuring that these components are suitable to complete the tasks necessary for this project. First, tests were performed with the transmitter with respect to its signal strength under various conditions. Second, the ability of the RF transmitter and receiver to send and receive data respectively was analyzed. Lastly, the Light Indicator was tested to ensure proper input would trigger the correct LED. Additionally, the tests done to show the functionality of the various components on the boards is shown.

8.1.2 Transmitter Signal Strength

A Real Time Spectrum Analyzer (RTSA) was used to measure the signal strength of the transmitter under a variety of different conditions. The conditions emulate the normal operating conditions of the transmitter. Overall, the signal strength shown on the instrument due to the transmitter ranged from –23 dBm to –55 dBm. Here the larger the number shown equates to a stronger signal (-23 dBm is stronger than -55 dBm). The Receiver RF chip has sensitivity up to -112 dBm. This means it can recognize the carrier-present signal of the Transmitter if the signal has a strength of at least -112 dBm.

This experiment evaluated four conditions for signal strength. First, the Transmitter was tested in free-air outside of the capsule, denoted *TX Test 1*. Second, the transmitter was tested inside of the capsule, denoted *TX Test 2*. Third, the Transmitter was tested while inside the capsule while floating in water in a horizontal orientation, denoted *TX Test 3*. Finally, the floating test (third test) was repeated with the capsule floating in a vertical orientation, denoted *TX Test 4*.

Table 9 contains the results of the transmitting power in free-air and from within the Capsule, TX Test 1 and TX Test 2. These tests were done at several orientations for both the RTSA probe and the transmitter. The number highlighted in red is the stronger signal. The results show that the transmitter signal is well above the necessary -112 dBm required for the receiver for all orientations in both cases. The results also show that there is no substantial degradation of the signal when placed within the capsule. In fact, in several instances the signal was stronger from within the capsule. This is counter-intuitive from what was expected but is an added benefit.

Table 9: Signal Strength Tests in Free Air and within Capsule (TX Test 1 and TX Test 2)

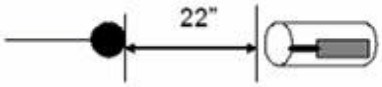
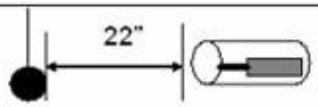
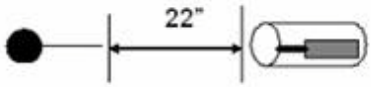
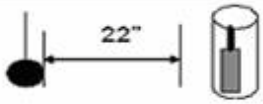
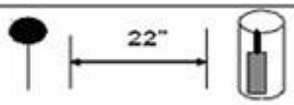
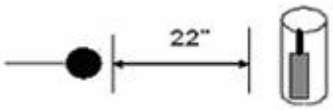
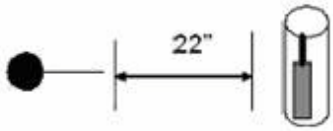
Probe-Capsule Orientation (Overhead View)	Power Seen on RTSA (Free Air)	Power Seen on RTSA (Within Capsule)
	-34 dBm	-31 dBm
	-40 dBm	-55 dBm
	-32 dBm	-29 dBm
	-25 dBm	-30 dBm
	-25 dBm	-23 dBm
	-28 dBm	-43 dBm
	-33 dBm	-41 dBm

Table 10 contains the results of the transmitting power while the capsule is floating in a horizontal orientation, TX Test 3. The RTSA probe was placed in an orientation parallel to the capsule and at an orientation perpendicular to the capsule. For each of these probe orientations, readings were taken with the capsule antenna pointing in four different directions. The resulting signals seen in this test were also well above the necessary strength of -122 dBm. The signal was slightly weaker than the signal tests done out of the water for most comparisons, but still well within the requirement of -122 dBm.

Table 10: Signal Strength Tests For Horizontal Floating Capsule (TX Test 3)

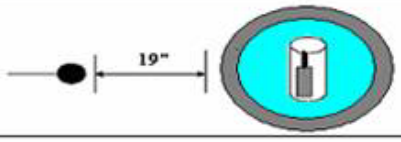
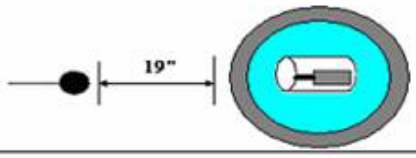
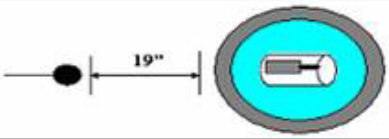
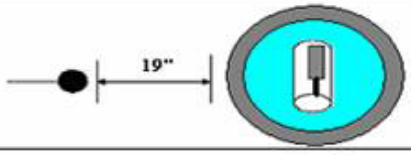
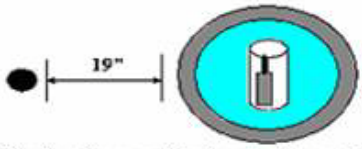
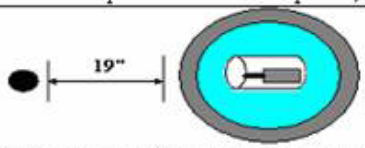
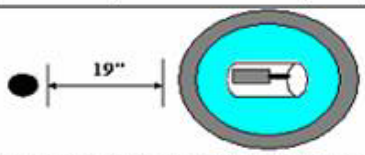
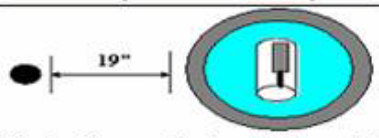
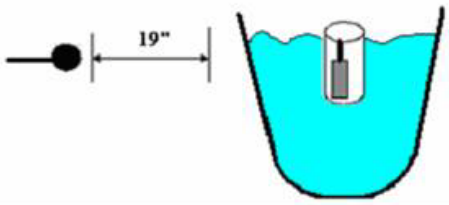
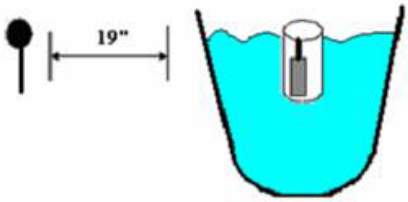
Probe-Capsule Orientation (Overhead View)	Power Seen On RTSA
	-50 dBm
	-51 dBm
	-50 dBm
	-51 dBm
 (Probe Perpendicular to Capsule)	-47 dBm
 (Probe Perpendicular to Capsule)	-42 dBm
 (Probe Perpendicular to Capsule)	-43 dBm
 (Probe Perpendicular to Capsule)	-50 dBm

Table 11 contains the results of the transmitting power while the capsule is floating in a vertical orientation, TX Test 4. Weight was added to the capsule in order to accomplish this orientation. The probe was placed in an orientation both parallel and perpendicular to the capsule. The readings for these tests were also sufficiently strong for our receiver. In comparison to the out of water tests, they were weaker than most readings in Table 9. In comparison to the horizontal floating test, the results were similar with the readings in Table 8 and there is no clear benefit to the vertical orientation over the horizontal orientation. The horizontal orientation is recommended because of the its simpler design and construction requirements.

Table 11: Signal Strength Tests for Vertically Floating Capsule (TX Test 4)

Probe-Capsule Orientation (Side View)	Power Seen On RTSA
	-44 dBm
	-47 dBm

8.1.3 Transmitter-Receiver RF Link

A transmitter-receiver RF link was evaluated in order to finalize the choice of the receiver. To verify the transmitter-receiver RF link, a known data pattern was provided to the transmitter. Simultaneously, the output of the receiver was monitored for this pattern. The following simple tests shown in Table 12 were done to achieve this. In each case, the software was composed to control the logic level on pin 21 of the microcontroller. This pin serves as the input to the RF transmitter chip. An oscilloscope was then used to view the logic level on the output line of the RF receiver chip. Results were as expected. In the case were a logical low (0 volts DC) was placed on the input of the RF transmitter chip, some noise was present on the output of the RF receiver chip. This noise is noted within the data guide for the RF receiver chip and will be handled by the Receiver Unit in software and hardware.

Table 12:Transmitter-Receiver RF Link Tests

	Condition of Input to RF Transmitter Chip	**Expected Logic Level seen on output of RF Receiver Chip**	**Logic Level seen on output of RF Receiver Chip**	**Test Result**
Test 1	Logical High	Logical High	Logical High	Success
Test 2	Logical Low	Logical Low	Low, Some Noise	Success
Test 3	Oscillating High and Low Levels	Oscillating High and Low Levels	Oscillating High and Low Levels	Success

**High Level here means ~3.0 volts DC and Low Level means ~0.0 volts DC

8.1.4 Sensor Unit Preliminary Component Testing

The top side of the constructed Transmitter prototype is shown in Figure 62 and the bottom side is shown in Figure 63. There are that there are two extra LEDs on the transmitter. These LEDs were simply added for testing the tilt switch, which will be explained shortly and will not be part of the final Sensor Unit. Below are a list of basic hardware tests done to date and their outcomes.

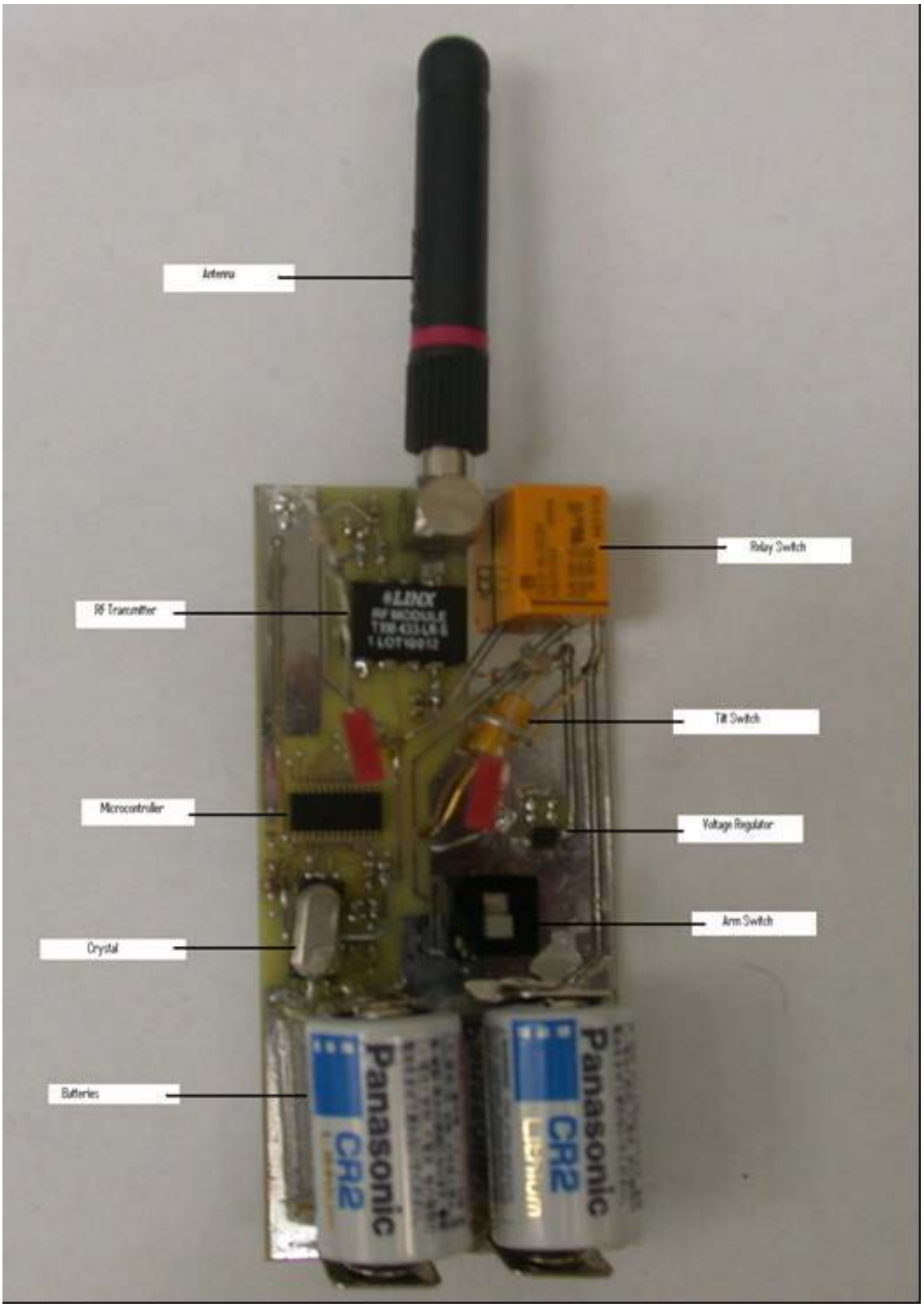


Figure 62: Transmitter prototype - top side

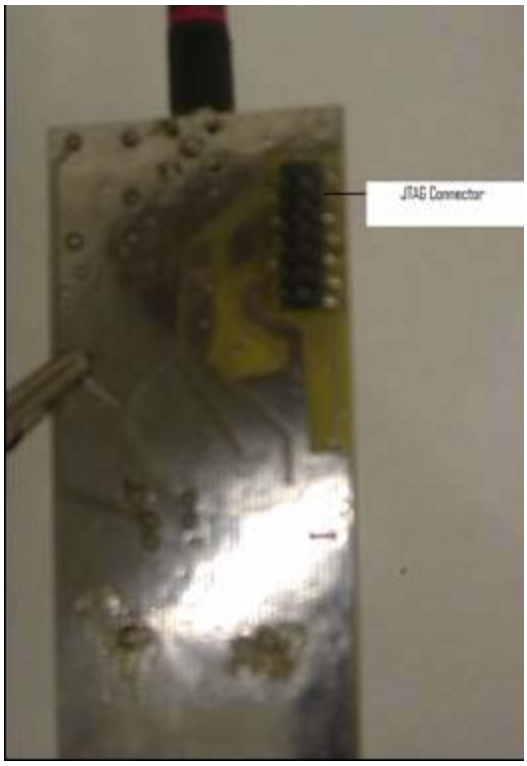


Figure 63: Transmitter prototype - bottom side

8.1.4.1 Battery Test

The batteries will be used to power the Sensor Unit. The trace connected to the positive potential of the batteries was probed using a multimeter to determine the voltage output.

Table 13: Battery Test

	Condition	Expected Outcome	Observed Outcome	Test Results
Test 1	Batteries Added	-3 volts DC on output trace	-3 volts DC on output trace	Success

8.1.4.2 Tilt Switch

The tilt switch will be used to connect the batteries to the arm switch when in the correct orientation (after release from soil). The common electrode is connected to the arm switch and serves as the output of the tilt switch. The common electrode of the tilt switch was tested using a multimeter and a LED.

Table 14: Tilt Switch Test

	Tilt Switch Condition	**Expected Tilt Switch Outcome**	**Observed Tilt Switch Outcome**	**Test Results**
Test 1	Vertical Orientation (See Figure 3)	-0 volts DC on common electrode -LED not lit	-0 volts DC on common electrode -LED not lit	Success
Test 2	Horizontal Orientation (See Figure 4)	-3 volts DC on common electrode -LED lit	-3 volts DC on common electrode -LED lit	Success

Figure 64: Tilt Switch test vertical orientation

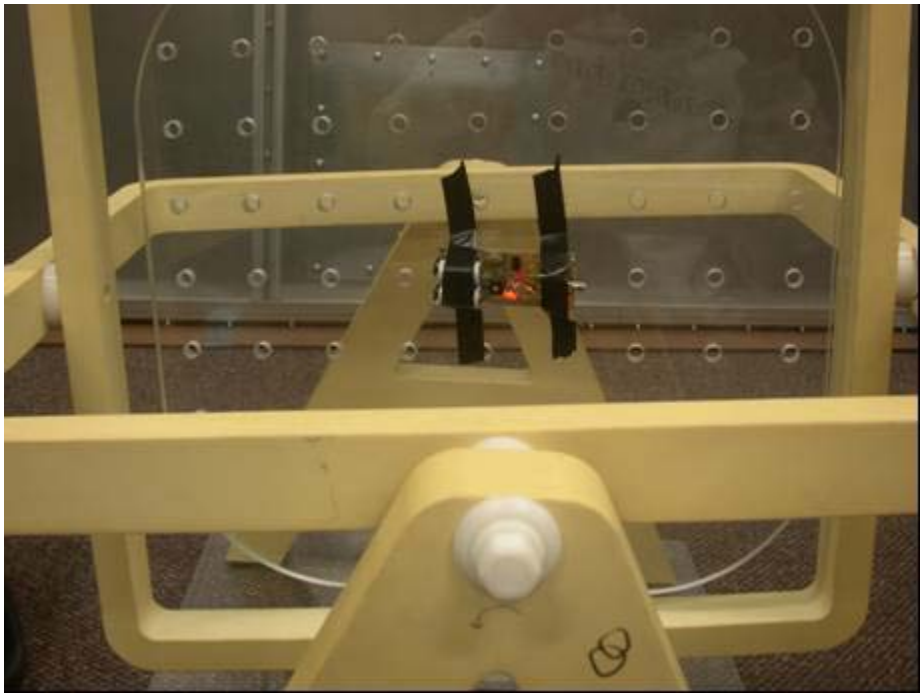


Figure 65:Tilt Switch test horizontal orientation

8.1.4.3 Arm Switch

The output of the arm switch is used to operate the relay switch. The state of pins 2 and 5 of the arm switch are used to control the relay switch. These two pins were tested for their voltage output in each of the arms switch's three positions using a multimeter. Table 15 displays the results.

Table 15:Arm Switch Test

	Arm Switch Condition	Expected Arm Switch Outcome	Observed Arm Switch Outcome	Test Results
Test 1	"Armed State" Connects pin 2 to 3 Connects pin 5 to 6	3 volts DC on pin 5 0 volts DC on pin 2	3 volts DC on pin 5 0 volts DC on pin 2	Success
Test 2	"Reset State" Connects pin 2 to 1 Connects pin 5 to 4	0 volts DC on pin 5 3 volts DC on pin 2	0 volts DC on pin 5 3 volts DC on pin	Success
Test 3	"Off State" No pins interconnected	0 volts DC on pin 5 0 volts DC on pin 2	0 volts DC on pin 5 0 volts DC on pin 2	Success

8.1.4.4 Relay Switch

The relay switch has two states. In one state, the common pin (Pin 2) is connected to the batteries (Pin 1). In the second state, the common pin is connected to nothing. The output of the common pin was tested using a multimeter and a LED. Figure 66 shows the relay switch in a horizontal orientation and Figure 67 shows the relay switch in a vertical orientation.

Table 16: Relay Switch Test

	Relay Switch Condition	Expected Relay Switch Outcome	Observed Relay Switch Outcome	Test Results
Test 1	3 volts DC on pin 1 3 volts DC on pin 4 0 volts DC on pin 5 (See Figure 5)	3 volts DC on pin 3 LED Lit	3 volts DC on pin 3 LED Lit	Partial Success
Test 2	Follows Test 1 3 volts DC on pin 1 0 volts DC on pin 4 0 volts DC on pin 5 (See Figure 6)	3 volts DC on pin 3 LED Lit	3 volts DC on pin 3 LED Lit	Success
Test 3	3 volts DC on pin 1 0 volts DC on pin 4 3 volts DC on pin 5	0 volts DC on pin 3 LED not Lit	0 volts DC on pin 3 LED not Lit	Success
Test 4	Follows Test 3 3 volts DC on pin 1 0 volts DC on pin 4 0 volts DC on pin 5	0 volts DC on pin 3 LED not Lit	0 volts DC on pin 3 LED not Lit	Success

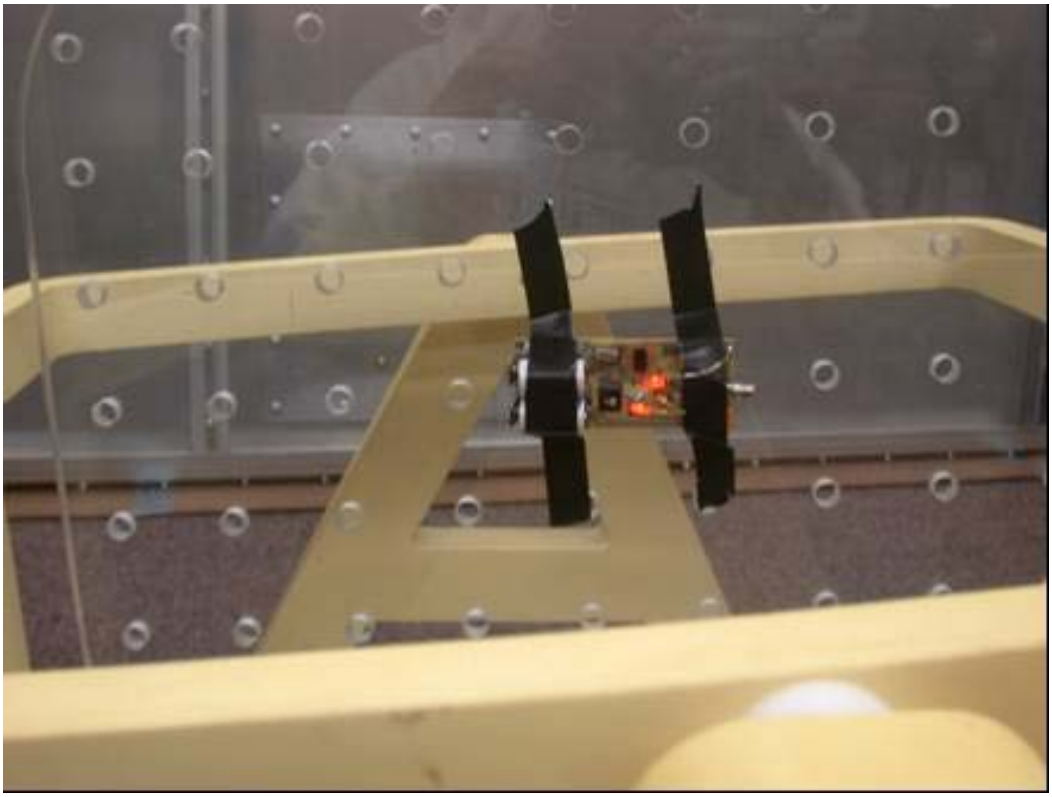


Figure 66: Relay Switch test in horizontal orientation

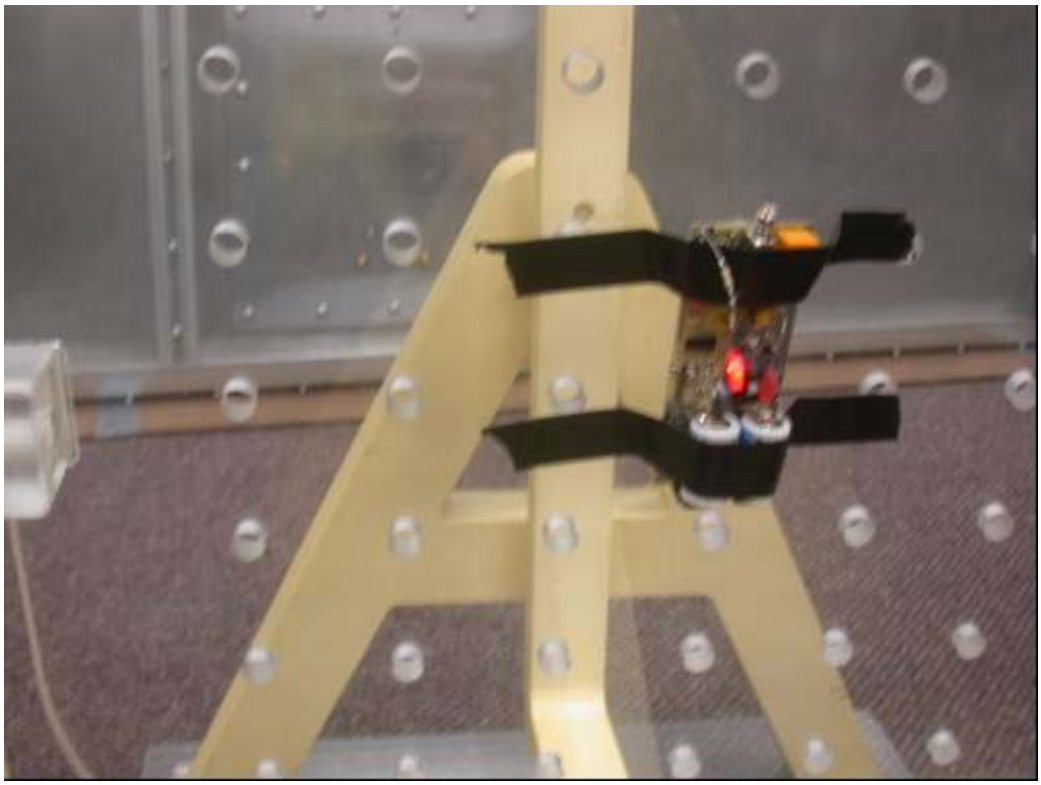


Figure 67: Relay Switch test in vertical position

8.1.5 Receiver Unit Preliminary Component Testing

The preliminary prototype of the Receiver Unit is shown below in Figure 68. The unit has shown the ability to be programmed and receive RF data at this point. The only component test unique to the Receiver Unit is the voltage regulator shown below.

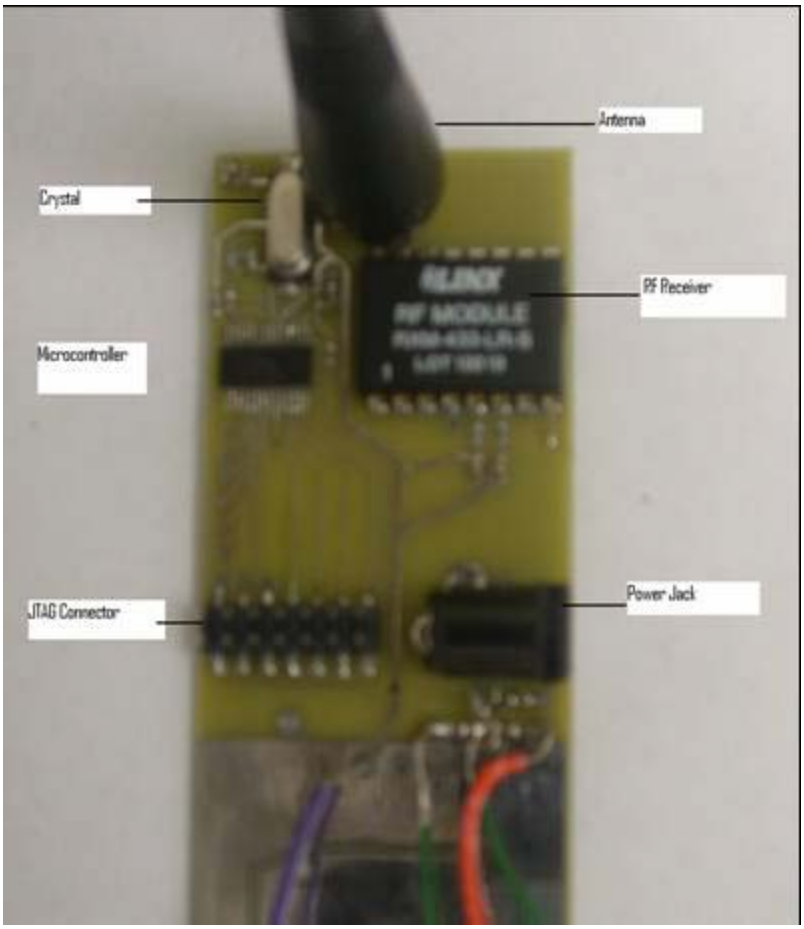


Figure 68: Receiver Unit prototype

8.1.5.1 Voltage Regulator

The voltage regulator connected to the power jack should be able to take 3.0 to 16.0 Volts of DC input range and output 3.0 Volts of DC output.

Table 17: Voltage Regulator Test

	Voltage Regulator Input Condition	Expected Output	Observed Result	Test Outcome
Test 1	3.0 Volts DC input	3.0 Volts DC output	3.0 Volts DC output	Success
Test 2	16.0 Volts DC input	3.0 Volts DC output	3.0 Volts DC output	Success

8.1.6 Light Indicator Preliminary Tests

The preliminary prototype of the Light Indicator Unit is shown in Figure 69. The Light Indicator Unit was connected to four DIP-switch modules. Each DIP-switch module contains eight switches, of which four are used to represent a single Receiver Unit. The four DIP-switch modules emulate four independent Receiver Units. Each switch represents one of the I/O pins of a single Receiver Unit that would control a corresponding LED on the Light Indicator Unit. One of the switch inputs was connected to a 3 volt DC power source and the output pin was connected to the Light Indicator Unit. Each switch was toggled and the corresponding LED was observed. All inputs lit the correct LED. Figure 69 shows the test setup for the Light Indicator Unit.

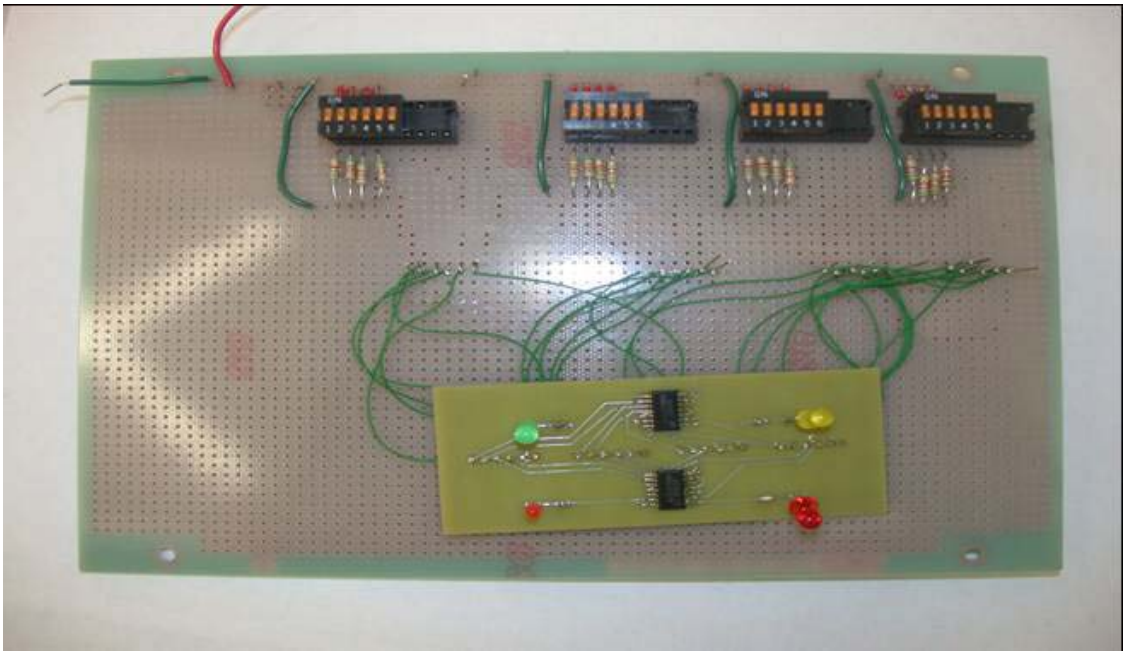


Figure 69: Light Indicator Unit prototype and test stand

8.2 CAPSULE FUNCTIONALITY TESTING

The Sensor Unit capsule has three functions it must perform. First, it must be watertight in order to prevent water from damaging the electronics of the Sensor Unit. Second, it must be able to rise to the surface quickly. Lastly, it must be able to float on the surface of water. The sections below describe tests done to confirm these properties.

8.2.1 Buoyancy

The capsule naturally floats as shown in Figure 70. The capsule will continue to have some portion of it's shell un-submerged until over 6 oz of weight has been added. This confirms its ability to float.

Figure 70: Capsule floating with no additional weight added

8.2.2 Rise times

Rise time tests were done to simulate the release of the capsule and measure how quickly the capsule will rise to the surface. A large bucket was used to do this. A housing was also built to place the capsule in while under water. This housing was composed of 3-inch diameter PVC

pipe attached to a wooden base. The 3-inch diameter PVC pipe is similar to the 3-3/16 inch diameter hollow stem auger that will eventually be used. To perform the tests the bucket was filled with water up to a height of 29 inches. Then, the capsule was pushed down this pipe and held down using a pole. The time it took the capsule to rise once the pole was lifted was then recorded. It took the capsule 1.33 seconds with nothing inside. With the Transmitter Unit added, it took 1.635 seconds. These times are fairly similar to rise times using simulation. Figure 71 and Figure 72 illustrate this test.

Figure 71: Rise time test before capsule release.

Figure 72: Rise Time Test after Sensor Release

8.2.3 Water Tightness

Water tightness tests were performed on the capsule. For each of these tests, weights were placed within the capsule as shown in Figure 73 that would sink the capsule. Once the capsule was sealed, it would be left underwater for three days. Following the three days the capsule was be opened and inspected for water.

Figure 73: Weight added to keep Capsule submerged for Water Tightness tests.

The first test done only used the threaded caps to the seal the capsule. The result of this test was unsuccessful. A substantial amount of water collected within the capsule for this test. The water collected is shown in Figure 74.

Figure 74: Water collected from the first Water Tight test using only the PVC pipe.

The second test was done with Teflon tape placed on the threads of the capsule as shown in Figure 75. The result of the test was greatly improved in comparison with the unthreaded test. However, a small amount of water could be seen on the wall of the capsule.

Figure 75: Capsule with Teflon tape added.

It was estimated at this point that the moisture was due to condensation. Therefore, Silica gel desiccants were used in conjunction with the Teflon tape. A packet of this is shown in Figure 76. The Silica gel desiccants absorb small amounts of moisture that may be present within the air. They are commonly used in several storage capacities. Two of the packets were used in the test. The result of this test was a success because no moisture was observed after three days underwater. Additionally, the Silica gel desiccants change color as they near the limit of moisture they can absorb. In this test, they did not change color and therefore are capable of absorbing more moisture.

Figure 76: One Silica gel desiccant packet

8.3 SYSTEM FUNCTIONALITY TEST

8.3.1 Overall System Demonstration

This section describes the demonstration that was run to verify system functionality. The goals realized from the demonstration were as follows:

1. Program Transmitter with a Data Block
2. Arm and Encapsulate Sensor Unit
3. Bury Sensor Unit under soil/aggregate using hollow 3 inch diameter pipe in a water environment
4. Uncover Sensor Unit resulting its rise to the surface and activation
5. Receive and decode Data Block on Receiver Unit
6. View visual confirmation of transmission and decode on Light Indicator

The demonstration took place within a lab environment within Benedum Hall. The Data Block used for the demonstration was a dummy block randomly created. The only piece of the Data Block critical to the demonstration was that of the Color Code. For this demonstration, the Color Code was given a value of 0x01. This corresponds to the activation of the yellow LED on the Light Indicator. A picture of what the dummy block looks like in memory is shown in Figure 77. The Data Block starts at address 0x1080 and ends at 0x1097.

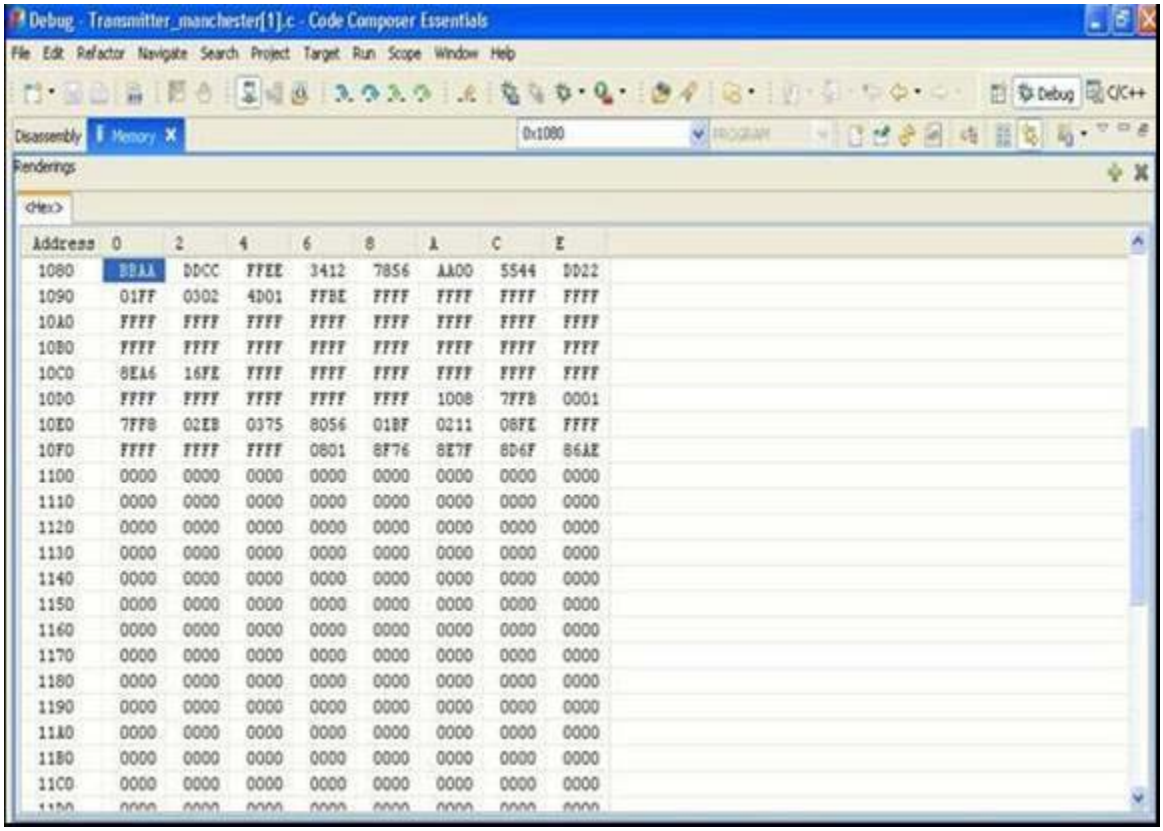


Figure 77: Transmitter Memory View of Dummy Data Block

The Sensor Unit is shown in Figure 78. This Sensor Unit has three tilt switches located on it. Two of the tilt switches can be seen on the front of the PCB while another one is located on the back. This is detailed more in Section 3 of this document. Once the Sensor Unit is armed using the toggle switch and the Teflon tape is applied to the threads, the caps can be screwed on. The Sensor is now encapsulated and ready to be deployed. Black tape was also referenced on the caps for indication of the orientation of the PCB within the Capsule while floating.

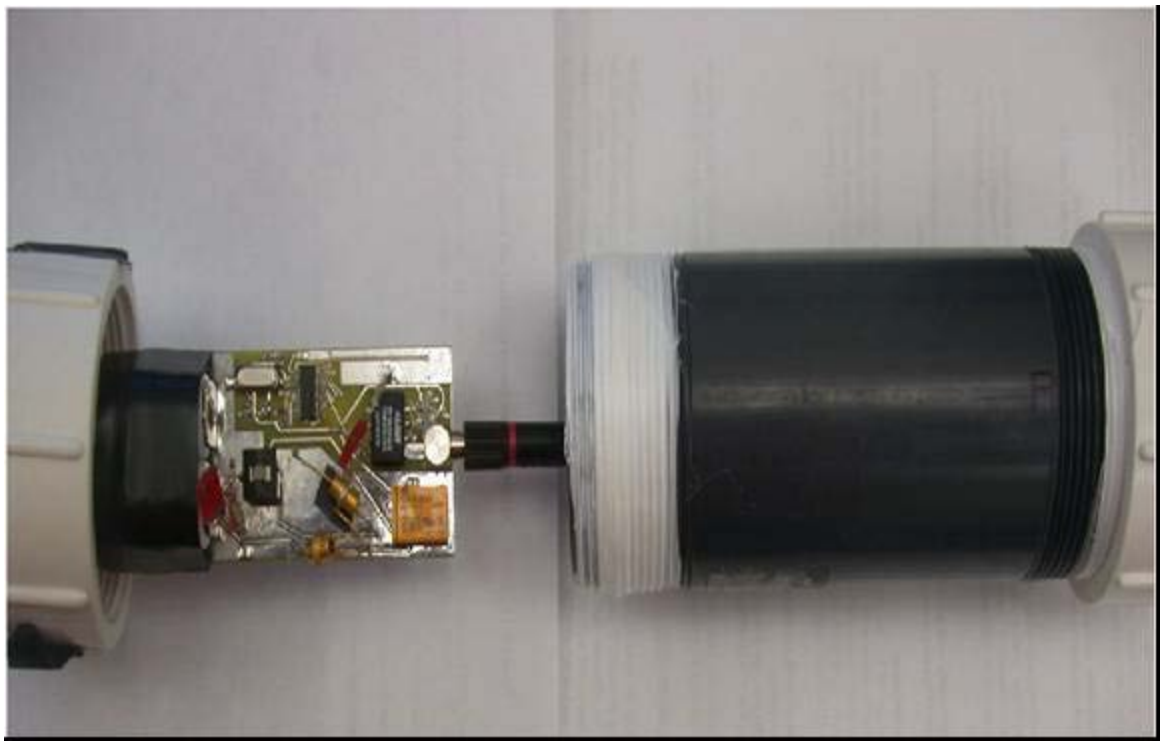


Figure 78: Sensor Unit

The physical setup that the Sensor Unit was placed in was a large yellow container, rock aggregate, and a 3-inch diameter by 5-foot length PVC pipe. The pipe was used to simulate the hollow stem auger that will be used in practice. This pipe was extended to the bottom of the container. It was surrounded by the rock aggregate as shown in Figure 79. The container was then filled with water as shown in Figure 80.

Figure 79: Container with aggregate and pipe

Figure 80: Container with aggregate, pipe, and water

The Receiver Unit and Light Indicator were also placed within the same room as shown in Figure 81. This version of the Receiver Unit can be connected through a power jack to a wall outlet. When initially plugged in it initiates the Light Indicator to light the green LED. In practice the green LED may not be lit to begin given that four Sensor Units are assigned to the Receiver Unit.

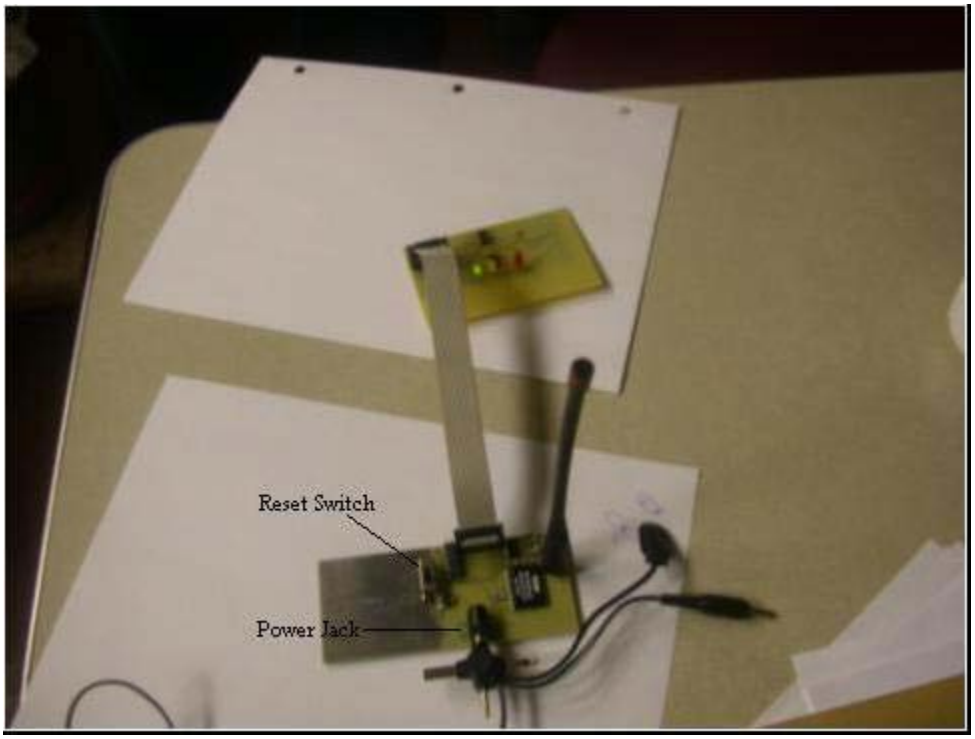


Figure 81: Receiver Unit and Light Indicator with no message received from a Sensor Unit.

The armed Senor Unit was then dropped down the pipe and pushed down to the bottom of the container using a long pole. Aggregate was then dropped down the pipe on top of the Sensor Unit to bury it. The pipe was then pulled up out of the aggregate leaving the Sensor Unit buried. The aggregate was then slowly brushed away until the Sensor Unit released itself and rose to the surface of the water as shown in Figure 82. This resulted in the Light Indicator quickly lighting

the yellow LED as shown in Figure 83. This proves the success of system functionality under anticipated conditions.

Figure 82: Sensor Unit after release from aggregate

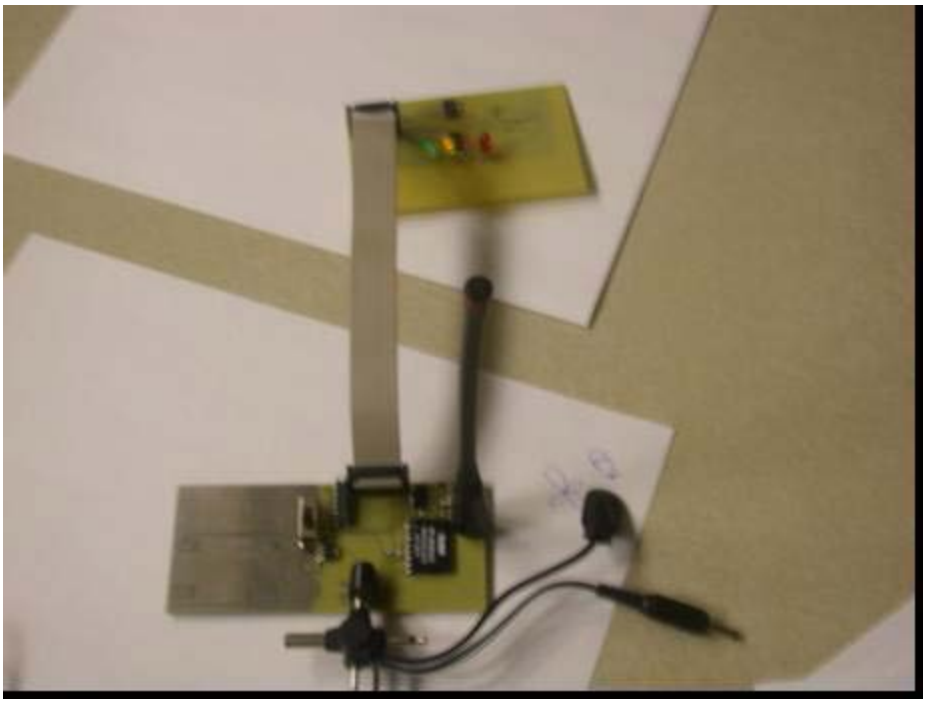


Figure 83: Receiver and Light Indicator after Sensor Unit release

8.3.2 Multiple Sensor Functionality

This section will provide images of the current system operating using four Sensor Units. Each of the Color Codes (green, yellow, orange, red) will be represented by one of the four Sensor Units. This demonstration does not take place in a water environment as the Sensor Unit has already been shown to have proper functionality within a water environment. For this demonstration the main concerns deal with ensuring that the Receiver Unit will trigger the correct output given multiple Sensor Unit as well as the system's ability to function properly with multiple Sensor Units transmitting at the same time. It should be noted that Sensor Units are placed very close to the Receiver Unit so that all component can easily be seen. This is not due to the systems inability to work at further distances. In fact, at such a close range the Receiver Unit does not pick the signals of the Sensor Units as cleanly.

Demonstration Setup – Figure 84 shows the setup of the demonstration. At the bottom of the figure are the four Sensor Units used. Written above each Sensor Unit is the Color Code that was programmed to it. At the top of the figure is the Receiver Unit connected to the current Light Indicator. When any of the Sensor Units are active, the red LED on the Sensor Unit will be lit.

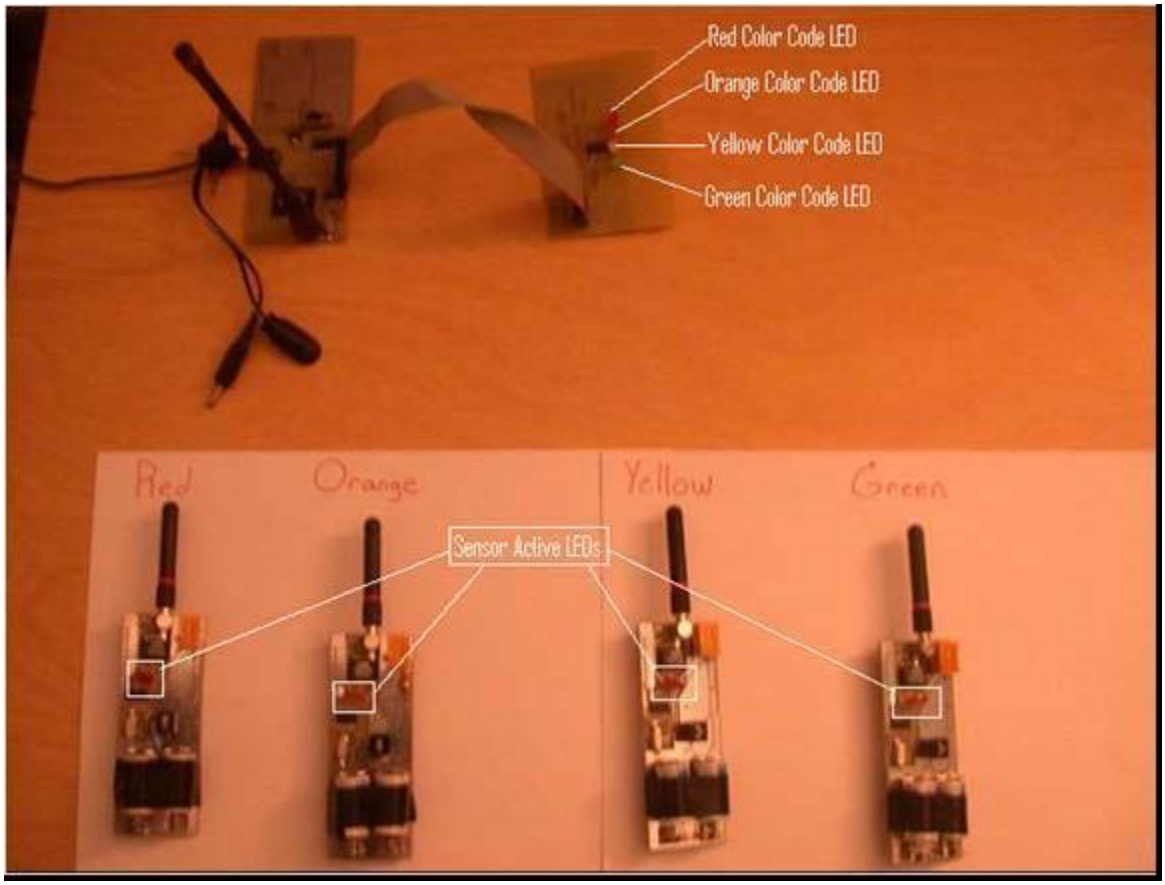


Figure 84: Demonstration Setup - No Sensor Units Active

Step 1: Green Sensor Unit Activation – Figure 85, below, shows the first action taken in the system demonstration. This action was activating the Sensor Unit programmed with the

green Color Code. In the figure, the green Sensor Unit's activation is shown by its red LED being lit. The proper response by the Receiver Unit is also indicated by the green LED being lit on the Light Indicator. This step was successful.

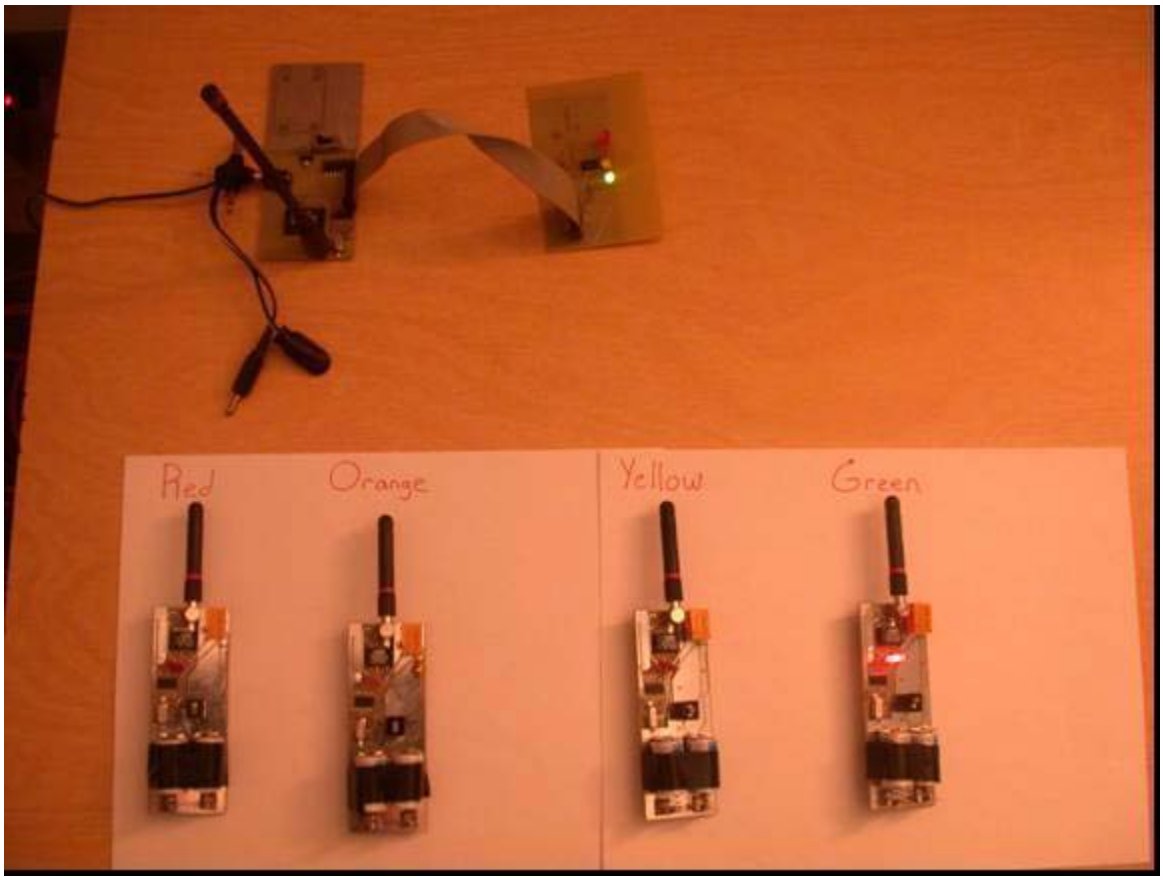


Figure 85: Step 1 - Green Sensor Unit Activation

Step 2: Yellow Sensor Unit Activation – Figure 86, below, shows the second action taken in the system demonstration. This action was activating the Sensor Unit programmed with the yellow Color Code. In the figure, the yellow Sensor Unit's activation is shown by its red LED being lit. Notice also that the green Sensor Unit is lit and active which would expose the system to collisions. The proper response by the Receiver Unit is also indicated by the yellow

LED being lit on the Light Indicator. This proves that the anti-collision protocol was successful, and a proper response received by the Receiver Unit state machine. This step was successful.

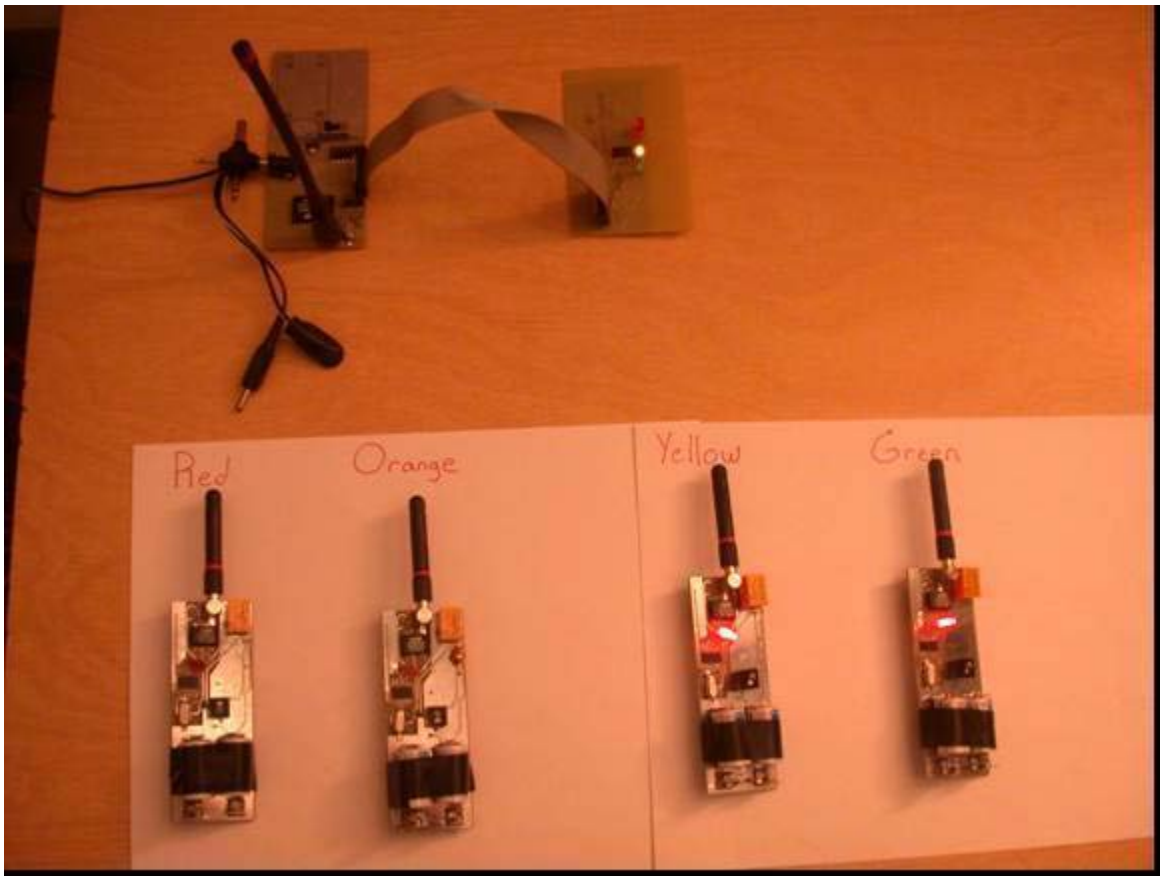


Figure 86: Step 2 - Yellow Sensor Unit Activation

Step 3: Orange Sensor Unit Activation – Figure 87 below shows the third action taken in the system demonstration. This action was activating the Sensor Unit programmed with the orange Color Code. In the figure, the orange Sensor Unit's activation is shown by its red LED being lit. Notice also that green and yellow Sensor Units are lit and active which would expose the system to collisions. The proper response by the Receiver Unit is indicated by the orange LED being lit on the Light Indicator. This proves that the anti-collision protocol can handle

three Sensor Units transmitting at the same time and a proper response by the Receiver Unit state machine. This step was successful.

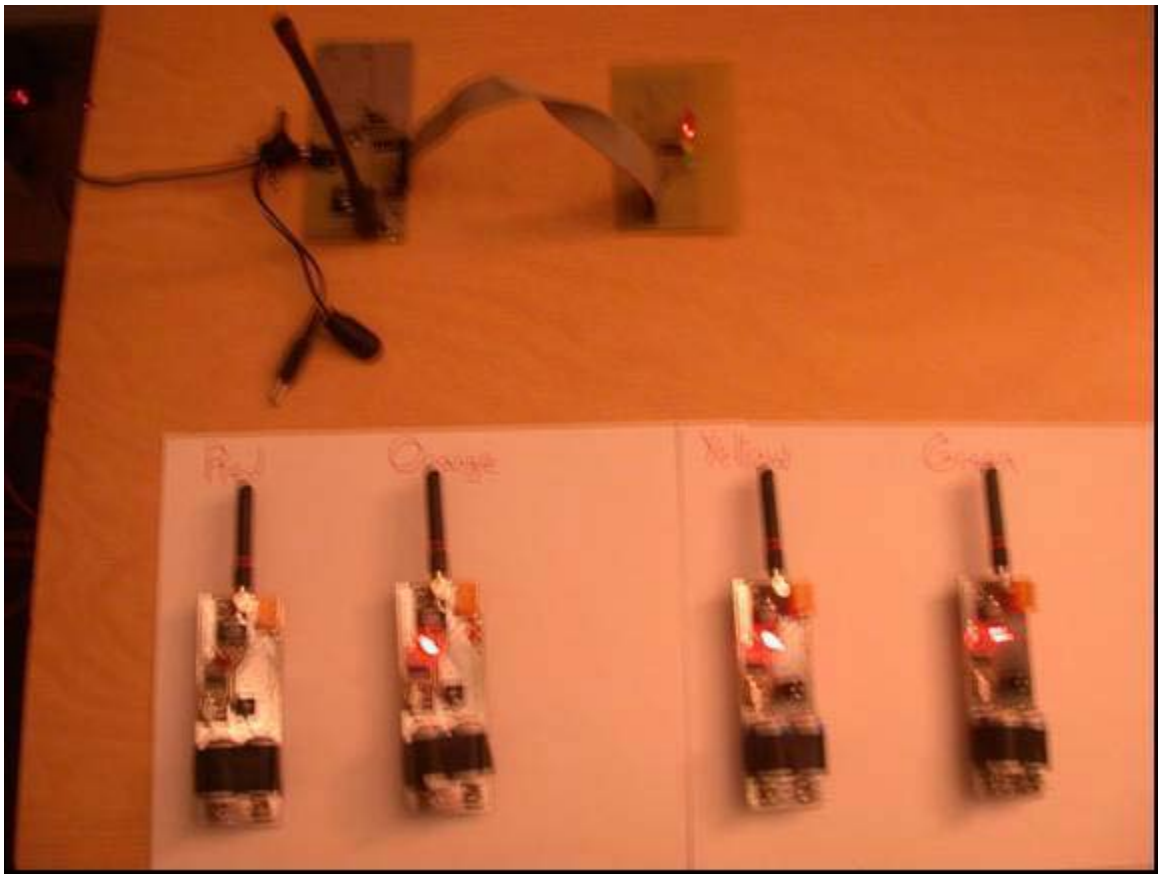


Figure 87: Step 3 - Orange Sensor Unit Activation

Step 4: Orange Sensor Unit Activation – Figure 88 below shows the fourth action taken in the system demonstration. This action was activating the Sensor Unit programmed with the red Color Code. In the figure, the red Sensor Unit's activation is shown by its red LED being lit. Notice also that green, yellow, and orange Sensor Units are lit and active which would expose the system to collisions. The proper response by the Receiver Unit is also indicated by the red LED being lit on the Light Indicator. This proves the anti-collision protocol allows four Sensor

Units be active at the same time while still allowing the messages to reach the Receiver Unit. The Receiver Unit state machine functioned according to specifications by lighting and maintaining the red LED when lower Color Code messages (orange, yellow, and green) were received. This step was successful.

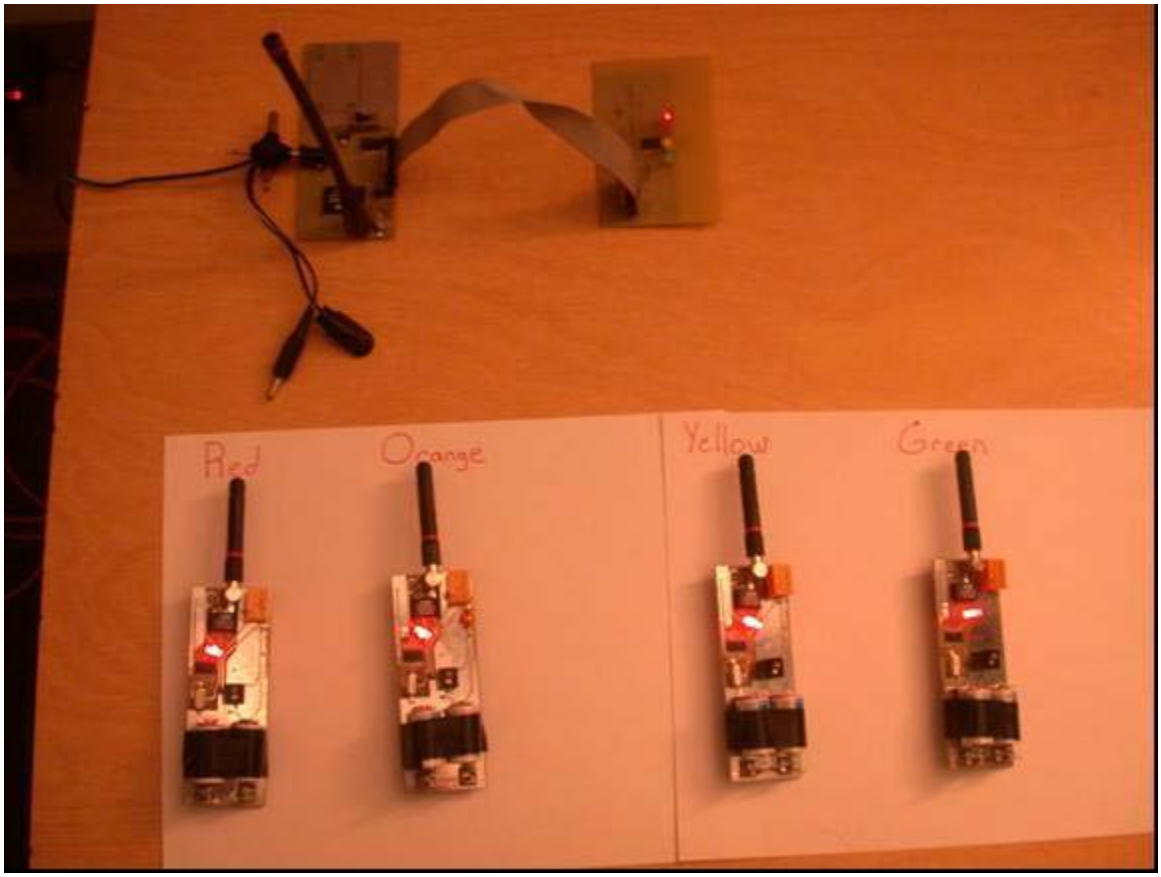


Figure 88: Step 4 - Red Sensor Unit Activation

9.0 CONCLUSIONS

A proof of concept for the remote sensing system has been developed and tested. The system is able to wirelessly transmit a message from a Sensor Unit to a Receiver Unit with the correct response displayed through a LED on the Light Indicator. The Sensor Unit is able to be buried under material within a water environment where it can remain dormant for several years. Upon release from this material, the Sensor Unit has shown the ability to float to the water surface and activate its internal circuitry using orientation sensitive switches. Operation with the use of multiple Sensor Units has also proven successful. Collision avoidance for simultaneous transmission by multiple Sensor Units is working correctly. The system displays the correct LED being lit on the Light Indicator given the highest priority Sensor Unit being active in this case. The functionality demonstrated by this system should allow for the monitoring of bridge scour for PennDOT in the future.

10.0 FUTURE WORK

Future work for the presented solution consists of the following:

- Test in real world environment. Testing at an actual bridge site would be optimal for this task.
- Design the Receiver Unit to have the capability to receiver multiple set of sensor units for individual parts of the bridge. It would also be required to drive multiple Light Indicators.

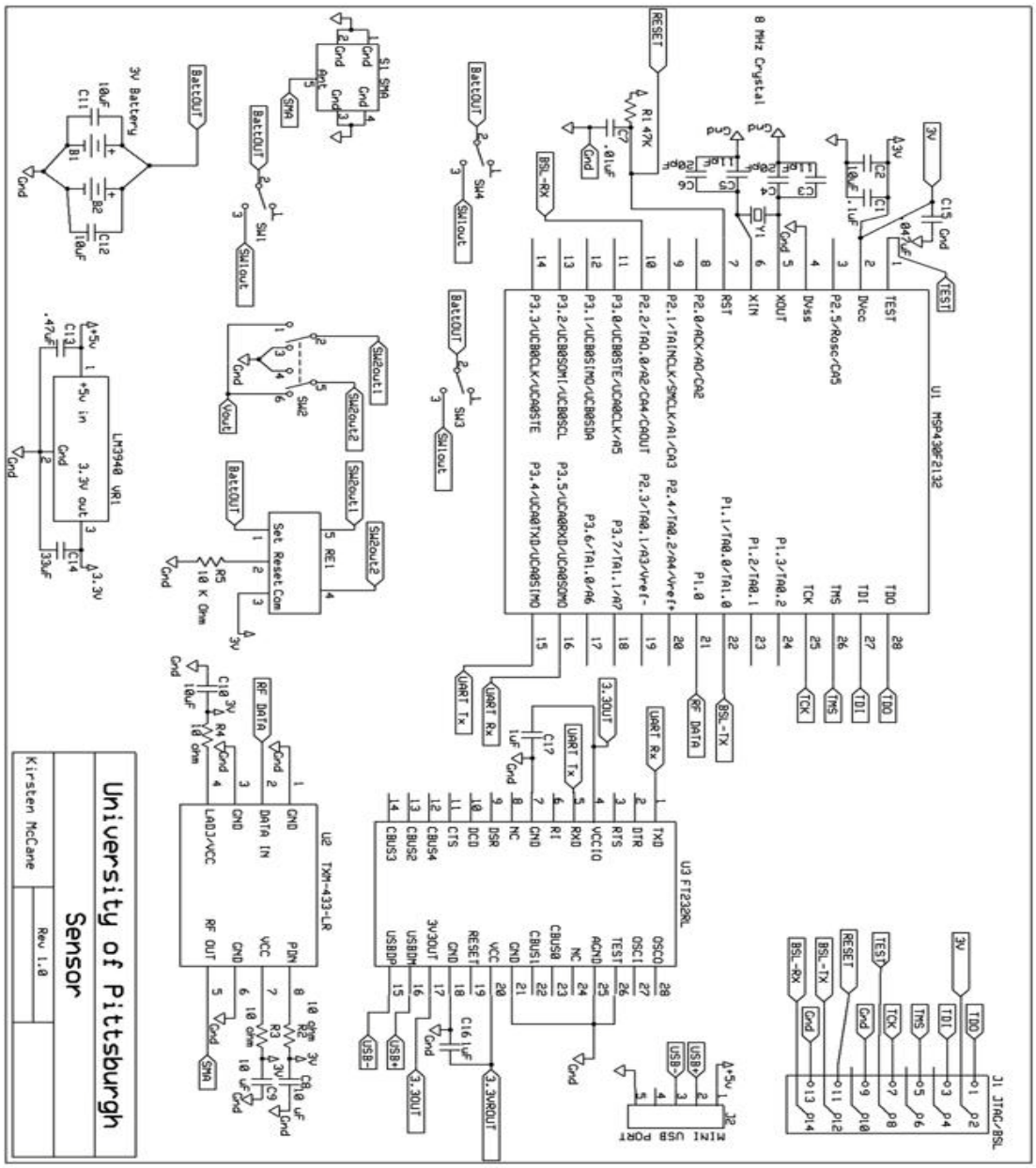
APPENDIX A

HARDWARE DESIGN FILES

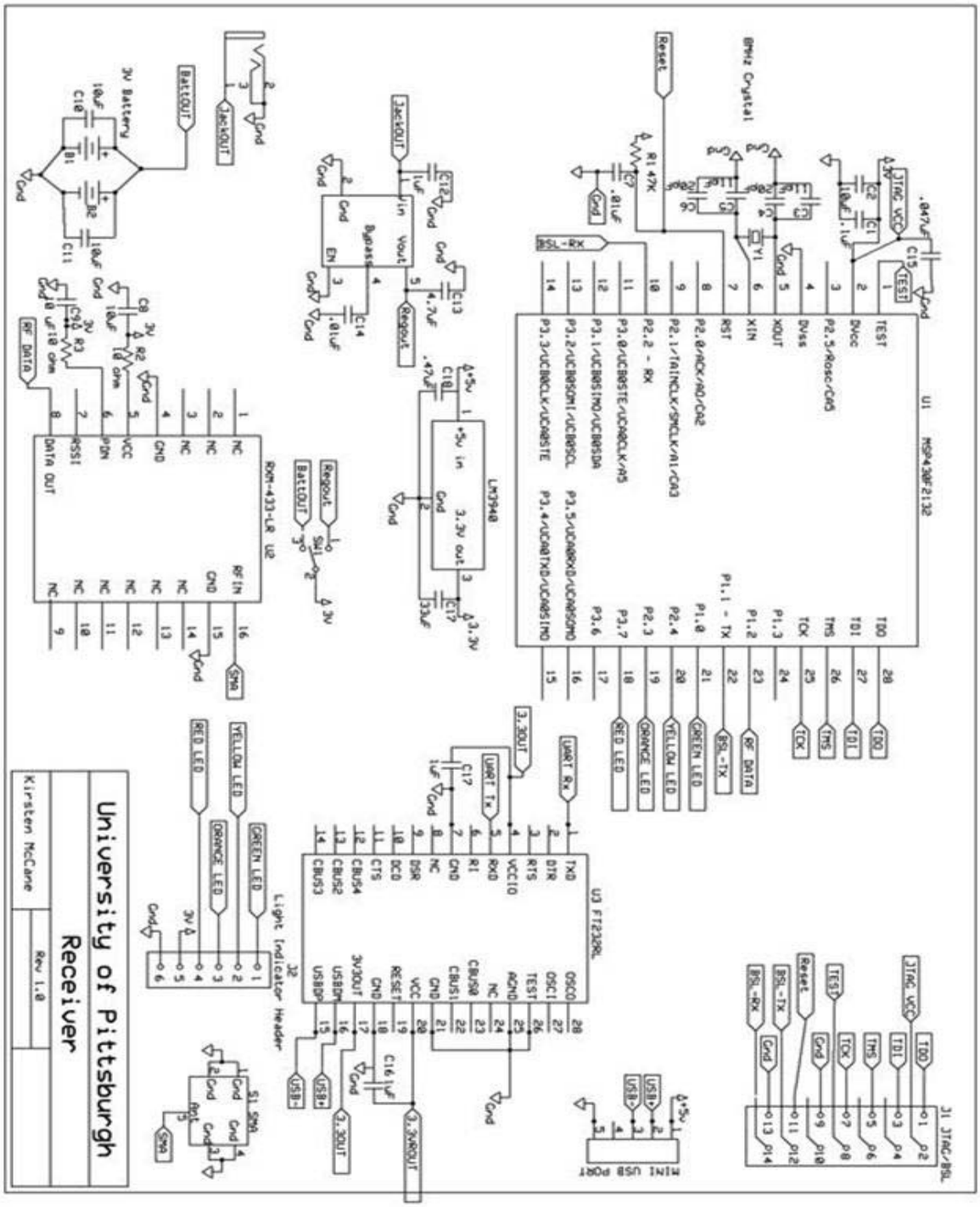
This appendix contains the schematics created for Sensor Unit, Receiver Unit, and Light Indicator within ExpressSCH software. It also contains the PCB design files created within ExpressPCB software for the three components.

SCHEMATICS

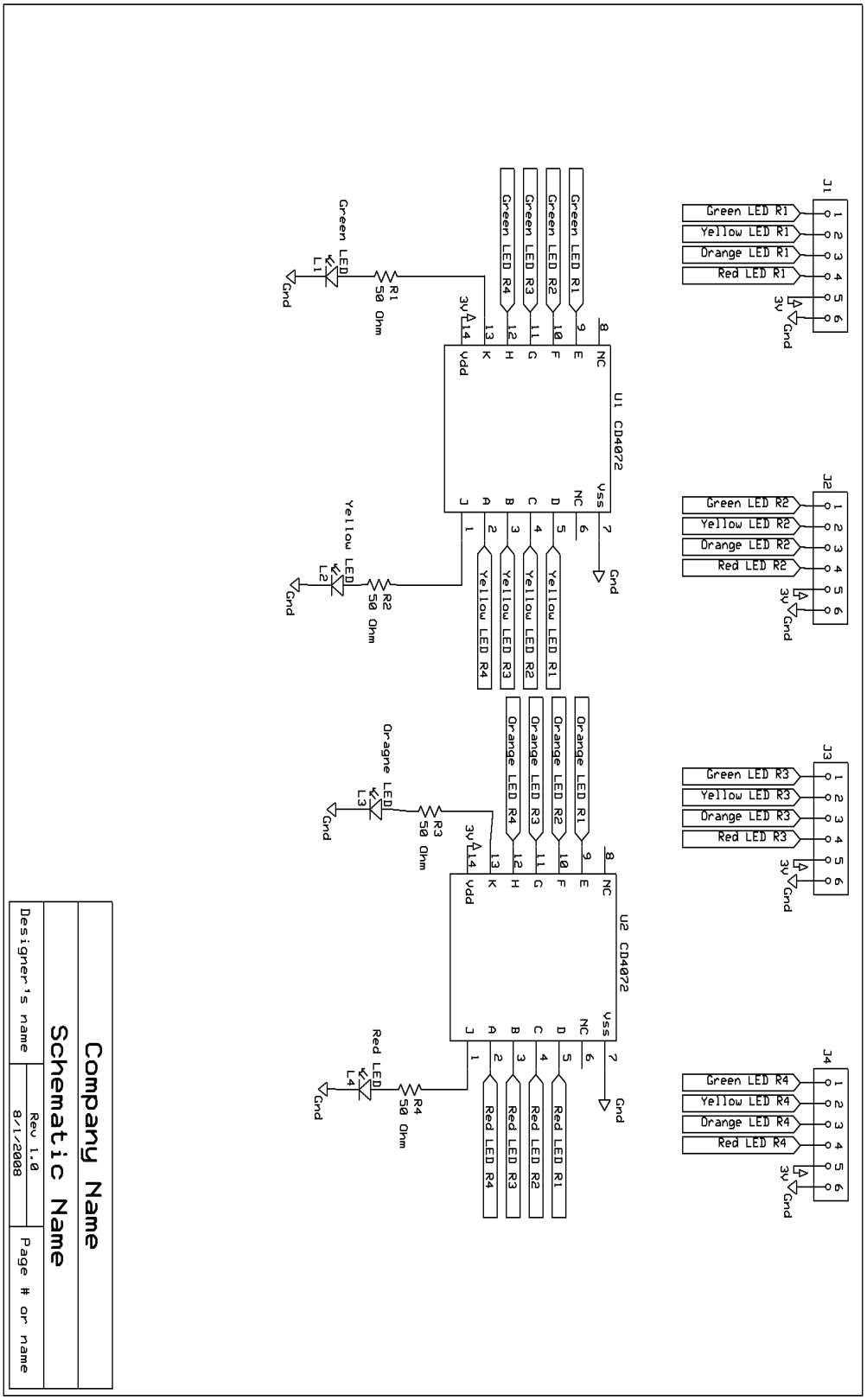
A1. Sensor Unit Schematic



A2. Receiver Schematic



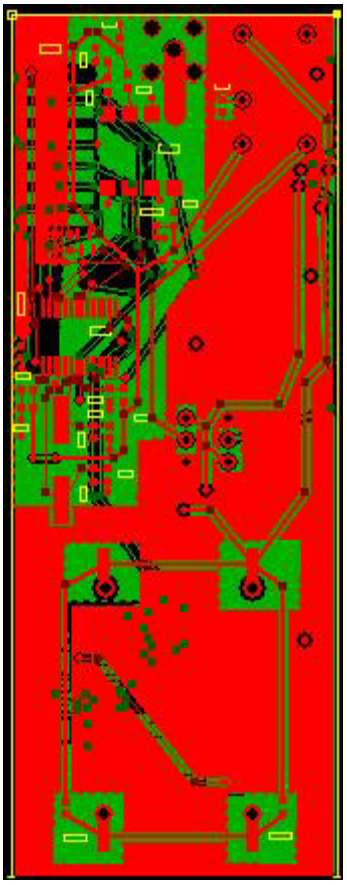
A3. Light Indicator Schematic



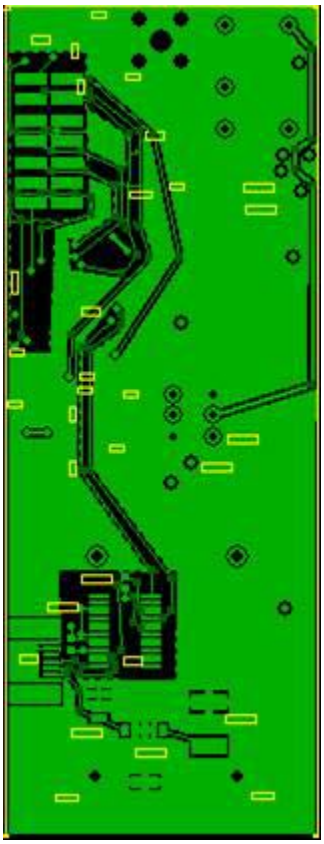
PCB DESIGN FILES

A4. Sensor Unit PCB Design

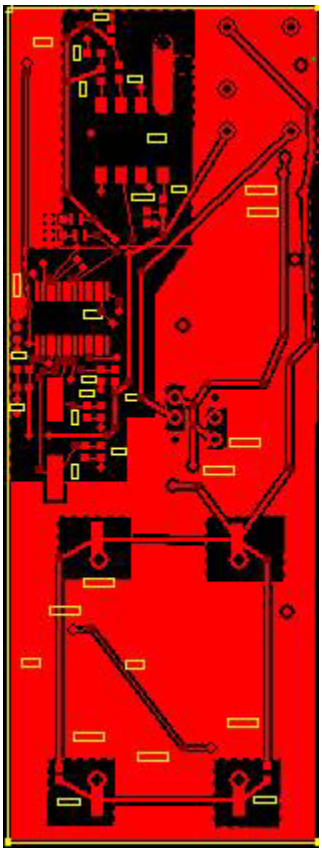
The following figure shows the layout of the Sensor Unit PCB with the top and bottom layers shown.



The following figure shows the layout of the Sensor Unit PCB with only the bottom layer shown.

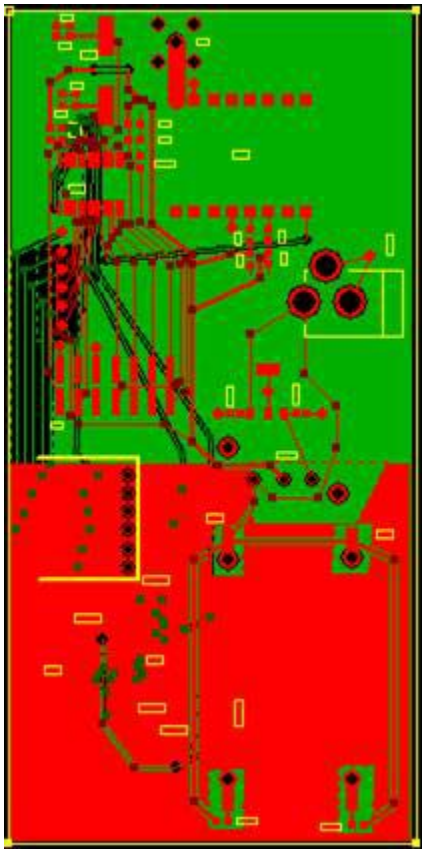


The following figure shows the layout of the Sensor Unit PCB with only the top layer shown.

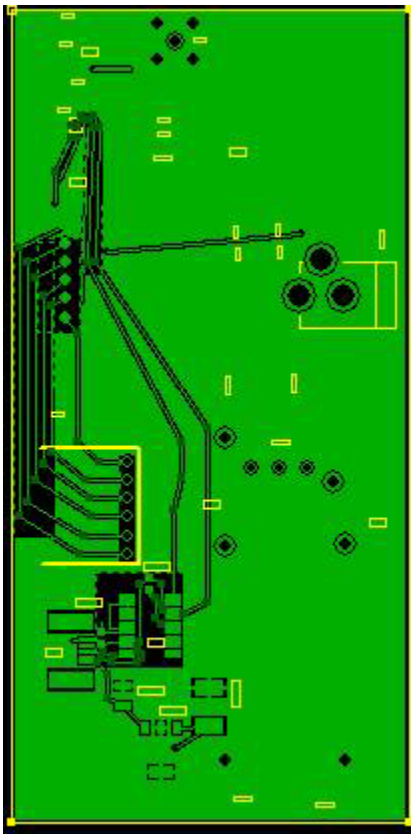


A5. Receiver Unit PCB Design

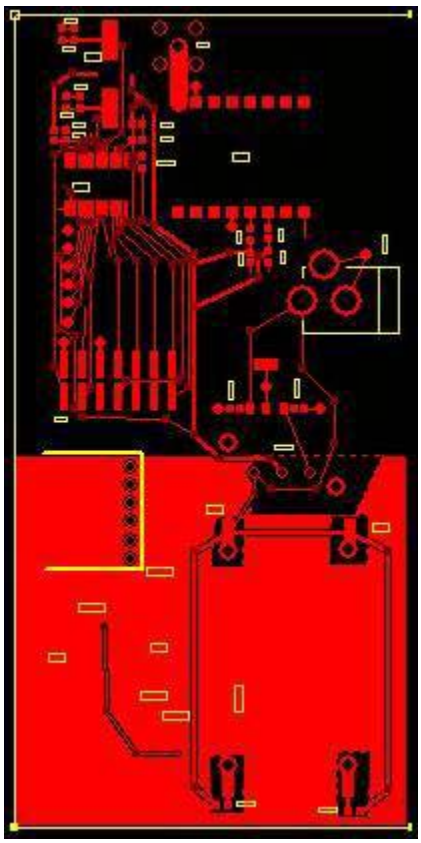
The following figure shows the layout of the Receiver Unit PCB with the top and bottom layers shown.



The following figure shows the layout of the Receiver Unit PCB with only the bottom layer shown.

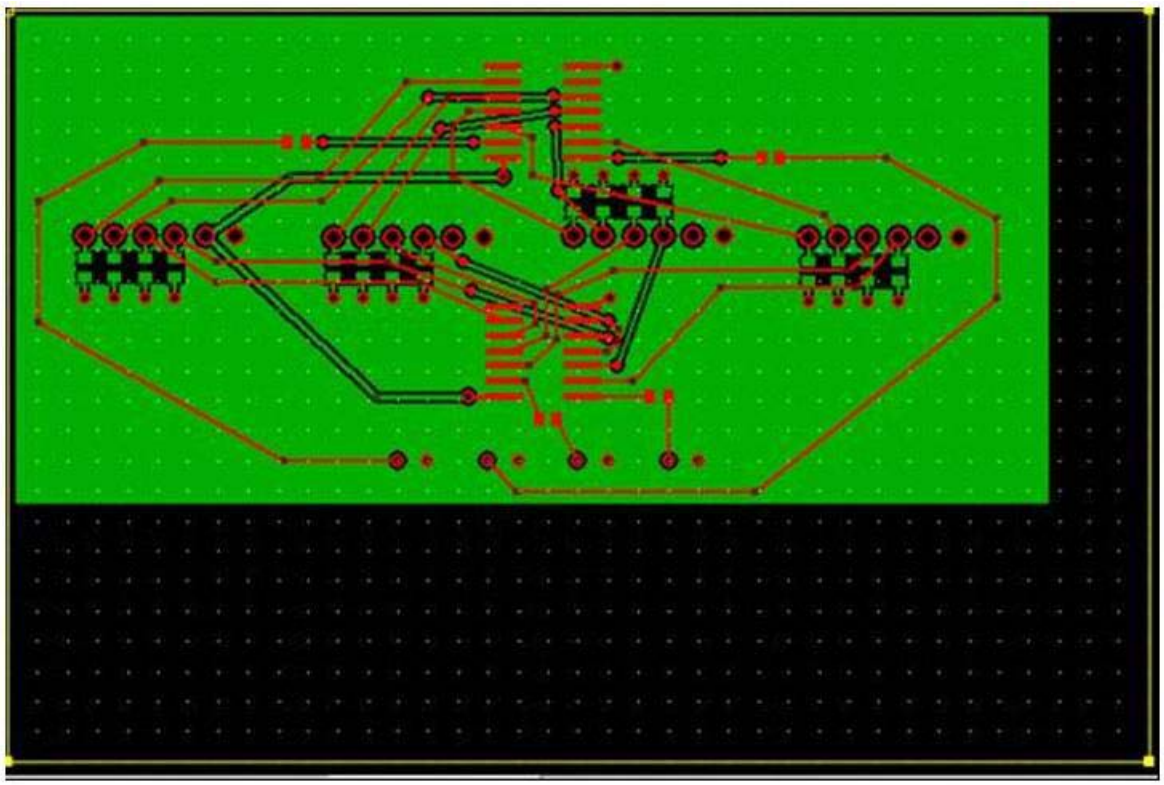


The following figure shows the layout of the Receiver Unit PCB with only the top layer shown.

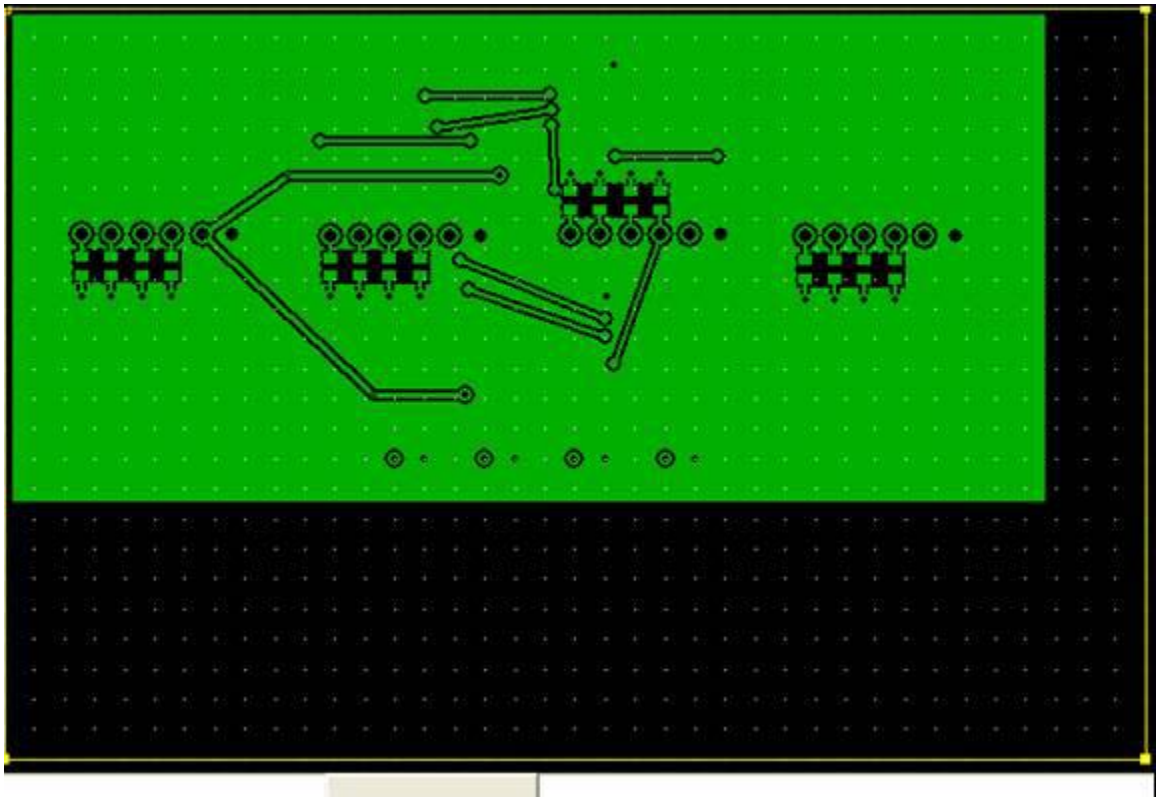


A6. Light Indicator PCB Design

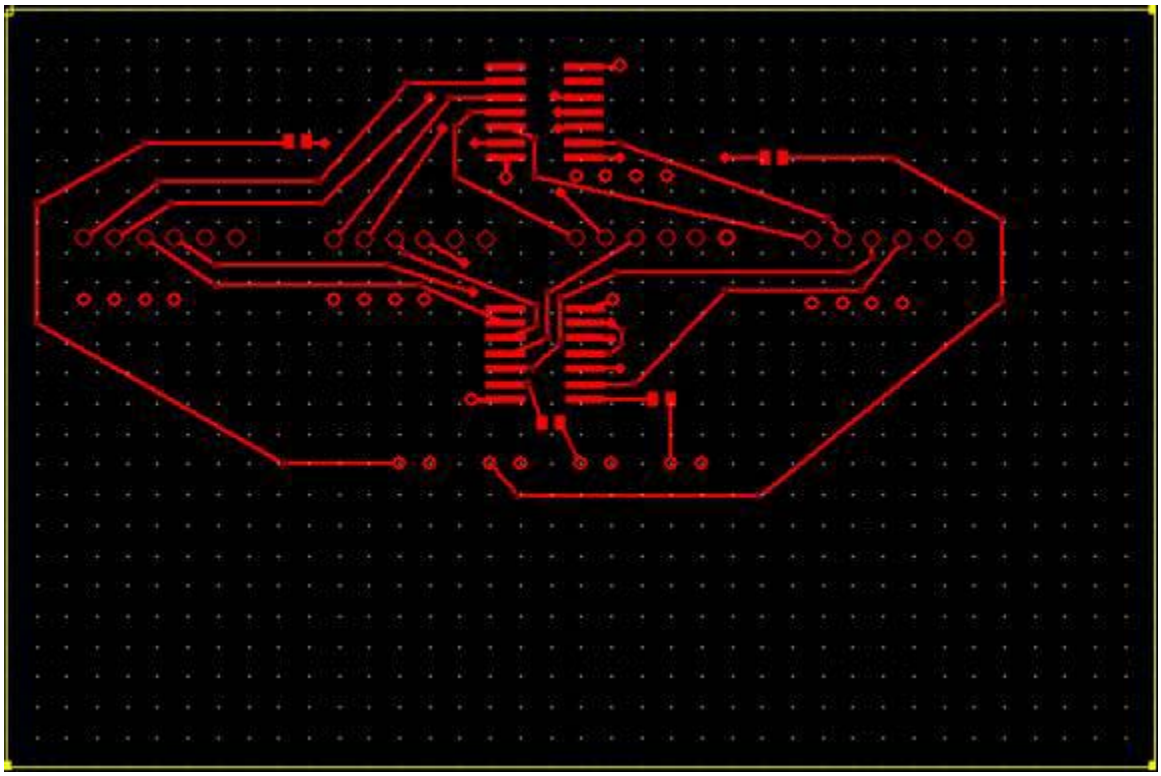
The following figure shows the layout of the Light Indicator PCB with the top and bottom layers shown.



The following figure shows the layout of the Light Indicator PCB with only the bottom layer shown.



The following figure shows the layout of the Light Indicator PCB with only the top layer shown.



APPENDIX B

SOFTWARE

B1. Transmitter C code

```
#include "msp430x21x2.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#define POLYNOMIAL 0x88108000  /*      CCITT polynomial (x^16 + x^12 + x^5 + 1)  followed by 0's */
#define POLY 0x8408#define Red_Priority 4
#define Orange_Priority 8
#define Yellow_Priority 12
#define Green_Priority 16


//GLOBAL VARIABLES
unsigned long dataToOutput_bit[17];
unsigned long bitIndex  = 0x00;
int     arrayIndex        = 0;
int     UART_rx_index =0;
int     ch_index = 0;
int     not_serial = 1;
int     UART_mode = 0;
int     priority;
int     mask;
int     i;
int     j;
int     mainflag;
int     ISR_flag;
int     bit_next;
int     bit_beginning;
int     bit_mid;
int     bit_end_level;
int     delay;
uint16_t crc;
char    *ptr;
char    data;
char    current_data_char = 0x00;
char    last_rx_char = 'a';
char    *Flash_ptr = (char *)0x1080;              // Initialize Flash pointer
char    my_array[21];
char    ch_buffer[21];
char    my_byte[2];

//FUNCTION PROTOTYPES
              void configTxTimer();
              void configDataOutputLine();
              void config_Clocks();
              void config_UART();
              void print_SegBtoUART();
              void print_byte_to_UART(char x);
              void write_SegB(char *data_p, int _crc,int length);
              void read_SegB_data_byte();
   uint16_t crc16(char *data_p,unsigned short length);
//END (FUNCTION PROTOTYPES)
```

```
////////////////////////////START MAIN/////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////



void main(void)
{
 uint32_t k,l, loop_limit;
 int delay_loop_choice, transmit_loop;



//DUMMY DATABLOCK
char bridge_ID_Byte_1 = 0xAA;
char bridge_ID_Byte_2 = 0xAA;
char bridge_ID_Byte_3 = 0xAA;
char bridge_ID_Byte_4 = 0xAA;
char bridge_ID_Byte_5 = 0xAA;
char bridge_ID_Byte_6 = 0xAA;
char bridge_ID_Byte_7 = 0xAA;
char bridge_ID_Byte_8 = 0xAA;
char bridge_ID_Byte_9 = 0xAA;
char bridge_ID_Byte_10 = 0xAA;
char bridge_ID_Byte_11 = 0xAA;
char bridge_ID_Byte_12 = 0xAA;
char bridge_ID_Byte_13 = 0xAA;
char bridge_ID_Byte_14 = 0xAA;
char serial_Num1 = 0xAA;
char serial_Num2 = 0xAA;
char location_1 = 0xAA;
char location_2 = 0xAA;
char location_3 = 0xAA;
char location_4 = 0xAA;
char location_5 = 0xAA;

my_array[0] = bridge_ID_Byte_1;
my_array[1] = bridge_ID_Byte_2;
my_array[2] = bridge_ID_Byte_3 ;
my_array[3] = bridge_ID_Byte_4;
my_array[4] = bridge_ID_Byte_5 ;
my_array[5] = bridge_ID_Byte_6 ;
my_array[6] =  bridge_ID_Byte_7;
my_array[7] =  bridge_ID_Byte_8 ;
my_array[8] =  bridge_ID_Byte_9 ;
my_array[9] =  bridge_ID_Byte_10;
my_array[10] =  bridge_ID_Byte_11 ;
my_array[11] =  bridge_ID_Byte_12;
my_array[12] =  bridge_ID_Byte_13;
my_array[13] = bridge_ID_Byte_14;
my_array[14] =  serial_Num1 ;
my_array[15] =  serial_Num2;
my_array[16] =  location_1 ;
my_array[17] =  location_2 ;
my_array[18] =  location_3 ;
my_array[19] =  location_4 ;
my_array[20] =  location_5;
ptr = &my_array[0];

            mask = 128; //used to determine bit within byte
            bit_beginning = 1;  //signifies beginning of new bit period
            bit_mid = 0;  //signifies middle of bit period
            bit_end_level = 1;  //level of bit at the end of last bit period
            delay = 0;  //control signal for synchonizing bit
            transmit_loop= 0;  //ensures two message transmissions take place before delays are applied

            WDTCTL = WDTPW + WDTHOLD; //disables watchdog timer
```

```
        //Configure uP registers
        config_Clocks();
        configTxTimer();
        configDataOutputLine();

        config_UART();

        _BIS_SR(GIE);


//calculates crc from data
crc = crc16(ptr, 21);
//write dummy block to flash memory location
write_SegB(ptr,crc, 23);


read_SegB_data_byte();

//intialize variables


        switch(location_5){
                case 0:
                priority = Green_Priority;
                break;
                case 1:
                priority = Yellow_Priority;
                break;
                case 2:
                priority = Orange_Priority;
                break;
                case 3:
                priority = Red_Priority;
                break;
                default:
                priority = Green_Priority;
                break;
        }


        srand((unsigned int) (serial_Num1 + serial_Num2));

        while(not_serial)
        {


       if(ISR_flag == 1)
       {

        ISR_flag = 0;

                                                        if(bit_beginning == 1)
                                                          {
                                                                bit_beginning = 0;
                                                                bit_mid = 1;
                                                          mask = mask /2 ;     //change masking bit
                                                j++;
                                                          }
                                                         else
                                                          {
                                                                bit_beginning = 1;
                                                                bit_mid = 0;
                                                          }
                                                        if(j == 8)
                                                        {
                                                        j=0;
                                                        mask = 128;
                                                        if (Flash_ptr > (char *)0x1097)
```

```
                                                {
                                        Flash_ptr = (char *)0x107E;
                                        bit_beginning = 0;
                                        bit_mid = 0;
                                                TA0CCTL0 &= !CCIE;

                                                if(transmit_loop %2 == 1)
                                                {
                                                        delay_loop_choice = rand() % priority;
                                                                                //delay_loop_choice = 3;
                        if(delay_loop_choice == 0  || delay_loop_choice == 1  || delay_loop_choice == 4 || delay_loop_choice == 8)
                                                                                {
                                                                                        loop_limit
= 50000;
                                                                                for(k = 0; k
<loop_limit; k++)
                                                                                 P1OUT &= 0x00;
//  set P1.0 to 0
                                                                                }
                                                                                else if(delay_loop_choice ==
2  || delay_loop_choice == 5  || delay_loop_choice == 6 || delay_loop_choice == 12 )
                                                                                {
                                                                                        loop_limit = 117000;
                                                                                        for(k = 0; k
<loop_limit; k++)
                                                                                 P1OUT &= 0x00;
//  set P1.0 to 0
                                                                                }
                                                                                        else
if(delay_loop_choice == 3  || delay_loop_choice == 13   || delay_loop_choice == 9 || delay_loop_choice == 10 )
                                                                                {
                                                                                        loop_limit = 275000;
                                                                                        for(k = 0; k
<loop_limit; k++)
                                                                                 P1OUT &= 0x00;
//  set P1.0 to 0
                                                                                }
                                                                                else if(delay_loop_choice ==
14  || delay_loop_choice == 15  || delay_loop_choice == 11 || delay_loop_choice == 7)
                                                                                {
                                                                                        loop_limit = 550000;
                                                                                        for(k = 0; k
<loop_limit; k++)
                                                                                 P1OUT &= 0x00;
//  set P1.0 to 0
                                                                                }
                                                                                        //for(k = 0; k <1000;
k++)
                                                                                 // P1OUT |= 0x01;
//  set P1.0 to 0
                                                        }
                                                        TA0CCR0  = 18000;
                                                        TA0CCTL0 = CCIE;
                                        P1OUT |= 0x01;      //  set P1.0 to 1
                                                delay = 1;
                                                transmit_loop++;

                                                 }
                                        read_SegB_data_byte();

                                        }
                                mainflag = 1;
        }

         }
}

//////////////////////////////START TIMER 0_A0//////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
```

```
void config_Clocks()
{
            unsigned long i;
            if (CALBC1_8MHZ ==0xFF || CALDCO_8MHZ == 0xFF)
 {
  while(1);                         // If calibration constants erased
                                    // do not load, trap CPU!!
 }
 BCSCTL1 = CALBC1_8MHZ;                  // Set DCO to 1\16MHz
 DCOCTL = CALDCO_8MHZ;
 FCTL2 = FWKEY + FSSEL0 + FN5 + FN3 + FN2 + FN1;   // MCLK/24 for Flash Timing

            BCSCTL1 |= XTS;                       // ACLK = LFXT1 = HF XTAL
            BCSCTL2 |= SELM_3 + SELS;                     // MCLK = LFXT1 (safe)
            BCSCTL3 |= LFXT1S1 + XT2S1;                     // 3 – 16MHz crystal or resonator
            do
            {
            IFG1 &= ~OFIFG;                     // Clear OSCFault flag
                        for (i = 0xFF; i > 0; i--);           // Time for flag to set
            } while (IFG1 & OFIFG);                 // OSCFault flag still set?
}

void configTxTimer()
{
            TA0CTL = TACLR;
            TA0CCR0  = 800;                                   // 519d = 0x0207; try the hex value too
            TA0CTL   = TASSEL_1 + MC_1;
            TA0CCTL0 = CCIE;
}

void configDataOutputLine()
{


            P1SEL &= !BIT0;
            P1DIR |= 0x01;
            P1OUT &= !0x01;
}
void config_UART()
{
P3SEL = 0x30;                       // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_3;                 // SMCLK
  UCA0BR0 = 0xA0;                     // 8MHz 19200
  UCA0BR1 = 0x01;                     // 8MHz 19200
  UCA0MCTL = UCBRS0;                    // Modulation UCBRSx = 1
  UCA0CTL1 &= ~UCSWRST;                 // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                    // Enable USCI_A0 RX interrupt
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer0_A0(void)
{

            ISR_flag = 1;

            if ( bit_beginning == 1)
                        {

                                    if (mask & data)      //determine bit value
                                       bit_next = 1;
          else
             bit_next = 0;

          if (bit_next == 1)
             {
                 if(bit_end_level == 0)
                   P1OUT |= 0x01;      //  set P1.0 to 1

             }
```

```
        else
          {
              if(bit_end_level == 1)
                P1OUT &= ~0x01;      //  set P1.0 to 0
          }
                        }
             else if( bit_mid == 1)
                    {
                        if (bit_next == 1)
                            {
                        P1OUT &= ~0x01;      //  set P1.0 to 0
                        bit_end_level = 0;
                            }
                        else if ( bit_next == 0)
                            {
                            P1OUT |= 0x01;       //  set P1.0 to 1
                            bit_end_level = 1;
                            }

                    }
  else if(delay == 1)
      {
              TA0CCR0  = 800;
          delay = 0;
          bit_beginning = 1;
      }


              _NOP();
mainflag = 0;
}// end timerA0 interrupt

// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
int i;
char *ptr2;
char tx_finish[] = {'d','a','t','a','_','t','x','e','d','!'};
char readout[] = {'d','a','t','a','_','b','l','o','c','k',':'};

 // while (!(IFG2 & UCA0TXIFG));             // USCI_A0 TX buffer ready?
  //UCA0TXBUF = UCA0RXBUF;                   // TX -> RXed character

if(UCA0RXBUF == '#')
{
//print_byte_to_UART(0xAB);
for(i =0;i<11;i++)
{
                                                        while (!(IFG2 & UCA0TXIFG));             // USCI_A0 TX buffer ready?
                                                while (  (UCA0STAT & UCBUSY ));
                                                        UCA0TXBUF = readout[i];
}
print_SegBtoUART();
                                                        while (!(IFG2 & UCA0TXIFG));             // USCI_A0 TX buffer ready?
                                                while (  (UCA0STAT & UCBUSY ));
                                                        UCA0TXBUF = 10;
                                                        while (!(IFG2 & UCA0TXIFG));             // USCI_A0 TX buffer ready?
                                                while (  (UCA0STAT & UCBUSY ));
                                                        UCA0TXBUF = 13;
}
  if(UART_mode == 1)
  {
                                                    if(ch_index % 2 == 0)
                                                    {
                                                        switch(UCA0RXBUF){
                                                            case '0':
                                                            current_data_char = 0x00;
                                                            break;
                                                            case '1':
```

```
                    current_data_char = 0x10;
                    break;
                    case '2':
                    current_data_char = 0x20;
                    break;
                    case '3':
                    current_data_char = 0x30;
                    break;
                    case '4':
                    current_data_char = 0x40;
                    break;
                    case '5':
                    current_data_char = 0x50;
                    break;
                    case '6':
                    current_data_char = 0x60;
                    break;
                    case '7':
                    current_data_char = 0x70;
                    break;
                    case '8':
                    current_data_char = 0x80;
                    break;
                    case '9':
                    current_data_char = 0x90;
                    break;



                    case 'A':
                    current_data_char = 0xA0;
                    break;
                    case 'B':
                    current_data_char = 0xB0;
                    break;
                    case 'C':
                    current_data_char = 0xC0;
                    break;
                    case 'D':
                    current_data_char = 0xD0;
                    break;
                    case 'E':
                    current_data_char = 0xE0;
                    break;
                    case 'F':
                    current_data_char = 0xF0;
                    break;
                    default:
                    current_data_char = 0x00;
                    break;
            }


            }

if(ch_index % 2 == 1)
{
            switch(UCA0RXBUF){
                    case '0':
                    current_data_char += 0x00;
                    break;
                    case '1':
                    current_data_char += 0x01;
                    break;
                    case '2':
                    current_data_char += 0x02;
                    break;
                    case '3':
```

```
                                        current_data_char += 0x03;
                                        break;
                                        case '4':
                                        current_data_char += 0x04;
                                        break;
                                        case '5':
                                        current_data_char += 0x05;
                                        break;
                                        case '6':
                                        current_data_char += 0x06;
                                        break;
                                        case '7':
                                        current_data_char += 0x07;
                                        break;
                                        case '8':
                                        current_data_char += 0x08;
                                        break;
                                        case '9':
                                        current_data_char += 0x09;
                                        break;



                                        case 'A':
                                        current_data_char += 0x0A;
                                        break;
                                        case 'B':
                                        current_data_char += 0x0B;
                                        break;
                                        case 'C':
                                        current_data_char += 0x0C;
                                        break;
                                        case 'D':
                                        current_data_char += 0x0D;
                                        break;
                                        case 'E':
                                        current_data_char += 0x0E;
                                        break;
                                        case 'F':
                                        current_data_char += 0x0F;
                                        break;
                                        default:
                                        current_data_char += 0x00;
                                        break;
                                }

                                 ch_buffer[UART_rx_index] = current_data_char;

                                        UART_rx_index++;
                                }

        ch_index++;
if(UART_rx_index ==21)
{
        //ptr2 = &ch_buffer[0];
//calculates crc from data

//write_SegB(ptr2,crc, 23);
  ptr2 = (char *) 0x1080;
  crc = crc16((char *) 0x1080, 21);
  FCTL3 = FWKEY;                        // Clear Lock bit
        FCTL1 = FWKEY + ERASE;                  // Set Erase bit
        *ptr2 = 0;

        FCTL1 = FWKEY + WRT;
        for(i = 0; i<21;i++)
        *ptr2++ = ch_buffer[i];
```

```
                    *ptr2++ = (char) (crc >> 8);
                    *ptr2++ = (char) crc;


                      FCTL1 = FWKEY;                    // Clear WRT bit
                    FCTL3 = FWKEY + LOCK;               // Set LOCK bit
                UART_mode = 0;
                UART_rx_index = 0;
                ch_index = 0;
                                for( i=0; i<10;i++)
                                {
                                    while (!(IFG2 & UCA0TXIFG));            // USCI_A0 TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = tx_finish[i];               // TX character
                                }
                                                while (!(IFG2 & UCA0TXIFG));            // USCI_A0 TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = 10;
                                                while (!(IFG2 & UCA0TXIFG));            // USCI_A0 TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = 13;
                TA0CCTL0 = CCIE;
                not_serial = 1;
                }

                }



                 if (UCA0RXBUF == '*'  && last_rx_char == '*')
                            {

                                    UART_mode = 1;
                                    TA0CCTL0 &= !CCIE;
                                    not_serial = 0;
                            }
 last_rx_char = UCA0RXBUF;

}

void write_SegB(char *data_p, int _crc,int length)
{
  char *This_Flash_ptr;  // Flash pointer
  uint8_t temp;
  unsigned int i;
    /*
     *place bytes in array
     *
     */
   char data_byte[23];
     data_byte[0] = data_p[0];
     data_byte[1] =  data_p[1];
     data_byte[2] = data_p[2];
     data_byte[3] = data_p[3];
     data_byte[4] = data_p[4];
     data_byte[5] = data_p[5];
     data_byte[6] = data_p[6];
     data_byte[7] = data_p[7];
     data_byte[8] = data_p[8];
     data_byte[9] = data_p[9];
     data_byte[10] = data_p[10];
     data_byte[11] = data_p[11];
     data_byte[12] = data_p[12];
     data_byte[13] = data_p[13];
     data_byte[14] = data_p[14];
     data_byte[15] = data_p[15];
     data_byte[16] = data_p[16];
     data_byte[17] = data_p[17];
     data_byte[18] = data_p[18];
     data_byte[19] = data_p[19];
```

```
    data_byte[20] = data_p[20];
    data_byte[21] = (char) (_crc >> 8);
    data_byte[22] = (char) _crc;




  This_Flash_ptr = (char *)0x107E;              // Initialize Flash pointer
  FCTL3 = FWKEY;                          // Clear Lock bit
  FCTL1 = FWKEY + ERASE;                  // Set Erase bit
  *This_Flash_ptr = 0;                     // Dummy write to erase Flash seg

  FCTL1 = FWKEY + WRT;                    // Set WRT bit for write operation
              *This_Flash_ptr++ = 0xAA;
              *This_Flash_ptr++ = 0x55;
  for (i = 0; i < 23; i++)
  {
    *This_Flash_ptr++ = data_byte[i];                // Write value to flash
  }

  FCTL1 = FWKEY;                          // Clear WRT bit
  FCTL3 = FWKEY + LOCK;                   // Set LOCK bit
  _NOP();
}


void read_SegB_data_byte()
{

data =        *Flash_ptr;
*Flash_ptr++;
}

void print_SegBtoUART()
{
char *print_ptr;
int i;

print_ptr = (char *) 0x1080;

for(i=0;i<23;i++)
{
print_byte_to_UART(*print_ptr++);
                                                    }
}

void print_byte_to_UART(char x)
{
int current_data_char;
char second_nibble_left_4 = (x << 4);
char second_nibble_iso = second_nibble_left_4 >> 4;
char first_nibble_iso = x >> 4;
              switch( first_nibble_iso){
                                                                    case 0:
                                                                    current_data_char = 48;
                                                                    break;
                                                                    case 1:
                                                                    current_data_char = 49;
                                                                    break;
                                                                    case 2:
                                                                    current_data_char = 50;
                                                                    break;
                                                                    case 3:
                                                                    current_data_char = 51;
                                                                    break;
                                                                    case 4:
```

```
                                current_data_char = 52;
                                break;
                                case 5:
                                current_data_char = 53;
                                break;
                                case 6:
                                current_data_char = 54;
                                break;
                                case 7:
                                current_data_char = 55;
                                break;
                                case 8:
                                current_data_char = 56;
                                break;
                                case 9:
                                current_data_char = 57;
                                break;




                                case 10:
                                current_data_char = 65;
                                break;
                                case 11:
                                current_data_char = 66;
                                break;
                                case 12:
                                current_data_char = 67;
                                break;
                                case 13:
                                current_data_char = 68;
                                break;
                                case 14:
                                current_data_char = 69;
                                break;
                                case 15:
                                current_data_char = 70;
                                break;
                                default:
                                current_data_char = 0x00;
                                break;
                        }

                while (!(IFG2 & UCA0TXIFG));            // USCI_A0 TX buffer ready?
while (  (UCA0STAT & UCBUSY ));
UCA0TXBUF = current_data_char;

switch(second_nibble_iso){
                                case 0:
                                current_data_char = 48;
                                break;
                                case 1:
                                current_data_char = 49;
                                break;
                                case 2:
                                current_data_char = 50;
                                break;
                                case 3:
                                current_data_char = 51;
                                break;
                                case 4:
                                current_data_char = 52;
                                break;
                                case 5:
                                current_data_char = 53;
                                break;
                                case 6:
                                current_data_char = 54;
```

```
                                        break;
                                        case 7:
                                        current_data_char = 55;
                                        break;
                                        case 8:
                                        current_data_char = 56;
                                        break;
                                        case 9:
                                        current_data_char = 57;
                                        break;




                                        case 10:
                                        current_data_char = 65;
                                        break;
                                        case 11:
                                        current_data_char = 66;
                                        break;
                                        case 12:
                                        current_data_char = 67;
                                        break;
                                        case 13:
                                        current_data_char = 68;
                                        break;
                                        case 14:
                                        current_data_char = 69;
                                        break;
                                        case 15:
                                        current_data_char = 70;
                                        break;
                                        default:
                                        current_data_char = 0x00;
                                        break;
                                }

                                        while (!(IFG2 & UCA0TXIFG));            // USCI_A0 TX buffer ready?
                                while (  (UCA0STAT & UCBUSY ));
                                UCA0TXBUF = current_data_char;
}


                // The CCITT CRC 16 polynomial is X + X + X + 1.
// In binary, this is the bit pattern 1 0001 0000 0010 0001, and in hex it
//  is 0x11021.
// A 17 bit register is simulated by testing the MSB before shifting
//  the data, which affords us the luxury of specifiy the polynomial as a
//  16 bit value, 0x1021.
// Due to the way in which we process the CRC, the bits of the polynomial
//  are stored in reverse order. This makes the polynomial 0x8408.
*/



/*
// note: when the crc is included in the message, the valid crc is:
//      0xF0B8, before the compliment and byte swap,
//      0x0F47, after compliment, before the byte swap,
//      0x470F, after the compliment and the byte swap.
*/


/**************************************************************************
//
// crc16() - generate a 16 bit crc
//
//
// PURPOSE
```

```
//      This routine generates the 16 bit remainder of a block of
//      data using the ccitt polynomial generator.
//
// CALLING SEQUENCE
//      crc = crc16(data, len);
//
// PARAMETERS
//      data    <-- address of start of data block
//      len     <-- length of data block
//
// RETURNED VALUE
//      crc16 value. data is calcuated using the 16 bit ccitt polynomial.
//
// NOTES
//      The CRC is preset to all 1's to detect errors involving a loss
//        of leading zero's.
//      The CRC (a 16 bit value) is generated in LSB MSB order.
//      Two ways to verify the integrity of a received message
//        or block of data:
//        1) Calculate the crc on the data, and compare it to the crc
//           calculated previously. The location of the saved crc must be
//           known.
/         2) Append the calculated crc to the end of the data. Now calculate
//           the crc of the data and its crc. If the new crc equals the
//           value in "crc_ok", the data is valid.
//
// PSEUDO CODE:
//      initialize crc (-1)
//      DO WHILE count NE zero
//        DO FOR each bit in the data byte, from LSB to MSB
//          IF (LSB of crc) EOR (LSB of data)
//            crc := (crc / 2) EOR polynomial
//          ELSE
//            crc := (crc / 2)
//          FI
//        OD
//      OD
//      1's compliment and swap bytes in crc
//      RETURN crc
//
*****************************************************************************/



uint16_t crc16(char *data_p,unsigned short length)
{
      unsigned char i;
      uint32_t data;
      uint32_t crc;

      crc = 0xFFFF;

      if (length == 0)
            return (~crc);

      do {
            for (i = 0, data = (unsigned int)0xff & *data_p++;
               i < 8;
               i++, data >>= 1) {
                 if ((crc & 0x0001) ^ (data & 0x0001))
                       crc = (crc >> 1) ^ POLY;
                 else
                       crc >>= 1;
            }
      } while (--length);

      crc = ~crc;

      data = crc;
```

```
    crc = (crc << 8) | (data >> 8 & 0xFF);

    return (crc);
}
```

B2. Receiver C Code

```
#include "msp430x21x2.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#define POLYNOMIAL 0x88108000  /*    CCITT polynomial (x^16 + x^12 + x^5 + 1)  followed by 0's
*/
#define POLY 0x8408



unsigned int new_cap=0;
unsigned int old_cap=0;
unsigned int cap_diff=0;
unsigned int store,index=0;
unsigned int j=0;
unsigned int k=0;
unsigned int mode =0;
unsigned int half_count = 0;
unsigned int bit_count = 0;
unsigned int byte_count = 0;
unsigned int mask = 128;
unsigned int bit_prev = 0;
unsigned int block_finished;
unsigned int current_bit = 0;
unsigned int skip_trans = 0;
unsigned int skip_set =0;
unsigned int diff_array[150];            // RAM array for differences
int hit_red = 0;
int hit_orange = 0;
int hit_yellow = 0;
int hit_green = 0;
int stored_byte_count = 0;
uint32_t i;
int UART_mode = 0;
int            UART_rx_index =0;
int            ch_index = 0;
int            not_serial = 1;
char data_byte = 0x00;
char data;
char current_data_char = 0x00;
char last_rx_char = 'a';
char data_block[25];
char ch_buffer[14];
char ID_array[14];
char *Flash_ptr = (char *)0x1000;              // Initialize Flash pointer
//FUNCTION PROTOTYPES


       void config_Clocks();
    void config_UART();
    void config_RxTimer();
    void print_BridgeID();
       void print_BlockstoUART();
       void print_byte_to_UART(char x);
    void store_and_process();
    int BridgeID_Repeat();
    uint16_t crc16(char *data_p,unsigned short length);
//END (FUNCTION PROTOTYPES)
```

```
//////////////////////////START MAIN/////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////



void main(void)
{
       char *ptr;
    uint16_t crc;
       WDTCTL = WDTPW + WDTHOLD;

       config_Clocks();
       config_UART();
    config_RxTimer();


       for(i = 0; i < 25;i++)
              data_block[i] = 0x00;
       for(i = 0; i < 150; i++)
       {
              diff_array[i] = 0x0000;
              //capture_array[i] = 0x0000;
       }
              //Outputs SMCLK through 2.1 (pin 9) and ACLK through 2.0 (pin 8)
       //P1DIR = P1_RFSLE + P1_RFSDATA + P1_RFSCLK + P1_RFCE;                     //Sets Port1
GIO
       P1SEL = 0x02;  //Connects RFDATA to Timer0_A0
       P1DIR |= 0x01;
       P2DIR |= 0x18;
       P3DIR |= 0x80;
       P1OUT &= !(0x01);

       P2OUT &= !(0x18);
       P3OUT &= !(0x80);
  //   P2OUT |= 0x10;
       //P3OUT |= 0x80;
    _NOP();
       //Enable Interrupt for RFDATACLK
       //P1IE |= P1_RFDATACLK;
       //P1IFG &= P1_RFDATACLK;
 for(i = 0; i < 100000; i++)
       P1OUT |= 0x01;

       P1OUT &= !(0x01);
       while(1)
       {
       if(mode == 2)
                     {
                            ptr = &data_block[2];
                            crc = crc16(ptr, 23);
                                if(crc == 0x470F)
                    {
                    //stores data and toggles LED lines

                if(BridgeID_Repeat()  == 1)
                     store_and_process();
                    }
                    mode = 0;                     //reset mode
                                CCTL0  = CM_3 + CCIS_0 + CAP + CCIE; //Rising Edge, Snyc Capture
Mode,
                                                                                    // Capture
on Input pin P1.1 Capture Mode,
                //Intterupts Enabled
                                       TACTL = MC_2 + TASSEL_1 + TACLR;
//SMCLK, Continuous Modess
                     }

       }
}
```

```
//////////////////////////////START TIMER 0_A0//////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////


void config_Clocks()
{
       unsigned long i;
       if (CALBC1_8MHZ ==0xFF || CALDCO_8MHZ == 0xFF)
  {
    while(1);                               // If calibration constants erased
                                            // do not load, trap CPU!!
  }
  BCSCTL1 = CALBC1_8MHZ;                    // Set DCO to 1\16MHz
  DCOCTL = CALDCO_8MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN5 + FN3 + FN2 + FN1;   // MCLK/24 for Flash Timing

       BCSCTL1 |= XTS;                           // ACLK = LFXT1 = HF XTAL
       BCSCTL2 |= SELM_3 + SELS;                       // MCLK = LFXT1 (safe)
       BCSCTL3 |= LFXT1S1 + XT2S1;                      // 3 – 16MHz crystal or resonator
       do
       {
       IFG1 &= ~OFIFG;                          // Clear OSCFault flag
               for (i = 0xFF; i > 0; i--);            // Time for flag to set
       } while (IFG1 & OFIFG);                     // OSCFault flag still set?
}

void config_UART()
{
P3SEL = 0x30;                             // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_3;                     // SMCLK
  UCA0BR0 = 0xA0;                             // 8MHz 19200
  UCA0BR1 = 0x01;                             // 8MHz 19200
  UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
}

void config_RxTimer()
{

       CCTL0  = CM_3 + CCIS_0 + CAP + CCIE; //Rising Edge, Snyc Capture Mode,
                                                                                    // Capture
on Input pin P1.1 Capture Mode,
                //Intterupts Enabled
       TACTL = MC_2 + TASSEL_1 + TACLR;                          //SMCLK, Continuous Modess
       _BIS_SR(GIE);                                             //enable global interrupt;
}

// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
int i, k = 1;
char *ptr2;
char tx_finish[] = {'d','a','t','a','_','t','x','e','d','!'};
char reset[] = {'R','e','c','e','i','v','e','r','_','R','e','s','e','t'};

if(UCA0RXBUF == '&'  && last_rx_char == '&')
{
       hit_green =0;
hit_yellow = 0;
hit_orange = 0;
hit_red = 0;

       P1OUT &= !(0x01);

       P2OUT &= !(0x18);
       P3OUT &= !(0x80);

for(i = 0;i<14;i++)
```

```
        {
                                                        while (!(IFG2 & UCA0TXIFG));              //
USCI_A0 TX buffer ready?
                                        while (  (UCA0STAT & UCBUSY ));
                                                        UCA0TXBUF = reset[i];
        }

}
 // while (!(IFG2 & UCA0TXIFG));              // USCI_A0 TX buffer ready?
  //UCA0TXBUF = UCA0RXBUF;                    // TX -> RXed character

if(UCA0RXBUF == '1'&& last_rx_char == '#')
{
print_BridgeID();
                                                while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                        while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = 10;
                                                while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                        while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = 13;
}
if(UCA0RXBUF == '2'  && last_rx_char == '#')
{


print_BlockstoUART();
                                                while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                        while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = 10;
                                                while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                        while (  (UCA0STAT & UCBUSY ));
                                                UCA0TXBUF = 13;

}
  if(UART_mode == 1)
  {

                                       if(ch_index % 2 == 0)
                                       {
                                                switch(UCA0RXBUF){
                                                        case '0':
                                                        current_data_char = 0x00;
                                                        break;
                                                        case '1':
                                                        current_data_char = 0x10;
                                                        break;
                                                        case '2':
                                                        current_data_char = 0x20;
                                                        break;
                                                        case '3':
                                                        current_data_char = 0x30;
                                                        break;
                                                        case '4':
                                                        current_data_char = 0x40;
                                                        break;
                                                        case '5':
                                                        current_data_char = 0x50;
                                                        break;
                                                        case '6':
                                                        current_data_char = 0x60;
                                                        break;
                                                        case '7':
                                                        current_data_char = 0x70;
                                                        break;
                                                        case '8':
                                                        current_data_char = 0x80;
                                                        break;
                                                        case '9':
```

```
                current_data_char = 0x90;
                break;




                case 'A':
                current_data_char = 0xA0;
                break;
                case 'B':
                current_data_char = 0xB0;
                break;
                case 'C':
                current_data_char = 0xC0;
                break;
                case 'D':
                current_data_char = 0xD0;
                break;
                case 'E':
                current_data_char = 0xE0;
                break;
                case 'F':
                current_data_char = 0xF0;
                break;
                default:
                current_data_char = 0x00;
                break;
        }


        }

if(ch_index % 2 == 1)
{
        switch(UCA0RXBUF){
                case '0':
                current_data_char += 0x00;
                break;
                case '1':
                current_data_char += 0x01;
                break;
                case '2':
                current_data_char += 0x02;
                break;
                case '3':
                current_data_char += 0x03;
                break;
                case '4':
                current_data_char += 0x04;
                break;
                case '5':
                current_data_char += 0x05;
                break;
                case '6':
                current_data_char += 0x06;
                break;
                case '7':
                current_data_char += 0x07;
                break;
                case '8':
                current_data_char += 0x08;
                break;
                case '9':
                current_data_char += 0x09;
                break;
```

```
                                        case 'A':
                                        current_data_char += 0x0A;
                                        break;
                                        case 'B':
                                        current_data_char += 0x0B;
                                        break;
                                        case 'C':
                                        current_data_char += 0x0C;
                                        break;
                                        case 'D':
                                        current_data_char += 0x0D;
                                        break;
                                        case 'E':
                                        current_data_char += 0x0E;
                                        break;
                                        case 'F':
                                        current_data_char += 0x0F;
                                        break;
                                        default:
                                        current_data_char += 0x00;
                                        break;
                                  }

                                    ch_buffer[UART_rx_index] = current_data_char;

                                        UART_rx_index++;
                                  }

                        ch_index++;

                      if(UART_rx_index ==14)
                      {

                        //ptr2 = &ch_buffer[0];
                      //calculates crc from data

                      //write_SegB(ptr2,crc, 23);

                        ptr2 = (char *) 0x10B1;

                        FCTL3 = FWKEY;                          // Clear Lock bit
                            FCTL1 = FWKEY + ERASE;                  // Set Erase bit
                            *ptr2 = 0;

                            FCTL1 = FWKEY + WRT;
                            for(i = 0; i<14;i++)
                            {
                            *ptr2++ = ch_buffer[i];
                            ID_array[i] = ch_buffer[i];
                            }


                              FCTL1 = FWKEY;                          // Clear WRT bit
                            FCTL3 = FWKEY + LOCK;                   // Set LOCK bit

                      UART_mode = 0;
                      UART_rx_index = 0;
                      ch_index = 0;
                                    for( i=0; i<10;i++)
                                    {
                                          while (!(IFG2 & UCA0TXIFG));              // USCI_A0 TX
buffer ready?
                                          while (  (UCA0STAT & UCBUSY ));
                                                 UCA0TXBUF = tx_finish[i];                    // TX
character
                                    }
                                              TA0CCTL0 |= CCIE;
                                                   not_serial = 1;
                                                 while (!(IFG2 & UCA0TXIFG));             // USCI_A0
TX buffer ready?
                                          while (  (UCA0STAT & UCBUSY ));
```

```
                                          UCA0TXBUF = 10;

                                          while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                          UCA0TXBUF = 13;
                        UART_mode = 0;
                }


                }


                  if (UCA0RXBUF == '*'  && last_rx_char == '*')
                        {

                              UART_mode = 1;
                              //TA0CCTL0 &= !CCIE;
                              //not_serial =0;

                        }
  last_rx_char = UCA0RXBUF;

}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer0_A0(void)
{

       new_cap = (unsigned int) TACCR0;                                          //capture time
on edge
    cap_diff = new_cap - old_cap;
   // if (new_cap != old_cap)
    //{
    //capture_array[index] = new_cap;
    old_cap = new_cap;

    diff_array[index] = cap_diff;            // record difference to RAM array
    diff_array[0] = 4;
    index++;
    //}

      if(mode == 1 )
     {
              if( (cap_diff > 650) && (cap_diff < 950) && (bit_prev == 0))
              {

                            if(half_count % 2 == 1)
                            {
                            mask = mask / 2;
                            bit_count++;
                            }
              bit_prev = 0;
              half_count++;
              }
              else if( (cap_diff > 1450) && (cap_diff < 1750) && (bit_prev == 0))
              {
              data_byte = data_byte | mask;
              mask = mask / 2;
              bit_count++;
              bit_prev =1;
              skip_set = 0;
              half_count = 0;
              }
              else if( (cap_diff > 650) && (cap_diff < 950) && (bit_prev == 1))
              {
              data_byte = data_byte | mask;
                            if(half_count % 2 == 1)
                            {
                            mask = mask / 2;
```

```
                                bit_count++;
                                }
                skip_trans = 1;
                bit_prev = 1;
                half_count++;
                skip_set = 1;
                }
                else if( (cap_diff > 1450) && (cap_diff < 1750) && (bit_prev == 1))
                {
                mask = mask / 2;
                bit_count++;
                bit_prev = 0;
                skip_set = 0;
                half_count = 0;
                }
      }

  //  if(skip_trans ==1)
    //  skip_trans = 0;

    if(bit_count == 8)
        {
                data_block[byte_count] = data_byte;
                data_byte = 0x00;
                bit_count = 0;
                byte_count++;
                mask = 128;

        }
    if (data_block[0] == 0xAA)
       _NOP();
    if(byte_count == 25)
    {
        mode = 2;
                bit_count = 0;
                        byte_count = 0;
                        skip_trans = 0;
                        mask = 128;
        CCTL0 = 0;
        TACTL = 0;

    }
   if(cap_diff > 17500&& cap_diff < 30000)
   {
                mode = 1;
                bit_prev = 1;
            data_byte = data_byte | mask;
            mask = mask / 2;
            bit_count++;
            skip_trans = 0;

   }

   if(cap_diff > 30000)
   {
        mode = 0;
        bit_count = 0;
        byte_count = 0;
        skip_trans = 0;
        mask = 128;

   }


        if(index == 150)      //Is memory full?
   {
   index = 0;
   }
}// end ISR
```

```
void store_and_process()
{



  FCTL3 = FWKEY;                            // Clear Lock bit

  FCTL1 = FWKEY + WRT;                      // Set WRT bit for write operation

  for (i = 0; i < 25; i++)
  {
    *Flash_ptr++ = data_block[i];                   // Write value to flash
            stored_byte_count++;
                     if (data_block[22] == 0 && hit_red == 0 && hit_orange == 0 && hit_yellow ==
0)
                        {
                              P1OUT |= 0x01;                     // Set Green LED high, others
low
                              P2OUT &= !(0x18);
                              P3OUT &= !(0x80);
                              hit_green = 1;
                         }
                     else if (data_block[22] == 1 && hit_red == 0 && hit_orange == 0)
                        {

                              P1OUT &= !(0x01);                     // Set Orange LED high,
others low
                              P2OUT &= !(0x08);
                              P2OUT |= 0x10;

                              P3OUT &= !(0x80);
                              hit_yellow = 1;
                         }
                     else if (data_block[22] == 2 && hit_red == 0)
                        {
                              P1OUT &= !(0x01);                     // Set Yellow LED high,
others low
                              P2OUT &= !(0x10);
                              P2OUT |= 0x08;

                              P3OUT &= !(0x80);
                              hit_orange = 1;
                         }
                     else if (data_block[22] == 3)
                        {
                              P1OUT &= !(0x01);                     // Set Red LED high, others
low
                              P2OUT &= !(0x18);
                              P3OUT |= 0x80;
                              hit_red = 1;
                          }



}
  FCTL1 = FWKEY;                            // Clear WRT bit
  FCTL3 = FWKEY + LOCK;                     // Set LOCK bit
}
void print_BridgeID()
{
       char *print_ptr;
int i,j,k;
char readout[] = {'B','r','i','d','g','e','_','I','D',':'};

print_ptr = (char *) 0x10B1;
k=0;

               for(j =0;j<10;j++)
                       {
```

```
                                        while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                        UCA0TXBUF = readout[j];

                        }
for(i=0;i<14;i++)
{


print_byte_to_UART(*print_ptr++);
                                    }
}

void print_BlockstoUART()
{
char *print_ptr;
int i,j,k;
char readout[] = {'d','a','t','a','_','b','l','o','c','k'};

print_ptr = (char *) 0x1000;
k=48;
for(i=0;i<stored_byte_count;i++)
{
        if(i %25 == 0)
        {
                k++;
                for(j =0;j<10;j++)
                        {
                                        while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                        UCA0TXBUF = readout[j];
                        }
                                        while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                    UCA0TXBUF = k;

                                    while (!(IFG2 & UCA0TXIFG));              // USCI_A0 TX
buffer ready?
                                    while (  (UCA0STAT & UCBUSY ));
                                   UCA0TXBUF = ':';

        }
print_byte_to_UART(*print_ptr++);
                                    }
}


void print_byte_to_UART(char x)
{
int current_data_char;
char second_nibble_left_4 = (x << 4);
char second_nibble_iso = second_nibble_left_4 >> 4;
char first_nibble_iso = x >> 4;
        switch( first_nibble_iso){
                                                        case 0:
                                                        current_data_char = 48;
                                                        break;
                                                        case 1:
                                                        current_data_char = 49;
                                                        break;
                                                        case 2:
                                                        current_data_char = 50;
                                                        break;
                                                        case 3:
                                                        current_data_char = 51;
                                                        break;
                                                        case 4:
                                                        current_data_char = 52;
```

```
                                        break;
                                        case 5:
                                        current_data_char = 53;
                                        break;
                                        case 6:
                                        current_data_char = 54;
                                        break;
                                        case 7:
                                        current_data_char = 55;
                                        break;
                                        case 8:
                                        current_data_char = 56;
                                        break;
                                        case 9:
                                        current_data_char = 57;
                                        break;




                                        case 10:
                                        current_data_char = 65;
                                        break;
                                        case 11:
                                        current_data_char = 66;
                                        break;
                                        case 12:
                                        current_data_char = 67;
                                        break;
                                        case 13:
                                        current_data_char = 68;
                                        break;
                                        case 14:
                                        current_data_char = 69;
                                        break;
                                        case 15:
                                        current_data_char = 70;
                                        break;
                                        default:
                                        current_data_char = 0x00;
                                        break;
                                }

                                 while (!(IFG2 & UCA0TXIFG));             //
USCI_A0 TX buffer ready?
                                while (  (UCA0STAT & UCBUSY ));
                                UCA0TXBUF = current_data_char;

                                switch(second_nibble_iso){
                                        case 0:
                                        current_data_char = 48;
                                        break;
                                        case 1:
                                        current_data_char = 49;
                                        break;
                                        case 2:
                                        current_data_char = 50;
                                        break;
                                        case 3:
                                        current_data_char = 51;
                                        break;
                                        case 4:
                                        current_data_char = 52;
                                        break;
                                        case 5:
                                        current_data_char = 53;
                                        break;
                                        case 6:
                                        current_data_char = 54;
                                        break;
```

```
                                                case 7:
                                                current_data_char = 55;
                                                break;
                                                case 8:
                                                current_data_char = 56;
                                                break;
                                                case 9:
                                                current_data_char = 57;
                                                break;




                                                case 10:
                                                current_data_char = 65;
                                                break;
                                                case 11:
                                                current_data_char = 66;
                                                break;
                                                case 12:
                                                current_data_char = 67;
                                                break;
                                                case 13:
                                                current_data_char = 68;
                                                break;
                                                case 14:
                                                current_data_char = 69;
                                                break;
                                                case 15:
                                                current_data_char = 70;
                                                break;
                                                default:
                                                current_data_char = 0x00;
                                                break;
                                        }

                                                while (!(IFG2 & UCA0TXIFG));              // USCI_A0
TX buffer ready?
                                        while (  (UCA0STAT & UCBUSY ));
                                        UCA0TXBUF = current_data_char;
}

int BridgeID_Repeat()
{
        int i,IDok = 1, repeat = 0;

        //char *check_ptr;
        //check_ptr = (char *) 0x10B1;

        for (i =2; i <16; i++)
        {
                if (ID_array[i-2] != data_block[i])
                        IDok =0;
                //*check_ptr++;
        }

        if(hit_green == 1 && data_block[22] == 00)
                repeat =1;
        else if(hit_yellow == 1 && data_block[22] == 01)
            repeat = 1;
        else if(hit_orange == 1 && data_block[22] == 02)
            repeat = 1;
        else if(hit_red == 1 && data_block[22] == 03)
            repeat = 1;

        if (IDok == 1 && repeat == 0)
                return 1;
        else
                return 0;
```

```
       }


       // The CCITT CRC 16 polynomial is X + X + X + 1.
// In binary, this is the bit pattern 1 0001 0000 0010 0001, and in hex it
//  is 0x11021.
// A 17 bit register is simulated by testing the MSB before shifting
//  the data, which affords us the luxury of specifiy the polynomial as a
//  16 bit value, 0x1021.
// Due to the way in which we process the CRC, the bits of the polynomial
//  are stored in reverse order. This makes the polynomial 0x8408.
*/



/*
// note: when the crc is included in the message, the valid crc is:
//      0xF0B8, before the compliment and byte swap,
//      0x0F47, after compliment, before the byte swap,
//      0x470F, after the compliment and the byte swap.
*/


/*************************************************************************
//
// crc16() - generate a 16 bit crc
//
//
// PURPOSE
//      This routine generates the 16 bit remainder of a block of
//      data using the ccitt polynomial generator.
//
// CALLING SEQUENCE
//      crc = crc16(data, len);
//
// PARAMETERS
//      data    <-- address of start of data block
//      len     <-- length of data block
//
// RETURNED VALUE
//      crc16 value. data is calcuated using the 16 bit ccitt polynomial.
//
// NOTES
//      The CRC is preset to all 1's to detect errors involving a loss
//        of leading zero's.
//      The CRC (a 16 bit value) is generated in LSB MSB order.
//      Two ways to verify the integrity of a received message
//        or block of data:
//        1) Calculate the crc on the data, and compare it to the crc
//           calculated previously. The location of the saved crc must be
//           known.
/         2) Append the calculated crc to the end of the data. Now calculate
//           the crc of the data and its crc. If the new crc equals the
//           value in "crc_ok", the data is valid.
//
// PSEUDO CODE:
//      initialize crc (-1)
//      DO WHILE count NE zero
//        DO FOR each bit in the data byte, from LSB to MSB
//          IF (LSB of crc) EOR (LSB of data)
//            crc := (crc / 2) EOR polynomial
//          ELSE
//            crc := (crc / 2)
//          FI
//        OD
//      OD
//      1's compliment and swap bytes in crc
//      RETURN crc
//
*************************************************************************/
```

```
uint16_t crc16(char *data_p,unsigned short length)
{
       unsigned char i;
       uint32_t data;
       uint32_t crc;

       crc = 0xFFFF;

       if (length == 0)
              return (~crc);

       do {
              for (i = 0, data = (unsigned int)0xff & *data_p++;
                  i < 8;
                  i++, data >>= 1) {
                    if ((crc & 0x0001) ^ (data & 0x0001))
                           crc = (crc >> 1) ^ POLY;
                    else
                           crc >>= 1;
              }
       } while (--length);

       crc = ~crc;

       data = crc;
       crc = (crc << 8) | (data >> 8 & 0xFF);

       return (crc);
}
```

BIBLIOGRAPHY

[1]"Deficient Bridges by State and Highway System," U.S. Department of Transportation, Federal Highway Administration. December 2008.

[2]Richardson, E.V. and S.R. Davis, "Evaluating Scour at Bridges," Hydraulic Engineering Circular 18, Fourth Edition, FHWA NHI 01-001, Federal Highway Administration, U.S. Department of Transportation, Washington, D.C. 2001

[3]Lagasse, P.F., L.W. Zevenbergen, J.D. Schall, and P.E. Clopper,"Bridge Scour and Stream Instability Countermeasures - Experience, Selection, and Design Guidelines, Hydraulic Engineering Circular No. 23, Second Edition, FHWA NHI 01- 003, Federal Highway Administration, Washington, D.C. 2001

[4]Haas, Carl, Jose Weissmann, and Tom Groll. "Remote Bridge Scour Monitoring: A Prioritization and Implementation Guideline," Research Project Number 3970-1, Center for Transportation Research, University of Texas at Austin, University of Texas at San Antonio. May 1999.

[5]Kattell, John and Merv Eriksson. "Bridge Scour Evaluation: Screening, Analysis, and Countermeasures," USDA Forest Service, San Dimas, California. September 1998

[6]Schall, James D. and G.R. Price, "Portable Scour Monitoring Equipment," NCHRP Report 515, Transportation Research Board, Washington, D.C. 2004

[7]Hearn, George, "Bridge Inspection Practices," NCHRP Synthesis 375, Transportation Research Board, Washington, D.C., 2007

[8]Sohn, Pam, "Bridging the money gap: Funding 'biggest obstacle' in making repairs," Chattanooga Times Free Press, Nov 2006

[9]Stein, Stuart and Karsten Sedmera, "Risk-Based Management Guidelines for Scour at Bridges with Unknown Foundations," National Cooperative Highway Research Program Project 24-25, Transportation Research Board, October 2006

[10] "Testimony of The American Society of Civil Engineers Before the Senate Environment and Public Works Committee on Improving the Federal Bridge Program: an Assessment of S. 3338 and H.R. 3999," Senate Committee on Environment and Public Works, September 2008.

[11]Richardson, E.V. and J. Hunt, 2000, Personal Communication.

[12] "Highway Accident Report: Collapse of New York Thruway(I-90) Schoharie Creek near Amsterdam, New York, April 5, 1987," National Transportation Safety Board, 1988

[13] Richardson, Everett V ,Jorge E. Pagan-Ortiz ,and James D. SSchall "Monitoring and Plans for Action for Bridge Scour: Instruments and State Departments of Transportation Experiences,"

[14] Lefter, J, "Instrumentation for Measuring Scour at Bridge Piers and Abutments." NCHRP Project 21-3, National Transportation Board. Washington, D.C. Jan 1993

[15] "BUREAU OF DESIGN BRIDGE MANAGEMENT SYSTEM 2 (BMS2) CODING MANUAL OFFICE VERSION," Commonwealth Of Pennsylvania Department Of Transportation, Pub. 100A, July 2007.

[16] "MSP430F21X2 MIXED SIGNAL CONTROLLER DATASHEET". Texas Instruments. May 2008.

[17] "LR SERIES TRANSMITTER MODULE DATA GUIDE". Linx Technologies, Inc. January 2008.

[18] "DS RELAYS". Panasonic Electric Works.

[19] "SUBMINIATURE SLIDES SERIES AS". NKK Switches. March 2007.

[20] "S1234 PRODUCT DATA SHEET". Comus Group of Companies. 2002.

[21] USB UART IC". Future Technology Devices International, Ltd.

[22]"LM3940 1A Low Dropout Regulator for 5V to 3.3V Conversion". National Semiconductor. July 2007

[23] "Panasonic_Lithium_CR2_CR123A". Panasonic. August 2005.

[24] MSP430x2xx FAMILY USER'S GUIDE". Texas Instruments. 2008

[25]"LM3940 1A Low Dropout Regulator for 5V to 3.3V Conversion". National Semiconductor. July 2007

[26] "LR SERIES RECEIVER MODULE DATA GUIDE". Linx Technologies, Inc. January 2008.

[27] Markus Koesler, Wolfgang Lutsch. "PROGRAMMING A FLASH-BASED MSP430

USING THE JTAG INTERFACE". Texas Instruments. February 2008.

[28] American Society for Testing and Materials. http://www.astm.org/Standards/F794.htm

[29] Nicholas Cheremisinoff and Ramesh Gupta. "HANDBOOK OF FLUIDS IN MOTION". Butterworth Publishers. Stoneham, MA. 1983.

[30] Olson, Reuben. "ESSENTIALS OF ENGINEERING FLUID MECHANICS". International Textbook Company. Scranton, Pennsylvania. 1961.

[31] Lloyd A. Reed and Marla H. Stuckey. "PREDICTION OF VELOCITIES FOR A RANGE OF STREAM CONDITIONS IN PENNSYLVANIA". U.S. Geological