Design and Implementation of a Wireless CAN bus Adapter and Power Pack

Prepared for: ECE 4600

Prepared By:
Adam Hawrysh
Owen Cail
Jessie Lapsley
Eric Hambley

Faculty and/or Industry Advisor:
Dr. Douglas Buchanan
Mr. Riley Dyck

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

March 2018

© Copyright by Adam Hawrysh, Owen Cail, Jessie Lapsley and Eric Hambley, 2018
Abstract

The purpose of this project was to design and deliver an electrical package to Elmer’s Manufacturing that replaced an existing wired controller area network (CAN) bus joystick with a wireless system. This package consisted of two parts: a transmitter that connected to the joystick providing power to it, and a receiver that connected to the hydraulic controller located on the grain cart. Elmer’s Manufacturing proposed this project as a method to reduce the amount of physical connections within the cab of the tractor. This served to reduce the risk of an operator getting tangled or tripping over a wiring harness.

The joystick and transmitter were powered by a lithium ion polymer battery and various power distribution circuitry was designed to deliver power throughout the circuit. The receiver was powered by a grain cart and used a design similar to the transmitter for power distribution. The joystick transmitted data on the CAN bus. This data was read by the MCP2515 CAN controller and stored in buffers that were read by the MSP430 microcontroller. The microcontroller relayed the data to the nRF24L01+ wireless transceiver and then transmitted. When the receiving nRF24L01+ chip received the data, it converted it into a CAN message that was sent to the hydraulic controller on the grain cart.

The completed prototype was subjected to many tests. The results indicated a ~9.5-day standby battery life and a ~34-hour continuous use battery life. The components chosen were measured to have a maximum current draw of ~140mA. A useful range from the cab of the tractor to grain cart was confirmed to be ~30ft. An operating delay of approximately 29ms was achieved. Finally, an enclosure was designed using a reference provided by Elmer’s Manufacturing that successfully fit within a cup holder. These results indicate all performance metrics were met which correlates to a successfully completion of this project. However, there are suggestions for future considerations to improve the system.

In conclusion, the system was successfully designed and constructed and meets or exceeds all desired performance metrics.
Acknowledgements

We would like to thank the following people for their help with our project this year. We would not have achieved the same success without the help of everyone listed.

Dr. Doug Buchanan
Daniel Card
Riley Dyck
Sinisa Janjic
Glen Kolansky
Dr. Derek Oliver
Zoran Trajkoski
Elmer’s Manufacturing
## Contributions

<table>
<thead>
<tr>
<th></th>
<th>Adam</th>
<th>Eric</th>
<th>Owen</th>
<th>Jessie</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Research and Component Selection</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN Bus Interface</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wireless Transmission</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Batteries and Charge</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Enclosure</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hardware Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN Bus Interface</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wireless Transmission</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Batteries and Charge</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Boost/Buck Converter</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enclosure</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schematic</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCB Design</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCB Assembly and Soldering</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Software Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can Bus Module Integration</td>
<td>●</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wireless Module Integration</td>
<td>●</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmitter Microcontroller</td>
<td>○</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver Microcontroller</td>
<td>○</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Testing and Implementation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmitter Hardware</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmitter Software</td>
<td>○</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver Hardware</td>
<td>○</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver Software</td>
<td>○</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power System</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Field Implementation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation Manual</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Troubleshooting Manual</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ● Lead
- ○ Support
# Table of Contents

Abstract ................................................................................................................................. ii
Acknowledgements .............................................................................................................. iii
Contribution ........................................................................................................................ iv
Table of Contents ................................................................................................................ v
List of Figures ....................................................................................................................... vii
List of Tables ....................................................................................................................... viii
Abbreviations ...................................................................................................................... ix
Chapter 1 - Introduction .................................................................................................... 1
  1.1 Motivation .................................................................................................................. 1
  1.2 Design Metrics ......................................................................................................... 1
Chapter 2 – Hardware ....................................................................................................... 3
  2.1 Battery ...................................................................................................................... 3
  2.2 Boost Converter ....................................................................................................... 3
  2.3 Buck Converter ........................................................................................................ 4
  2.4 Voltage Regulation .................................................................................................. 5
  2.5 Charge Circuit .......................................................................................................... 6
  2.6 Switch ....................................................................................................................... 7
  2.7 Circuit Design .......................................................................................................... 9
  2.8 PCB ........................................................................................................................... 9
Chapter 3 – Hardware Implementation ........................................................................... 13
  3.1 Microcontroller ........................................................................................................ 13
  3.2 Controller Area Network Hardware ....................................................................... 14
  3.3 Wireless Transceiver ............................................................................................... 15
Chapter 4 – Software Implementation .......................................................................... 16
  4.1 Controller Area Network Protocol ......................................................................... 16
  4.2 RF Module ............................................................................................................... 17
  4.3 Battery Monitoring .................................................................................................. 17
Chapter 5 – Transmitter Software .................................................................................. 19
  5.1 Overview .................................................................................................................. 19
  5.2 Variable Initialization .............................................................................................. 20
  5.3 Transmitting Section ............................................................................................... 20
      5.3.1 Transmitter Software Explanation .................................................................. 21
5.4 State Variables........................................................................................................21
5.5 Transmitter State Select........................................................................................22
  5.5.1 Possible States..................................................................................................22
  5.5.2 Transmitter State Select Code..........................................................................25
    5.5.2.1 Battery Charging.......................................................................................25
    5.5.2.2 Battery Not Charging...............................................................................25
    5.5.2.3 Battery Check State of Charge.................................................................25
    5.5.2.4 Sleep/Low Power Mode...........................................................................25
Chapter 6 – Receiver Software....................................................................................27
Chapter 7 – Enclosure.................................................................................................30
  7.1 Transmitter Enclosure.........................................................................................30
  7.2 Receiver Enclosure.............................................................................................31
Chapter 8 – Testing....................................................................................................33
  8.1 Communication Systems Testing.......................................................................33
  8.2 Battery Life Testing............................................................................................34
  8.3 Operational Delay Testing................................................................................36
  8.4 Onsite Testing....................................................................................................37
Chapter 9 – Budget.....................................................................................................38
Chapter 10 – Future Considerations..........................................................................39
  10.1 Single Circuit Board Design............................................................................39
  10.2 Microcontroller Selection.................................................................................39
  10.3 Wireless Communication Protocol.................................................................39
  10.4 Circuit Protection..............................................................................................39
Chapter 11 – Conclusion.............................................................................................40
Appendices................................................................................................................43
  A Parts List................................................................................................................44
  B Transmitter variables............................................................................................46
  C Receiver Variables................................................................................................48
  D Transmitter Code..................................................................................................49
  E Receiver Code........................................................................................................56
  F User Manual...........................................................................................................59
  G System Testing......................................................................................................73
List of figures

2-1 Operation of the buck converter showing successful conversion of +12V to 3.3V ..........................4
2-2 The low boundary of the voltage regulator displaying proper operation.................................5
2-3 Transmitter current draw with closed switch operation..........................................................8
2-4 Transmitter current draw with open switch operation..........................................................9
2-5 Schematic for the transmitter and receiver PCBs.................................................................10
2-6 The Footprint Created for the Microcontroller.................................................................10
2-7 Physical dimension restrictions to ensure proper fit within the enclosure.............................11
2-8 PCB layout showing all traces, GND plane and +3.3V plane. ...............................................12
2-9 Complete fabricated PCB..................................................................................................12
3-1 Data Transmission and Reception Hardware Implementation...............................................13
3-2 Block Diagram of the MCP2515 CAN Controller...............................................................14
4-1 Pinout of MSP430G2553 Chip.............................................................................................16
4-2 Operating parameters of the nRF24L01+ chip [4]. (a) RF Output Power ...............................17
  Compared to Current Consumption. (b) Data Rate Compared to Receiver
  Sensitivity
5-1 Flow Chart showing how Transmitter sections work together..............................................19
5-2 Linear flowchart showing transmit section step by step.......................................................20
5-3 Flow chart of each possible battery state in the overall transmitter software.......................24
6-1 Flowchart for the receiver’s MSP430....................................................................................29
7-1 Top view of the transmitter enclosure..................................................................................30
7-2 The front and back view respectively of the transmitter enclosure.....................................31
7-3 Side view of the receiver enclosure.....................................................................................32
7-4 Top view of the receiver enclosure.....................................................................................32
8-1 Block diagram of test 1 and test 2.......................................................................................34
8-2 The state of charge (%) of the battery as a function of time................................................35
  This is for continuous use
8-3 The state of charge (%) of the battery as a function of time................................................36
  This is while the transmitter is in standby (low power) mode.
List of Tables

1-1 Performance Metrics at the outset of the project .........................................................2
11-1 Completed design metrics .........................................................................................40
Abbreviations

CAN    Controller Area Network
CRC    Cyclical Redundancy check
I²C    Inter-Integrated Circuit
IC     Integrated Circuit
IDE    Integrated Development Environment
LDO    Low DropOut regulator
LED    Light Emitting Diode
LiPo   Lithium Ion Polymer
MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor
NMOS   N-channel MOSFET
PCB    Printed Circuit Board
RF     Radio Frequency
RGB LED Red-Green-Blue Light Emitting Diode
SoC    State of Charge
SPI    Serial Peripheral Interface
UART   Universal Asynchronous Receiver-Transmitter
USB    Universal Serial Bus
USCI   Universal Serial Communication Interface
This page is left intentionally blank
Chapter 1: Introduction

1.1 Motivation

Elmer’s Manufacturing is always striving to improve their products, producing the best farm equipment for the needs of the progressive farmer. Their design for their grain cart, named the HaulMaster, boasts features such as an extremely durable and low-profile frame, as well as a joystick controlled hydraulic dual auger. The joystick, located in the cab of the tractor, is currently wired directly to a controller located on the grain cart. This controller is responsible for the hydraulic functions of the auger. Elmer’s Manufacturing is always looking for innovative improvements to their farm machinery. In this project, a wireless communication system was designed and manufactured to facilitate a wireless communication link between the cab and the grain cart.

A wireless connection will serve to resolve issues of safety for the operator. Having a wireless connection instead of a wired connection reduces the amount of physical connections between the tractor and the grain cart. This serves to reduce the risk of an operator getting tangled or tripping over the wiring harness.

1.2 Design Metrics

The purpose of this project was to design and deliver an electrical package to Elmer’s Manufacturing that turned an existing wired CAN bus joystick into a wireless joystick. This package consisted of 2 parts; a transmitter that was attached to the joystick and a receiver that was connected to the controller located on the grain cart. The transmitter circuit was equipped with a battery to power the joystick and transmitter related circuitry. It was also capable of charging the battery using a standard micro-USB port and delivering information regarding the state of charge of the battery to an external LED. The transmitter circuit was designed to fit into an enclosure roughly the size of a soda can. The receiver was powered by the existing controller on the grain cart and had very little size restrictions relative to the transmitter. The transmitter and the receiver pair could easily be switched back to the original implementation. The goals of the project were set in the form of performance metrics near the beginning of the school year. These goals are summarized in Table 1-1.
<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Proposed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Tractor Cab to Grain cart ~30 ft</td>
</tr>
<tr>
<td>Battery Life (continuous use)</td>
<td>&gt; 4 hours</td>
</tr>
<tr>
<td>Battery Life (standby)</td>
<td>&gt; 7 Days</td>
</tr>
<tr>
<td>Operating Delay</td>
<td>&lt; 500ms</td>
</tr>
<tr>
<td>Enclosure Size</td>
<td>&lt;= 355 cm³</td>
</tr>
<tr>
<td>Discharge Rate</td>
<td>&lt; 500mA</td>
</tr>
</tbody>
</table>
Chapter 2: Hardware

The power system needed to deliver power to all of the major components of the transmitter and receiver. A voltage of +6V needed to be supplied for proper joystick operation from the transmitter. A voltage of +3.3V was necessary for the correct operation for the microcontroller, RF module, and CAN module of both the transmitter and receiver. The following sections discuss how hardware was selected and used to create the power system of the transmitter and receiver circuits.

2.1 Battery

There were 3 criteria used to select an appropriate battery for this project. These criteria included the amount of charge that the battery could hold, the physical size of the battery, and the voltage the battery could output. The battery needed to contain the largest possible charge within the smallest amount of space. This was realized with a DTP605068-3P Polymer Li-ion (LiPo) Rechargeable Battery.

The LiPo chosen consisted of 3 cells, each outputting a nominal charge 2000mAH at +3.7V. The 3 cells were connected in parallel to realize a total nominal charge of 6000mAH at +3.7V. This battery’s dimensions were 60mm x 45mm x 15mm which was sufficiently small to fit into the enclosure along with the completed PCB. The nominal voltage when fully charged was ~+4.2V and ~+3.4V when the battery was nearly discharged completely. The battery was equipped with a JST connector in order to connect it to the charging circuit.

2.2 Boost Converter

The transmitter circuit required a voltage, supplied by the battery, to be nominally +3.3V. However, the joystick powered by the same battery, needed a minimum of +6V. A “boost converter” was employed to facilitate the larger voltage.

The customer confirmed the joystick’s operating voltage was +6-34V. The greater the voltage required a greater current. Therefore, the lower limit of this voltage range was chosen.

The boost converter selected used an LM2577 chip, which controlled all the power and control functions for boost conversion. The only issue with using a LM2577 based module was a relatively high quiescent current of about 10mA [1]. The current draw directly related to the battery life. To solve this issue a switch was implemented to cut the boost converter out of the circuit during the “low power” mode.
2.3 Buck Converter

In the receiver circuit, the power supplied from the grain cart had a nominal voltage of +12-14V. However, the voltage required for the microcontroller, RF module, and CAN/SPI converter was +3.3V. To realize the voltage of +3.3V a buck converter was employed.

The buck converter chosen used the LM2596 IC, which provided all the functions and control for step down conversion [2]. There were no design requirements, regarding current draw, for the receiver as it was powered from the grain cart. Figure 2-1 shows conversion from a +12V source, supplied from a bench power supply, to the +3.3V required for the microcontroller and other modules on the receiver circuit.

![Image of buck converter operation]

Figure 2-1: Operation of the buck converter showing successful conversion of +12V to +3.3V.
2.4 Voltage Regulation

During testing of the battery charge circuit in the transmitter, output voltages were discovered to be as high as +4.29V and there was no available method to limit this output to the rated value of +3.3V for the transmitter microcontroller. The voltage source providing power to the receiver, located on the grain cart, delivered a voltage range of +12-14V. The buck converter, was unable to consistently deliver a steady voltage of +3.3V for the receiver microcontroller. To supply a steady and constant +3.3V for the transmitter and the receiver, linear voltage regulators were used.

The voltage regulator selected was the MCP1825. This IC was a low dropout regulator (LDO). Dropout voltage is the minimum voltage across the regulator, required to maintain a steady and constant voltage [3]. Low dropout regulators have a small dropout voltage. The LDO data sheet gave a value of +0.35V for this regulator [4]. This meant, theoretically, +3.65V was required to ensure a regulated +3.3V was produced at the output. The LDO was tested and found to be reliable down to +3.42V as shown in Figure 2-2. The quiescent current of this LDO was 220μA [4] and as such there were few issues with the current draw. LiPo batteries risk irreversible damage if run at the lower limit (in this project’s case, +3.4V) and software was implemented to reduce the chance of hitting that limit. Therefore, reliable voltage regulation down to +3.42V was sufficient for this project.

As stated previously, the battery was rated for +3.7V, which was higher than the +3.3V required for the microcontroller, the RF module, and the CAN/SPI converter. It was decided to include a LDO as a measure to protect these devices from potential damage from higher voltages. As the battery was charged

Figure 2-2: The low boundary of the voltage regulator displaying proper operation.
and discharged the voltage output would rise and drop respectively. To deliver a constant unchanging voltage of +3.3V the LDO was used.

On the receiver side, the buck converter was used to reduce the voltage from the +12V-14V used on the grain cart to a value of +3.3V required for the microcontroller, the RF module and the CAN/SPI converter. A LDO was used here because the output from the buck converter was not a true DC source. The buck converter realized a DC value through it’s switching frequency, which created a ripple. The ripple was superimposed on the DC value and went from a maximum value to a minimum value. The DC value it delivered was the average voltage. To deliver a stable +3.3V supply, the output of the buck converter was fed to the input of a LDO. The output was set to be approximately +3.65V. This value was used to match the theoretical value needed to reliably produce +3.3V.

2.5 Charge Circuit

It was required to have the ability to charge the battery as well as convey information about its state of charge. Both of these were solved by the selection of a “Sparkfun Battery Babysitter”. This charge circuit served to alleviate 3 of the design goals; charging of the battery, battery monitoring information via I²C, and acted as the physical disconnect switch.

The charging of the circuit was controlled by the BQ24075 IC chip. This chip was an integrated lithium ion linear charger and was operated from a USB port [5]. Charging was only capable with the physical power switch of the Battery Babysitter in the ON position. Indication of charging the battery was delivered using a blue LED. When the blue LED lit up the circuit was charging the battery and it would turn off when the battery was fully charged. Charging of the battery did not happen while the physical power switch was in the OFF position. However, the blue LED would still light up. There were two small DIP switches on the Battery Babysitter that allowed for changing the charge rate. The four selectable charge rates were no charge, 100mA, 500mA and a variable rate up to 1500mA.

Battery monitoring information was delivered by the BQ2744-G1 fuel gauge IC. This chip was a microcontroller peripheral that used the patented Impedance Track ™ to provide information such as remaining battery capacity, state-of-charge (SoC) and battery voltage [6]. It used I²C, a two-wire serial interface, as its controlling interface to retrieve information from the battery. This information was then delivered to the transmitter microcontroller. A RGB LED was installed to display information regarding state of charge to the user.

The transmitter was powered through the output of the Battery Babysitter. This meant that if the physical power switch was in the OFF position the battery would be physically disconnect from the transmitter. Therefore, this was used as the physical disconnect switch.
2.6 Switch

Even when not in use, the joystick and the boost current used a significant current. Neither of these devices contained a low power mode. To conserve current such that the design metric of maximum standby battery life could be achieved, a disconnect switch was implemented with an n-channel IRLB3034PbF MOSFET.

The MOSFET was selected based largely on two criteria; the threshold voltage and the on-resistance. Threshold voltage, $V_t$, is when the “channel” is formed and the MOSFET begins to turn on. For $V < V_t$ the MOSFET is off. When the voltage between the gate and source is above the threshold voltage ($V_g > V_t$) the MOSFET effectively turns on. For a voltage $V_{gs} < V_t$ the MOSFET is off and behaves as open circuit [7]. The easiest accessible voltage was +3.3V, therefore the threshold voltage had to be lower than this value. This MOSFET had a maximum $V_t = +2.5V$ [8]. Since $V_{gs} = +3.3V > V_t$ proper operation of this switch was expected.

The MOSFET, when on, was not an ideal short circuit. There was a small on-resistance in the induced channel between the drain and source. To ensure that the power draw from this MOSFET was negligible and no unwanted heating issues were created, the smallest on-resistance was desired. This NMOS had a typical on resistance of 1.6mΩ at +4.5V [7]. This was sufficiently small to cause no excess power losses.

To implement the MOSFET as a switch, the drain was connected to both ground connections of the boost converter. Operation of the switch was controlled by the gate voltage supplied by the microcontroller. A low signal (GND) opened the switch and a high signal (+3.3V) closed the switch. This was chosen because if there was a malfunction in the circuit and it grounds, the switch will open and the joystick will be cut off from the circuit.

The switch design sufficiently reduced current draw within the transmitter. While the switch was closed (Figure 2-3) the transmitter current draw was approximately 131mA. Opening the switch (Figure 2-4) disconnected the boost converter and joystick from the circuit reducing the current draw to 49mA. A push button was used as a physical interface to toggle the signal from high to low.
Figure 2-3: Transmitter current draw with closed switch operation.
2.7 Circuit Design

A single PCB design that could be used for both the transmitter and receiver circuits was desired. This allowed for only one PCB order to be made and resulted in a reduced budget. This was possible because the transmitter and receiver shared many of the same components.

2.8 PCB

In order to realize the size constraint metric of the transmitter circuit and battery fitting into an enclosure roughly the size of a soda can, a PCB was used. All of the modules and ICs used were through hole. Therefore, a 2 layer, through hole PCB was designed.

The program used to design the PCB was KiCAD. This was chosen for two reasons; it was recommended by the Zoran (technical advisor), and it was a freeware. The program needed a schematic, footprints of the many modules and then connections needed to be defined. The output for the design consisted of gerber files and drill files. These were sent to PCBway for fabrication.

The schematic was the blueprint that the rest of the KiCAD design was based on. This dictated connections only. All modules needed to be constructed in this environment and everything that was physically connected needed to be defined in this step. In this project’s design, the transmitter and
receiver were both implemented on the same PCB. Therefore, both the connections for the transmitter and the receiver PCB needed to be defined in one single schematic. Figure 2-5 displays the schematic used for this design.

Figure 2-5: Schematic for the transmitter and receiver PCBs

Footprints were the next step that needed to be done. A footprint was the physical dimensions, along with pads and pad spacing to implement the circuit physically. Each device was measured using calipers to determine size dimensions, pin spacing and screw spacing. This information was drawn into KiCAD and would be used to create a layout. Figure 2-6 displays the microcontroller footprint used in KiCAD for this design.

Figure 2-6: The footprint created for the microcontroller.
After all the footprints were matched up to the corresponding component in the schematic, they were imported into the PCB Editor. This is where the items were physically placed and traces were run for the purposes of connection. The PCB contained modules that protruded outwards. To fit into the enclosure these modules had to be placed in specific spaces. This meant that the design of the PCB had to happen in parallel with the design of the transmitter enclosure. Figure 2-7 shows the restrictions of where modules needed to be in order to meet the enclosure design metric.

![Figure 2-7: Physical dimension restrictions to ensure proper fit within the enclosure.](image)

With all the components placed, the next task was to define the physical connections. Traces were used for this purpose. The width of the trace determined how much current could be safely run through it. It was desirable to make the traces as wide as possible. With that information all the traces that connected +Vout were 30mils. This value was recommended by the technical advisor. The PCB layout showing all trace connections, GND plane and +3.3V plane is displayed in Figure 2-8.
Figure 2-8: PCB layout showing all traces, GND plane and +3.3V plane.

All the traces were made with the exception of all pads going to +3.3V and GND. To accomplish this, the top plane was set to +3.3V and the bottom plane was set to GND. To minimize interference the space underneath the RF module contained neither the +3.3V layer nor the GND plane. All of the remaining connections were connected to the plane it corresponded to. Figure 2-9 shows the completed layout for the PCB design.

Figure 2-9: Complete fabricated PCB.
Chapter 3: Hardware Implementation

The hardware implementation of the data transmission and reception consisted of 3 sections. Namely, the TI MSP430 microcontrollers, the CAN/SPI modules and the RF modules. Figure 3-1 below shows the complete signal hardware implementation. The following sections describe the implementation of these components.

![Data Transmission and Reception Hardware Implementation](image)

**Figure 3-1: Data Transmission and Reception Hardware Implementation**

3.1 Microcontroller

The TI MSP430 was chosen as the microcontroller for both the transmitter and receiver due to its low cost, low power consumption, small size, and low power modes. The MSP430 operated at 16MHz using an internal digital controlled oscillator, which reduced the number of external components that other controllers might require. This clock speed was fast enough to complete all transmitter and receiver operations while not missing any messages from the joystick. The MSP430 also offered all of the necessary communication protocols required, namely, the FC protocol for communication with the battery charging module, the SPI protocol for communication with the CAN/SPI module, and a UART for external debugging during testing. The Integrated Development Environment (IDE) for the MSP430 is compatible with libraries for the Arduino platform, which allowed for the use of existing libraries, rather than creating new ones [9].
3.2 Controller Area Network Hardware

The MSP430 chip chosen does not include an internal CAN controller, so an external controller and transceiver were required. The MikroElectronika CAN/SPI click breakout board was chosen over a custom design due to its ease of integration. The breakout board allowed testing to occur during code development rather than waiting for the custom PCB design to be completed.

The breakout board included the TI SN65HVD230 CAN transceiver chip, and the Microchip MCP2515. The SN65HVD230 was a CAN transceiver that converted the CAN high and the CAN low differential signals into a single-ended signal that was compatible with the CAN controller and vice versa. The MCP2515 was a CAN controller that communicated between the CAN transceiver and the microcontroller. It stored the CAN messages in buffers that were accessible by the microcontroller using the SPI communication protocol. Both chips were designed to operate at +3.3V and also offered a low power standby mode. However, due to the design of the breakout board, only the low power mode of the MCP2515 could be used since the required pins on the SN65HVD230 were not accessible to the MSP430. The MCP2515 offered an interrupt pin which allowed the microcontroller to check if there were messages waiting to be read without requiring SPI communication.

![Figure 3-2: Block Diagram of the MCP2515 CAN Controller](image)
As shown in the figure above, the MCP2515 included receive and transmit buffers. There were 3 separate buffers for the transmit and receive modules. When a message was received from the CAN network, it was immediately stored in one of the receive buffers. This allowed the MSP430 to continue its current task without the message being lost. Once the MSP430 had completed its task, it could read the CAN message from the MCP2515. When the MSP430 was attempting to send a CAN message, the data was written to one of the transmit buffers. If the MCP2515 was not able to immediately send the message due to bus activity, the MCP2515 would automatically attempt to retransmit the message that was stored in the buffers without any required action from the MSP430.

3.3 Wireless Transceiver

The nRF24L01+ single chip transceiver was chosen for the wireless communication required for the project. Both the Sparkfun WRL-00691 and the Sparkfun WRL-00705 breakout boards were chosen due to their different antenna types. One of the requirements for the project was that the receiver package used an external antenna, so the WRL-00705 was chosen. Due to the space constraints of the transmitter package, the WRL-00691 was chosen since it included a surface mount ceramic antenna right on the breakout board. Since these are the only differences between the two breakout boards, they will both be referred to as the wireless breakout board for the remainder of the document.

The wireless breakout boards used a proprietary communication protocol in the 2.4 GHz ISM band. The operating frequency could be chosen from up to 126 different channels between 2.4 GHz and 2.525 GHz. This allowed for selection of a channel that minimized the interference from other sources. The wireless transmission data rate could also be set at 250kbps, 1Mbps, and 2Mbps. The lower data rate used decreased the current draw and increased the transmission range of the boards.

The wireless transmissions also included a 5-bit address for each transmitter/receiver pair. This could be used to control multiple grain carts with a single joystick, or allow up to 32 individual pairs on the same channel without interference.

The Enhanced ShockBurst™ technology of the boards allowed 16-bit CRC for error checking and automatic re-transmission if a packet was not received or if it contained an error. Cyclical Redundancy Check (CRC) was a code used for error detection in digital data transmissions. The error checking and re-transmission system was used to simplify the project, which reduced the chances that a message was missed.
Chapter 4: Software Implementation

All software was created using the Energia IDE. This IDE was compatible with many libraries created for the Arduino platform. This simplified the software implementation due to the large number of open-source libraries available online. The following sections describe how the software was implemented for the various modules.

4.1 Controller Area Network Protocol

The CAN/SPI breakout boards were implemented using an open source library for the Arduino platform [11]. It was decided that the SPI from the hardware Universal Serial Communication Interface (USCI) B would be used due to the higher importance of the SPI data over the battery information which used I2C. A hardware implementation for SPI allowed faster transmissions, and reduced the workload of the MSP430. The second USCI interface was chosen so that it was possible to use the first USCI interface for debugging.

![Pinout of MSP430G2553 Chip](image)

Figure 4-1: Pinout of MSP430G2553 Chip [9]

The MCP2515 was designed to be used for many different CAN bit rates, so it was required to calculate the timing parameters to be set on the chip. The Kvaser CAN Bus Bit Timing Calculator was used for these calculations [12]. The CAN/SPI Breakout board included a 10 MHz clock crystal, and the J1939 protocol that is used by the Elmer’s Manufacturing system used a 250-kbps bit rate. The calculator returned the 3 values for the timing configuration registers.
4.2 RF Module

An open-source library was used for the wireless breakout boards. The library was created for Energia, no additional modifications were required for implementation [13]. The wireless chip also used the SPI protocol, which allowed it be integrated to the USCI B hardware interface on the MSP430 for the same reasons as the MCP2515 above.

It was chosen to operate the chip at 2.40 GHz, with a bit rate of 250kbps. The transmitter transmission power level was set to -12 dBm. This bit rate was chosen due to the low data transfer rate of the project. This allowed an increased operating distance without an increased RF output power which would have increased the power consumption in the transmitter package. At 250 kbps, the receiver sensitivity was -94dBm.

<table>
<thead>
<tr>
<th>SPI RF-SETUP (RF_PWR)</th>
<th>RF output power</th>
<th>DC current consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0dBm</td>
<td>11.3mA</td>
</tr>
<tr>
<td>10</td>
<td>-6dBm</td>
<td>9.0mA</td>
</tr>
<tr>
<td>01</td>
<td>-12dBm</td>
<td>7.5mA</td>
</tr>
<tr>
<td>00</td>
<td>-18dBm</td>
<td>7.0mA</td>
</tr>
</tbody>
</table>

(a)

(b)

Figure 4-2: Operating parameters of the nRF24L01+ chip [14]. (a) RF Output Power Compared to Current Consumption. (b) Data Rate Compared to Receiver Sensitivity

The wireless chip on the breakout board also included an integrated CRC and auto-acknowledgement features. A 16-bit CRC greatly reduced the chance of an error being introduced during transmission. In the event of an error, the automatic acknowledgement and retransmission was handled by the chip which would attempt, up to 15 times, to retransmit the message every 500μs. Packets that were not acknowledged were handled in the same way.

4.3 Battery Monitoring

The BQ27441-G1 IC on the battery monitor module used the I2C protocol to send battery data to the microcontroller. An open source Arduino library was used to implement software-based I2C in the
MSP430 [15]. The BQ27441-G1 offered many measurements and status flags relating to battery status, however, only the `AverageCurrent` (command 0x10) and `StateOfCharge` (Command 0x1C) measurements were used [6]. Both variables required a 2-byte read from the IC. The `AverageCurrent` command returned a 16-bit signed value of the current entering the battery. The returned value was negative when the battery was discharging and positive when charging. The `StateOfCharge` command returned a 16-bit unsigned value between and 0 and 100 which represented the remaining percent of charge in the battery.
Chapter 5: Transmitter Software

5.1 Overview

The transmitter software was responsible for transmitting an SPI message wirelessly to the receiver. The software could be divided into four parts; state variable initialization, a transmit section, a state select and a state variable incrementation, as shown in Figure 5-1.

![Flow Chart showing how Transmitter sections work together.](image)

When the CAN interrupt pin (CAN0_INT) was detected as low, the transmitter software entered the first part of the code in which a message to the receiver was sent. This pin was used to determine when a signal must be sent via the CAN/SPI module. If the CAN0_INT pin was detected as high, the transmit section of the code was skipped over and the following two sections (as shown in Fig 5-1) executed.
5.2 Variable Initialization

The first part of the transmitter software was responsible for state variable initialization. Each of these variables were re-initialized when the transmitter package woke up from sleep mode or the software was operating for its first time after a battery disconnect. All the variables utilized in this section are explained in Appendix B.

5.3 Transmitting Section

In the flowchart below (Fig 5-2) a step by step process of how transmission occurred is illustrated.

![Flowchart]

Figure 5-2: Linear flowchart showing transmit section step by step.
5.3.2 Transmitter Software Explanation

In the first statement of the code, the state of the CAN module interrupt pin was read. A low value indicated that the software should read the output from the joystick.

The functions readMsgBuf and elmersCANToLong were utilized for reading the most recent joystick message from the CAN module, storing the variable in an array, and then translating this array of variables into a 64-bit variable. This data was then stored in the variable rxBuf which was an 8-byte character array, with each entry containing an 8-bit number.

This data was then sent to the RF module for wireless transmission. In the functions ‘Radio.println’ and ‘radio.flush’ a message to the RF module buffer was added and then transmitted wirelessly to the receiver. These function calls were contained in the RF module library.

Initial testing of the joystick revealed that output was always varying. A test was performed to clarify valid user input by determining the maximum and minimum outputs while a user was not pressing any joystick buttons. The function ‘noJoystickInput’ specified thresholds that occurred when a user was not pressing any joystick buttons. The result of this function incremented the variable timeout when the threshold was not passed. After timeout reached the pre-determined value, the transmitter package entered sleep mode.

In the following “if/else block”, the pushbutton switch was polled and the variables btnPush and btnSoc were incremented if the pushbutton output was low. Both of these variables were reset to zero when the pushbutton was released. Once the variable btnSoc surpassed the threshold stored in the variable readSOC (roughly 1 second of pushing the pushbutton down), the variable soc_flag was set high which allowed the state select section of the software to choose the correct operational state.

5.4 State Variables

The state machine controlled and handled battery monitoring information and state incremented variables to determine the required used state variables to determine which state it would operate in. The state variables used were: btnPush, btnSoc, timeout, current, soc, zero_counter, and soc_counter.

The three variables, btnPush, btnSoc, and timeout were incremented in the transmitting section of the software. These variables had specific times associated with the state functions and used the known 50ms delay of the transmit section. This delay was used to define time limits for the state machine. The time limits for the variables were as follows:

1. One second was the amount of time the pushbutton is held down (stored in btnSOC)
2. Five seconds was the amount of time to go to sleep after the pushbutton was pressed and held (stored in btnPush).

3. 300 seconds was the amount of joystick inactivity required to go to sleep (stored in timeout).

The variable zero_counter stored the count of consecutive cycles that the variable, current, was read as zero. Once current had been read as zero for 50 cycles, the transmitter package entered sleep mode. The purpose of this variable was to prevent data transmission while the battery was disconnected. The soc_counter was used to count how many times consecutive cycles that the state of charge was below the set threshold of 10% which was a chosen software battery limit. Once soc had been zero for 10 cycles, the state of charge was confirmed to be low and stable, and the transmitter package entered a sleep mode. This was used to ensure the battery did not transmit when the battery charge was low and prevented battery damage.

The function ‘wire.fuelGaugeRead()’ used the I2C library to interface with the battery babysitter module to read both state of charge (soc) and current (current) data from the battery once per cycle. The soc variable stored an integer from 0 to 100 which represented battery charge percent. This variable was used by the state select section of the code to display battery charge data via an LED. The LED was flashed one of three colours once btnSOC was held down for one second.

The current variable was the measure of current entering the battery. When the battery was discharging the variable, current was negative, and when charging, current was positive. The variable current was used to put the state machine in two of five possible states.

5.5 Transmitter State Select

5.5.1 Possible States

The final section of the software handled the finite state machine. In this section of the code, the state machine determined the required operational state by reading the variables from the previous sections of the code as shown in Figure 5-3. The battery states in the finite machine were battery charging, battery not charging, SoC check, sleep mode, and battery disconnected.

During the battery not charging state, the green LED flashed on then off for 200ms and 2000ms respectively. The green flashing LED was used to indicate to the user that the transmitter was not in sleep mode, actively transmitting and not charging. When the USB-charger was connected to the battery babysitter module, the transmitter package entered a charging state. In this state, the green LED faded in and out. The LED fading indicated to the user than the transmitter was not in sleep mode, actively transmitting and charging. In the SoC check state, an RGB LED was held on for three seconds which
displayed the battery state of charge information. A green LED represented battery life above 66%, red below 33%, and yellow in between the former values. This indicated the battery charge percentage to the user. After the LED displayed state of charge, the transmitter would return to its previous state.

In the sleep mode state, the power was cut off to the joystick and the MSP, CAN and RF modules were placed in a low power mode. This was to preserve the battery life without it being fully disconnected. The battery disconnect state was automatically entered once the battery is disconnected.
Figure 5-3: Flow chart of each possible battery state in the overall transmitter software.
5.5.2 Transmitter State Select Code

5.5.2.1 Battery Charging

If the variable current was greater than zero, the transmitter was in a charging state. In this state, the red LED was turned off and the green LED faded in and out a number of time/cycles according to the variable fadeAmount every cycle. In this state, the transmitter did not go to sleep due to joystick inactivity. To ensure this, timeout was set to zero during every loop while in this state.

5.5.2.2 Battery Not Charging

In this software state, the green LED flashed on for 200ms, and off for 2000ms. In order to implement this timing, the ‘millis()’ function was used. This function returned the number of milliseconds since the MSP had started running. By comparing a previous millis() result, (saved in previousMillis), to the current millis() result (saved in currentMillis), the amount of time passed could be calculated. The difference between currentMillis and previousMillis was compared to the variable interval. If this difference was greater than interval, the LED was toggled on or off. To have uneven on/off periods for the LED, the interval variable had to be changed each time the LED flashed on or off. When the LED was turned on, interval was redefined as interval_HI. When the LED was turned off, interval was redefined as interval_LO.

5.5.2.3 Battery Check State of Charge

In the next state, the battery state of charge was checked. As stated previously, btnSOC was a variable that was used to poll the pushbutton switch and set the soc_flag high when the threshold (readSOC) was passed. To enter this state, the soc_flag variable must be high, and current must be less than zero (battery was not charging). This state first sets interval to interval_hold, which was compared to currentMillis and previousMillis. If the difference was greater than interval, the LED is toggled off or on. This argument is true twice in each state of charge check, one for on and one for off.

5.5.2.4 Sleep/Low Power Mode

If at any time one of the while loop arguments is false, (idle for five minutes, push and hold pushbutton for 5 seconds, battery disconnected or low state of charge) the modules began to enter sleep mode. All LEDs were shut off, and the MSP AUX_PWR pin was set low which operated a switch to
disable power to the joystick. The next two functions were responsible for putting both the RF and CAN modules into sleep mode. The command `suspend()` was used to put the microcontroller to sleep. An interrupt was then set to the pushbutton switch to wake up the microcontroller and other modules once the push button was pressed. The microcontroller was then woken up with the `wakeup()` command. The RF/CAN modules were then woken up and AUX_PWR was set to high to supply power to the joystick.
Chapter 6: Receiver Software

The goal of the receiver program was to read messages sent from the transmitter via the RF module, and relay these messages to the grain cart CAN bus at regular intervals. The receiver program continuously sent CAN messages, even when there were no new messages from the transmitter. If there were no messages from the transmitter for an extended period of time, the receiver entered a safety mode that told the grain cart controller there are no joystick inputs. The receiver used a polling approach to accomplish this. There was an infinite loop which checks if it had time to send a message to the grain cart, and if there was a message from the transmitter.

In order to decide when to send messages to the grain cart, a built-in function called `millis()` and a variable called `timestamp` were used. The `millis()` function returned the number of milliseconds since the MSP430 started running. Every time a CAN message was sent to the controller, the result of `millis()` was stored in `timeStamp`. To check when to send a message to the grain cart, the current value of `millis()` was compared to `timeStamp`. When the difference was greater than 50, then 50ms had passed and a message should be sent to the grain cart. The duration of 50ms was chosen because that was the time between messages sent from the joystick.

In the event that there was a disconnect between the transmitter and the receiver, the receiver entered a safety mode that continually sent a “default” message to the grain cart. The “default” message was stored in `sleepData[]` and corresponded to no buttons being pressed and no movement on the joystick. The transmitter entered sleep mode after a number of seconds defined by `SAFETY_MODE` had passed. Since the transmitting branch of the code ran at regular 50ms intervals, the number of seconds passed could be calculated by keeping track of the number of missed transmissions in the variable `missedTransmissions`. This variable was incremented every time a CAN message was sent, and was set to 0 every time a message was received from the transmitter. As such, the time passed in seconds was `missedTransmissions*20`.

The other important function of the receiver was to receive messages from the transmitter. All of the wireless transmission was taken care of by the nRF24L01+ chip discussed in chapters 3 and 4. The nRF24L01+ chip sent the message in a character (char) array which was then converted into a 64-bit variable. The joystick command information was then extracted from this 64-bit variable and stored as a byte array which was then ready to be sent to the CAN bus. A flowchart which summarizes the receiver
software is shown in Figure 6-1, and a summary of the important variables in the program can be found in Appendix C.

The status of the receiver was indicated by an RGB LED. The LED was turned on during the process of sending a CAN message, so it was lit very briefly once every 50ms. This produced a steady, quick blinking effect. The blinking color was red when the receiver was in safety mode, and it was green otherwise. If there was no blinking light, then the receiver was not sending messages to the CAN bus.
Figure 6-1: Flowchart for the receiver's MSP430
Chapter 7: Enclosure

7.1 Transmitter Enclosure

One of the requirements of the project was to ensure the transmitter enclosure was roughly the size of a soda can. The outer dimensions were taken from a reference model provided by Elmer’s Manufacturing. This ensured that the enclosure would meet the size requirements.

Many modifications were made to this reference model to fit the electrical components inside. The chamfer on the opening of the lower half was removed to allow for a larger circuit board to be inserted. Two slots were added along the sides of the enclosure that hold the PCB in place once it was inserted. Other cut-outs were added for the LED indicator, push button, Micro-USB charging port, and ON/OFF switch.

Figure 7-1: Top view of the transmitter enclosure.
Figure 7-2: The front and back view respectively of the transmitter enclosure.

7.2 Receiver Enclosure

There were less restrictions on the size of the receiver enclosure, relative to the transmitter enclosure. So, a simple box-shaped enclosure was designed from scratch. The enclosure has 4 screw posts at the bottom which lined up with the screw holes on the PCB in order to fasten it in place. The enclosure also featured 6 mounting bracket holes. The holes were elongated which allowed for greater flexibility of where the receiver was placed.

A hole was added to allow the antenna, power wires, and CAN bus wires to leave the enclosure. The receiver was mounted on the grain cart which would be exposed to dust and dirt. As such, the hole was designed to be as small as possible. The hole had a pointed top so that it was easier to print on the fused filament fabrication 3D printer being used to fabricate the enclosure. The enclosure also featured a lid that fit into place, but could also be secured with 2 screws. All screw holes on the enclosure were designed to fit a standard #6 screw.
Figure 7-3: Side view of the receiver enclosure.

Figure 7-4: Top view of the receiver enclosure.
Chapter 8: Testing

The Elmer’s Manufacturing site was located in Altona, MB - 100km south of Winnipeg. Due to this separation of the design work space and the customer’s product, testing was divided into two sections: bench testing where the project was tested under lab conditions, and onsite testing, at the Elmer’s Manufacturing site where tests could be performed under real world conditions.

8.1 Communication Systems Testing

Specific tests were set up to analyze the reliability of the CAN and wireless communications systems over long periods of use. The transmitter generated random messages which were sent to a computer via the serial monitor. The receiver was also hooked up to a computer via serial monitor with every received message printed. After the test was completed, the list of printed messages sent by the transmitter and received by the receiver were compared for any differences. Any differences or missed messages were counted as errors.

In test 1 shown in Figure 8-1, the two CAN modules were wired directly together for testing. This test was run for over 7 hours, totalling 531,318 messages sent. After comparing the sent and received messages, it was found that there were no errors in communication.

In test 2 shown in Figure 8-1, the wireless communication was to be tested. For this test, the transmitter and receiver were placed in adjacent rooms to more closely simulate a true working environment. The test was run for over 6 hours, totalling 448,712 messages sent. After comparing to the received messages, it was found that there was 1 dropped message, and no further errors.
Given that our communications systems experienced an error only once in a 6-7-hour time period suggests that the systems are very reliable. Since the time between messages was only 50ms, a single missed message would not have any noticeable impact on operation of the machine.

8.2 Battery Life Testing

Battery life was emphasized as an important design attribute by Elmer’s Manufacturing. One goal of this project was to make the transmitter last as long as possible on a single battery charge. Thus, it was important to test battery life of the completed design.

The first test was designed to check the battery life during continuous use (also known as full power operation). For this test, the sleep mode on the transmitter was disabled, and it was allowed to run for an extended period of time. The data collected can be seen in Figure 8-2. It was discovered that the SoC value returned from the battery monitor changes linearly with time. As such, the data can be extrapolated to estimate how long a fully charged battery would last in full power operation. The battery discharged by 79% in 31 hours, which was a rate of 2.5% decreased per hour. At this rate, the battery would last over 34 hours of continuous use as extrapolated in Figure 8-2. This was well within the design metrics.
The system was then tested for duration while in standby mode. The test ran for 56.1 hours during which time the battery state of charge went from 68% to 46%. The discharge was also found to be linear in time. In 56.1 hours, the state of charge decreased by 22% which equaled a rate of 0.39% loss per hour. At this rate, it would take 230 hours to drain a fully charged battery to the software battery limit of 10%, which is ~9.5 days. A graph showing the measured and extrapolated data can be seen in Figure 8-3.
After the battery life tests were complete, it was decided that a software battery limit of 10% would be put in place in order to preserve the life of the LiPo battery. This meant the transmitter would enter sleep mode immediately while the state of charge of the battery was less than 10%. Preventing the battery from fully discharging increased the number of discharge cycles the battery can produce in its lifetime.

8.3 Operational Delay Testing

Operational delay was tested using a logic analyzer. An operation delay was defined as the time between when a message was being sent from the joystick to an electrical signal being sent to the actuator on the grain cart. The logic analyzer was connected to the input of the transmitter and the output of the grain cart controller so that the full system delay could be measured. The delay was measured to be 29ms, which was much less than a typical human reaction time [16]. As such it would not add any noticeable delay to operation of the grain cart auger.
8.4 Onsite Testing

There were two site visits to Elmer’s Manufacturing: October 5th, 2017 and December 11th, 2017. During these visits the wireless transmitter and receiver successfully replaced the current wired CAN bus. All commands from the joystick were verified.

The operating range of the project was also tested. The transmitter and joystick were carried by one team member who walked away from the grain cart while attempting to use the joystick to control the auger. A rough measurement was taken by counting the number of steps taken away from the grain cart. This rough measurement was sufficient because the project continued to work over a distance of 100 steps while the required distance was only 30 feet.
Chapter 9: Budget

The total cost of the project was approximately $1469.09 due to the unknown exact price of the joystick and falcon controller. A total of $386.71 was paid for by the University of Manitoba, the remainder was supplied by Elmer’s Manufacturing. The detailed budget is shown in Appendix A. The budget in the appendix includes extra parts that were not used in the final project, but were purchased to replace broken parts or parts that could be used for alternate solutions. To test the project, a harness had to be created to interface between the Falcon controller and a PC, which would not need to be reconstructed for future builds. Because of this, the total cost of the system would be reduced if it was reproduced. In addition, parts that were obtained at no cost were excluded from the budget (e.g. resistors, enclosures, wires). When compared to the current implementation at Elmer’s Manufacturing (which already includes the joystick and Falcon controller), the approximate cost to reproduce this project is $301.58.
Chapter 10: Future Considerations

10.1 Single Circuit Board Design

To reduce the cost of the project, the system could be redesigned to remove the use of breakout boards and have all the chips installed on a single board. However, this would greatly increase the complexity of the board and would require a new microcontroller with more pins. With access to all the pins of all the integrated circuit chips, additional low power modes could be added that would increase the standby battery life.

10.2 Microcontroller Selection

The use of a different microcontroller could remove some of the external ICs that are currently required. Microcontrollers that include CAN controllers or RF transceivers could be used instead of the MCP2515 CAN controller or the NRF24L01+ module respectively.

10.3 Wireless Communication Protocol

Rather than a transmitter and receiver pair being required, it may be possible to have the transmitter communicate directly to the Falcon controller. This would greatly reduce the cost of the system.

10.4 Circuit Protection

Elmer’s Manufacturing is still developing features on their grain cart. This results in connections changing because of this there is a possibility the cable supplying power could be wired in reverse. This would cause damage to the current circuit design. Protection circuitry placed before the buck converter would provide protection for accidental reverse voltage and could be added to subsequent prototypes.
Chapter 11: Conclusion

In this project, all design metrics were either met or exceeded, as shown in Table 11-1. Transmission of messages from the transmitter to the receiver was reliable and responsive, the battery lasted as long as possible due to the transmitter’s automatic sleep mode, and all of the circuitry was packed nicely into an enclosure that would fit in a typical cup holder. In addition, this project could seamlessly be integrated into the HaulMaster’s current setup. The end product was a clean and tidy solution that removed the extra wires running from the grain cart to the tractor cab.

Table 11-1: Completed design metrics.

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Proposed Value</th>
<th>Tested Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Tractor Cab to Grain cart: ~30ft</td>
<td>Tractor Cab to Grain Cart: ~30ft</td>
</tr>
<tr>
<td>Battery Life (continuous use)</td>
<td>&gt; 4 Hours</td>
<td>&gt; 34 Hours</td>
</tr>
<tr>
<td>Battery Life (standby mode)</td>
<td>&gt; 7 Days</td>
<td>&gt; 9.5 Days</td>
</tr>
<tr>
<td>Operating Delay</td>
<td>&lt; 500 ms</td>
<td>29 ms</td>
</tr>
<tr>
<td>Enclosure Size</td>
<td>&lt;= 355 cm³</td>
<td>355 cm³</td>
</tr>
<tr>
<td>Discharge Rate</td>
<td>&lt; 500mA</td>
<td>140mA (max)</td>
</tr>
</tbody>
</table>
References


Appendices
## Appendix A

### Parts List

Table A-1: Items Supplied by the University of Manitoba

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Price</th>
<th>Quantity</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>$3.83</td>
<td>2</td>
<td>$7.66</td>
</tr>
<tr>
<td>Wireless Module</td>
<td>$24.09</td>
<td>1</td>
<td>$24.09</td>
</tr>
<tr>
<td>Wireless Module w/ External Antenna</td>
<td>$26.78</td>
<td>1</td>
<td>$26.78</td>
</tr>
<tr>
<td>CAN Bus Transceiver Module</td>
<td>$28.95</td>
<td>2</td>
<td>$57.90</td>
</tr>
<tr>
<td>Lithium-Ion Polymer (LiPo) Battery</td>
<td>$34.99</td>
<td>1</td>
<td>$34.99</td>
</tr>
<tr>
<td>Battery Babysitter LIPO Battery Charger</td>
<td>$26.95</td>
<td>1</td>
<td>$26.95</td>
</tr>
<tr>
<td>20W Adjustable DC-DC Buck Converter</td>
<td>$6.62</td>
<td>1</td>
<td>$6.62</td>
</tr>
<tr>
<td>DC-DC Boost Converter</td>
<td>$10.14</td>
<td>1</td>
<td>$10.14</td>
</tr>
<tr>
<td>Linear Regulator</td>
<td>$1.23</td>
<td>5</td>
<td>$6.15</td>
</tr>
<tr>
<td>20 Pin Connector Socket</td>
<td>$0.39</td>
<td>2</td>
<td>$0.78</td>
</tr>
<tr>
<td>n-Channel MOSFET</td>
<td>$4.04</td>
<td>3</td>
<td>$12.12</td>
</tr>
<tr>
<td>Push button Switch</td>
<td>$0.24</td>
<td>2</td>
<td>$0.48</td>
</tr>
<tr>
<td>2.45GHz RP-SMA Antenna</td>
<td>$6.72</td>
<td>1</td>
<td>$6.72</td>
</tr>
<tr>
<td>2-Position Screw Terminal</td>
<td>$2.01</td>
<td>2</td>
<td>$4.02</td>
</tr>
<tr>
<td>3-Position Screw Terminal</td>
<td>$3.03</td>
<td>2</td>
<td>$6.06</td>
</tr>
<tr>
<td>5-Position Screw Terminal</td>
<td>$2.32</td>
<td>2</td>
<td>$4.64</td>
</tr>
<tr>
<td>RGB LED</td>
<td>$1.28</td>
<td>2</td>
<td>$2.56</td>
</tr>
<tr>
<td>Perfboard</td>
<td>$16.46</td>
<td>2</td>
<td>$32.92</td>
</tr>
<tr>
<td>PCB (Price Approximated)</td>
<td>$40.00</td>
<td>2</td>
<td>$80.00</td>
</tr>
<tr>
<td><strong>Taxes, Shipping</strong></td>
<td><strong>$35.18</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>$386.71</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Unit Price</td>
<td>Quantity</td>
<td>Total Price</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Joystick</td>
<td>$500.00*</td>
<td>1</td>
<td>$500.00</td>
</tr>
<tr>
<td>Falcon Controller</td>
<td>$500.00*</td>
<td>1</td>
<td>$500.00</td>
</tr>
<tr>
<td>6 Pin Plug Deutsche Connector</td>
<td>$2.81</td>
<td>1</td>
<td>$2.81</td>
</tr>
<tr>
<td>6 Pin Connector Wedge Lock</td>
<td>$0.59</td>
<td>1</td>
<td>$0.59</td>
</tr>
<tr>
<td>Deutsche Socket Terminals (Loose)</td>
<td>$0.44</td>
<td>10</td>
<td>$4.38</td>
</tr>
<tr>
<td>12 Pin Plug Deutsche Connector (Key A)</td>
<td>$6.82</td>
<td>1</td>
<td>$6.82</td>
</tr>
<tr>
<td>12 Pin Plug Deutsche Connector (Key B)</td>
<td>$6.78</td>
<td>1</td>
<td>$6.78</td>
</tr>
<tr>
<td>12 Pin Plug Deutsche Connector (Key C)</td>
<td>$8.13</td>
<td>1</td>
<td>$8.13</td>
</tr>
<tr>
<td>12 Pin Plug Deutsche Connector (Key D)</td>
<td>$8.33</td>
<td>1</td>
<td>$8.33</td>
</tr>
<tr>
<td>12 Pin Connector Wedge Lock</td>
<td>$0.56</td>
<td>4</td>
<td>$2.25</td>
</tr>
<tr>
<td>Deutsche Socket Terminals (Loose)</td>
<td>$0.59</td>
<td>40</td>
<td>$23.76</td>
</tr>
<tr>
<td>2 Pin Receptacle Deutsche Connector</td>
<td>$1.29</td>
<td>1</td>
<td>$1.29</td>
</tr>
<tr>
<td>2 Pin Connector Wedge Lock</td>
<td>$0.24</td>
<td>1</td>
<td>$0.24</td>
</tr>
<tr>
<td>3 Pin Receptacle Deutsche Connector</td>
<td>$1.73</td>
<td>1</td>
<td>$1.73</td>
</tr>
<tr>
<td>3 Pin Connector Wedge Lock</td>
<td>$0.42</td>
<td>1</td>
<td>$0.42</td>
</tr>
<tr>
<td>Deutsche Pin Terminals (Loose)</td>
<td>$0.39</td>
<td>8</td>
<td>$3.12</td>
</tr>
<tr>
<td>USB Type B Receptacle</td>
<td>$1.63</td>
<td>1</td>
<td>$1.63</td>
</tr>
<tr>
<td>Taxes, Shipping*</td>
<td></td>
<td></td>
<td>$10.12</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$1082.38</strong></td>
</tr>
</tbody>
</table>

*Approximate cost
## Appendix B

### Transmitter Variables

Table B-1: list of the variables used in the transmitter program.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Incremented to count subsequent transmit loops of non-user output from the joystick.</td>
</tr>
<tr>
<td>soc</td>
<td>Variable that stores a state of charge reading from 0-100 from the battery babysitter module.</td>
</tr>
<tr>
<td>btnPush</td>
<td>Keeps track of how long pushbutton is held down for sleep mode purposes.</td>
</tr>
<tr>
<td>btnSOC</td>
<td>Keeps track of how long pushbutton is held down to set the soc_flag.</td>
</tr>
<tr>
<td>soc_flag</td>
<td>Used to alert to software to enter the state of charge display state.</td>
</tr>
<tr>
<td>zero_counter</td>
<td>Stores number of subsequent loops of zero current.</td>
</tr>
<tr>
<td>soc_counter</td>
<td>Counts the number of subsequent loops of the soc variable being less than the software battery limit of 10%.</td>
</tr>
<tr>
<td>interval</td>
<td>Time in milliseconds that LED should be turned on/off for.</td>
</tr>
<tr>
<td>timeout_mins</td>
<td>Timeout variable, enters sleep after set minute value (5 seconds).</td>
</tr>
<tr>
<td>pb_seconds</td>
<td>Timeout variable, enters sleep after pushbutton held for set number of seconds (5 seconds).</td>
</tr>
<tr>
<td>soc_seconds</td>
<td>Check state of charge after set time in seconds (1 second).</td>
</tr>
<tr>
<td>brightness</td>
<td>Declare how bright the fading LED starts at while charging.</td>
</tr>
<tr>
<td>fadeAmount</td>
<td>Set how many points to fade the LED by per cycle while charging.</td>
</tr>
<tr>
<td>current;</td>
<td>Stores value of current entering the battery.</td>
</tr>
<tr>
<td>soc;</td>
<td>Stores value of charge of battery from 0-100%.</td>
</tr>
<tr>
<td>int rxId;</td>
<td>Received eID from CAN module (not used).</td>
</tr>
<tr>
<td>len</td>
<td>Received data length from CAN module (not used).</td>
</tr>
<tr>
<td>rxBuf[8];</td>
<td>Received data from CAN module.</td>
</tr>
<tr>
<td>spout_x</td>
<td>Variable to store current value of joystick X.</td>
</tr>
<tr>
<td>spout_z</td>
<td>Variable to store current value of joystick Z.</td>
</tr>
<tr>
<td>idleTimeout</td>
<td>Number of cycles required for timeout based on timeout_mins.</td>
</tr>
<tr>
<td>pbTimeout</td>
<td>Number of cycles required for push button timeout based on pb_seconds.</td>
</tr>
<tr>
<td>readSOC</td>
<td>Number of cycles required to check state of charge based on soc_seconds.</td>
</tr>
<tr>
<td>battHigh</td>
<td>High battery threshold, above this value the LED displays green when checking SOC (66).</td>
</tr>
<tr>
<td>battLow</td>
<td>Low Battery threshold, if below, LED displays RED and if less than 66 and greater than 33 LED displays yellow</td>
</tr>
<tr>
<td>buttonState</td>
<td>Variable to store state of pushbutton, when the button is pressed, buttonState is high = 1</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>minSOC</td>
<td>Variable to put modules to sleep if battery has low charge (10)</td>
</tr>
<tr>
<td>greenvalue</td>
<td>PWM value to decrease intensity of green LED.</td>
</tr>
<tr>
<td>ledState</td>
<td>State of LED Off=LOW, ON = HIGH</td>
</tr>
<tr>
<td>previousMillis</td>
<td>Previous value of milliseconds on previous loop</td>
</tr>
<tr>
<td>interval;</td>
<td>Millisecond value of interval to hold LED on and off while checking state of charge</td>
</tr>
<tr>
<td>interval_LO</td>
<td>Millisecond value of interval to hold LED ON while LED flashing for normal operation</td>
</tr>
<tr>
<td>interval_HI</td>
<td>Millisecond value of interval to hold LED ON while LED flashing for normal operation</td>
</tr>
<tr>
<td>interval_hold</td>
<td>Millisecond value of interval to hold LED ON while checking battery state of charge</td>
</tr>
<tr>
<td>txaddr[]</td>
<td>Define &quot;address&quot; for wireless transmission</td>
</tr>
<tr>
<td>rxaddr[]</td>
<td>Define &quot;address&quot; for wireless transmission</td>
</tr>
<tr>
<td>x_max</td>
<td>Maximum X axis vibration from joystick</td>
</tr>
<tr>
<td>x_min</td>
<td>Minimum X axis vibration from joystick</td>
</tr>
<tr>
<td>z_max</td>
<td>Maximum Z axis vibration from joystick</td>
</tr>
<tr>
<td>z_min</td>
<td>Minimum Z axis vibration from joystick</td>
</tr>
</tbody>
</table>
### Appendix C

**Receiver Variables**

Table C-1: list of the variables used in the receiver program.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Notes and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFETY_MODE</td>
<td>A user defined constant used to determine when the receiver will enter safety mode. Safety mode will start after SAFETY_MODE seconds of not receiving any messages from the joystick. Default value: 1s.</td>
</tr>
<tr>
<td>MESSAGE_FREQUENCY</td>
<td>A user defined constant used to determine the frequency of CAN messages sent to the grain cart. A message will be sent to the grain cart every MESSAGE_FREQUENCY milliseconds. Default value: 50ms.</td>
</tr>
<tr>
<td>timeStamp</td>
<td>Stores the result of millis() from the last CAN message transmission. Used to determine when to send the next message to the grain cart.</td>
</tr>
<tr>
<td>inbuf[]</td>
<td>Character (char) array which stores the last message received from the transmitter.</td>
</tr>
<tr>
<td>message</td>
<td>64 bit variable which stores the intermediate between char array and byte array conversion.</td>
</tr>
<tr>
<td>data[]</td>
<td>Byte array which contains the last recovered CAN message to be sent to the grain cart.</td>
</tr>
<tr>
<td>sleepData[]</td>
<td>Byte array which contains a CAN message that corresponds to no buttons being pressed and no movement on the joystick. This is sent to the grain cart instead of data[] while the receiver is in safety mode.</td>
</tr>
</tbody>
</table>
#include <SPI.h>   // include SPI library for CAN and RF modules
#include <Enrf24.h>   // Include library for RF module
#include <nRF24L01.h>   // Include library for RF module
#include <mcp_can.h>   // Include library for CAN module
#include <I2cMaster.h>   // Include library for Battery Charger module

#define PB P1_0 // Define Pushbutton as Pin 1_0
#define AUX_PWR P1_1 // Define Auxiliary PWR to Mosfet On/Off switch for the joystick as Pin 1_1
#define GR_LED P1_2 // Define Green LED Power as Pin 1_2
#define R_LED P1_3 // Define Red LED Power as Pin 1_3
#define CAN0_INT P1_4 // Define Can Interrrupt as Pin 1_4
#define CAN_CSn P2_0 // Define CAN Chip Select as Pin 2_0
#define SDA_PIN P2_1 // Define SDA_Pin (battery babysitter) as P2_1
#define SCL_PIN P2_2 // Define SCL_Pin (battery babysitter) as P2_2
#define RF_CE P2_3 // Define RF Chip Enable as P2_3
#define RF_CSn P2_4 // Define RF SPI Chip select as P2_4
#define RF_INT P2_5 // Define RF module interrrupt as P 2_5

#define FUEL_GAUGE_SoC 0x1C // Define memory in Battery Babysitter as SOC variable
#define FUEL_GAUGE_CURRENT 0x10 // Define memory in Battery Babysitter as current variable

int timeout_mins = 5; // Timeout variable, enters sleep after set minute value
int pb_seconds = 5 // Timeout variable, enters sleep after pushbutton held for set number of seconds
int soc_seconds = 1; // Check state of charge after set time in seconds
int brightness = 5; // Declare how bright the fading LED starts at while charging
int fadeAmount = 1; // Set how many points to fade the LED by per cycle while charging

int current; // Stores value of current entering the battery
int soc; // Stores value of charge of battery from 0-100%
long unsigned int rxId; // Received eID from CAN module (not used)
unsigned char len = 0; // Received data length from CAN module (not used)
unsigned char rxBuf[8]; // Received data from CAN module
unsigned int spout_x = 0; // Variable to store current value of joystick X
unsigned int spout_z = 0; // Variable to store current value of joystick Z

const int idleTimeout = (timeout_mins * 60) / (50E-3); // Number of cycles required for timeout based on timeout_mins
const int pbTimeout = (pb_seconds) / (50E-3); // Number of cycles required for push button timeout based on pb_seconds
const int readSOC = (soc_seconds) / (50E-3); // Number of cycles required to check state of charge based on soc_seconds
const int battHigh = 66; // High battery threshold, above this value the LED displays green when checking SOC
const int battLow = 33; // Low Battery threshold, if below, LED displays RED and if less than 66 and greater than 33 LED displays yellow
int buttonState = 0; // Variable to store state of pushbutton, when the button is pressed, buttonState is high (1)
int minSOC = 10; // Variable to put modules to sleep if battery has low charge.
int greenvalue = 0; // Duty cycle value to decrease intensity of green LED
int ledState = LOW; // State of LED Off=LOW, ON = HIGH
unsigned long previousMillis = 0; // previous value of milliseconds on previous loop
unsigned long interval; // Millisecond value of interval to hold LED on/off while checking state of charge
unsigned long interval_LO = 2000; // Millisecond value of interval to hold LED ON while LED flashing for normal operation
unsigned long interval_HI = 200; // Millisecond value of interval to hold LED ON while LED flashing for normal operation
unsigned long interval_hold = 3000; // Millisecond value of interval to hold LED ON while checking battery state of charge

// Transmitter variable declarations
const uint8_t txaddr[] = "0"; // Define "address" for wireless transmission
const uint8_t rxaddr[] = "0"; // Define "address" for wireless transmission

const int JOYSTICK_THRES = 2000;
unsigned int x_max = 2048 + JOYSTICK_THRES; // max x output from non-user input
unsigned int x_min = 2048 - JOYSTICK_THRES; // min x output from non-user input
unsigned int z_max = 2048 + JOYSTICK_THRES; // max z output from non-user input
unsigned int z_min = 2048 - JOYSTICK_THRES; // min z output from non-user input
SoftI2cMaster wire(SDA_PIN, SCL_PIN); // Pin declaration of I2cMaster function to check battery current and state of charge
MCP_CAN CAN0(CAN_CSn); // Instantiate RF module and declare (CSn)
Enrf24 radio(RF_CE, RF_CSn, RF_INT); // Instantiate RF module and declare pins (CE, CSn, INT)

void setup()
{
    pinMode(CAN0_INT, INPUT); // Configuring pin for CAN Interrupt input
    pinMode(AUX_PWR, OUTPUT); // Configure MOSFET switch driver as output
    pinMode(GR_LED, OUTPUT);  // Configure green LED as output
    pinMode(R_LED, OUTPUT);   // Configure red LED as output
    pinMode(PB, INPUT_PULLUP); // Make push button input
    CAN0.begin(MCP_ANY, CAN_250KBPS, MCP_10MHZ); // Setup CAN module for our specific bit rate and oscillator frequency
    CAN0.setMode(MCP_NORMAL); // Change to normal mode to allow messages to be transmitted/received
    CAN0.setSleepWakeup(1); // Enable Can module sleep mode
    radio.begin(); // Setup RF module to defaults of 1Mbps, channel 0, max TX power
    radio.setTXaddress((void*)txaddr); // Set transmit address for data
    radio.setChannel(0); // Set transmission channel to channel 0 (2400 MHZ)
    radio.setSpeed(250000); // Set RF transmission to 250kbps (increases range, but can be increased if necessary)
    radio.setTXpower(-12); // Set transmission power to -12 dBm
}

void loop() {
    int timeout = 0; // Loop value for idle timeout counter
    int btnPush = 0; // Counter of subsequent button presses for module timeout
    int btnSoc = 0; // Counter of subsequent button presses to check battery state of charge
    int zero_counter = 0; // Counts subsequent zeros to confirm that the battery is disconnected
    int soc_flag = 0; // Flag is set high once btnPush exceeds the readSOC variable.
    int soc_counter = 0; // Checks how many consecutive cycles that soc < minSOC.
    interval = 0; // Chosen interval in milliseconds of millis function
    soc = minSOC + 10; // Initialize soc to greater than minSOC to be able to enter operational while loop
    digitalWrite(AUX_PWR, HIGH); // Power mosfet, which supplies power to the joystick
while (timeout < idleTimeout && btnPush < pbTimeout && zero_counter <= 50 && soc_counter < 10) {
    // If any of these arguments are invalid, module goes to sleep.
    // While in this loop, module is transmitting.
    // 1st arg: Do not enter loop if joystick is idle for set time (timeout)
    // 2nd arg: Do not enter loop if button is pushed for the sleep threshold time
    // 3rd arg: zero_counter checks if current is zero for 50 loops (batt is disconnected in this case)
    // 4th arg: Checks if state of charge is less than minSOC for 10 loops.

    if (!digitalRead(CAN0_INT)) // If CAN0_INT pin is low, read receive buffer of CAN module
    {
        CAN0.readMsgBuf(&rxId, &len, rxBuf); // Read data from CAN module and store data in rxBuf array
        unsigned long message = elmersCANToLong(rxBuf);
        radio.println(message); // Add message to RF transmit buffer
        radio.flush(); // Force transmission of message to receiver
        if (noJoystickInput(rxBuf)) timeout++; else timeout = 0;
        // Check if joystick had buttons pushed
        buttonState = digitalRead(PB);
        if (buttonState == LOW){
            btnPush++; // Increment button push counter for sleep threshold counter
            btnSoc++; // Increment button push counter for state of charge counter
            if (btnSoc >= readSOC){ // Soc_flag is used to toggle the soc check.
                soc_flag = 1;
            }
        }
        else{ // If no subsequent button press, set both counters to zero
            btnPush=0;
            btnSoc=0;
        }
    }
}

soc = wire.fuelGaugeRead(FUEL_GAUGE_SoC); // i2cmaster function that retrieves battery state of charge 0-100%
current = wire.fuelGaugeRead(FUEL_GAUGE_CURRENT); // i2cmaster function that retrieves current entering the battery
if (current == 0){ // Checks how many times current is zero subsequently, this is to assure our battery is disconnected (current = 0)
    zero_counter++;
else{
    zero_counter = 0;
}

if (soc < minSOC){
    soc_counter++;
} else{
    soc_counter = 0;
}

if (current > 0) { // LED is charging in this state, LED fades.
    analogWrite(GR_LED, brightness); // Declare analog input
    analogWrite(R_LED, LOW); // Turn red LED off, not needed in this state
    brightness = brightness + fadeAmount; // increment LED brightness by fade value during each loop.
    // reverse the direction of the fading at the ends of the fade:
    if (brightness == 0 || brightness == 150) {
        fadeAmount = -fadeAmount;
    }
    timeout = 0; // reset timeout so module never goes to sleep while idle and charging.
} else { // LED not charging in this state, this checks the battery state of charge,
    if (soc_flag == 1 && current < 0) {
        // 1st arg: Enters if flag is high, (previously set when pushbutton is pressed down for appropriate number of cycles)
        // 2nd arg: Enters if battery is not being charged, which means current is less than zero.
        interval = interval_hold; // switch interval so that LED stays on for 5 secs to check battery charge.
        unsigned long currentMillis = millis(); // declare most recent millisecond value of most recent loop
        if (currentMillis - previousMillis > interval)// if the subtraction is greater than "interval" we enter the if statement, this is our LED on/off time.
        {
            previousMillis = currentMillis;
            if (ledState == LOW) {
                ledState = HIGH;
                greenvalue = 30;
            }
            else {
                ledState = LOW;
            }
        }
    }
}
soc_flag = 0; //set flag to zero after checking state of charge

greenvalue = 0;

if (soc < battLow) { //LED is red for 5 secs if state of charge is below our battery low threshold.
  digitalWrite(R_LED, ledState);
}
else if (soc > battHigh) { //LED is green for 5 secs if state of charge is above our battery high threshold.
  digitalWrite(GR_LED, ledState);
}
else { //if state of charge is either the above, LED is yellow for 5 seconds to display mid battery life.
  digitalWrite(R_LED, ledState);
  analogWrite(GR_LED, greenvalue);
}

else { //LED not charging in this state "Normal operation state" LED flashes on for 200ms and off for 2000ms
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis > interval)
  {
    previousMillis = currentMillis;
    if (ledState == LOW) {
      ledState = HIGH; // toggle state variable for LED high
      interval = interval_HI; // swap interval to small value so LED is on only briefly
    }
    else {
      ledState = LOW; // toggle state variable for LED Low
      interval = interval_LO; // swap interval to large value so LED is off for a large period of time
    }
  }
  digitalWrite(R_LED, LOW); // Ensure red LED is off
  digitalWrite(GR_LED, ledState); // turn LED off or on depending on state variable set previously.
}

digitalWrite(GR_LED, LOW); // Turn Red and Green LED off
digitalWrite(R_LED, LOW);
CAN0.setMode(MCP_SLEEP); // Put CAN module to sleep
Serial.flush();
radio.deepsleep(); // Put RF module to sleep
digitalWrite(AUX_PWR, LOW); // Cut power from joystick
zero_counter = 0; // Reset zero counter
soc_counter = 0;
delay(3000); // To debounce
attachInterrupt(PB, interrupt, FALLING); // Allow module to be woken up by pushbutton press
suspend(); // Enter sleep mode
}

void interrupt()
{
wakeup(); // Wake up from sleep mode
digitalWrite(AUX_PWR, HIGH); // Apply constant power to joystick
radio.enableRX(); // Wake up RF module
CAN0.MCPwakeup(); // Wake up CAN Module
CAN0.begin(MCP_ANY, CAN_250KBPS, MCP_10MHZ);
CAN0.setMode(MCP_NORMAL);
detachInterrupt(PB); // Remove pushbutton wake up interrupt
}

long elmersCANToLong(uint8_t rxBuf[])
{
    long message = 0;
    message |= (long)rxBuf[5] << 24;
    message |= (long)rxBuf[0] << 16;
    message |= (long)rxBuf[1] << 12; // Add received CAN data to message to be sent by RF module
    message |= (long)rxBuf[2] << 4;
    message |= (long)rxBuf[3];

    return message;
}

int noJoystickInput(uint8_t rxbuf[])
{
    if (rxBuf[5] != 0) return 0;
    int spout_x = (rxBuf[1] << 8) + (rxBuf[0]);
    int spout_z = (rxBuf[3] << 8) + (rxBuf[2]);
    if ((spout_x < x_min) | (spout_x > x_max)) return 0;
    if ((spout_z < z_min) | (spout_z > z_max)) return 0;

    return 1;
}
Appendix E

Receiver Code

```c
#include <SPI.h> // include SPI library since CAN and RF modules use SPI
#include <Enrf24.h> // Include library for RF module
#include <nRF24L01.h> // Include library for RF module
#include <mcp_can.h> // Include library for RF module
#include <string.h> // Include string library for "Serial.print" command for testing

#define CAN0_INT P1_4
#define CAN_CSn P2_0
#define RF_CE P2_3
#define RF_CSn P2_4
#define RF_INT P2_5

#define SAFETY_MODE 1 // how long before the reciever package goes into safety mode (seconds)
#define MESSAGE_FREQUENCY 50 // the rate at which the CAN messages are sent to the falcon controller (milliseconds)

MCP_CAN CAN0(CAN_CSn); // Instantiate RF module and declare (CSn)
Enrf24 radio(RF_CE, RF_CSn, RF_INT); // Instantiate RF module and declare pins (CE, CSn, INT)

byte data[8] = {0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00}; // Create variable for CAN message to be sent to the falcon controller
byte sleepData[8] = {0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00}; // Create variable for CAN message to be sent to the falcon controller when in sleep mode
uint32_t message = 0; // Create 64 bit variable for received RF module message

const uint8_t rxaddr[] = "0"; // Define "address" for wireless transmission
const long ADDR = 0x00EF0010; // Define 29-bit CAN extended ID

unsigned long timeStamp = 0;
unsigned long temp = millis();
unsigned long missedTransmissions = SAFETY_MODE * 20; // initialize so that the program starts in safety mode.
```
#define LED_R P1_3
#define LED_G P1_2

void setup() {
    SPI.begin();       // Set SPI pins (MISO = P1_6, MOSI = P1_7, SCLK = P1_5)
    SPI.setDataMode(SPI_MODE0); // Set SPI mode to MODE0
    SPI.setBitOrder(MSBFIRST); // These 2 lines may not be necessary, ARE THESE NECESSARY?
    CAN0.begin(MCP_ANY, CAN_250KBPS, MCP_10MHZ);
    CAN0.setMode(MCP_NORMAL); // Change to normal mode to allow messages to be transmitted/received
    radio.begin();       // Setup RF module to defaults of 1Mbps, channel 0, max TX power
    radio.setRXaddress((void*)rxaddr); // Set receive address to previously defined
    radio.setChannel(0); // Set transmission channel to channel 0 (2400 MHZ?)
    //radio.setCRC(1, 1);
    //radio.autoAck(1);     // Set automatic acknowledgements (15 times at 250 us intervals)
    radio.setSpeed(250000); // Set RF transmission to 250kbps (increases range, but can be increased if necessary)
    radio.enableRX();    // Start listening from RF module for transmissions
    pinMode(LED_R, OUTPUT);
    pinMode(LED_G, OUTPUT);
}

void loop() {
    char inbuf[10]; // define variable for rf message reception
    if (radio.read(inbuf, 10)) { // read 10 bytes from rf module and store in inbuf array
        missedTransmissions = 0;
        message = atol(inbuf); // convert received char array (ASCII codes) into 64bit variable
        data[5] = ((message >> 24) & 0xFF);  //
        data[6] = ((message >> 16) & 0xFF);  //
        data[1] = ((message >> 12) & 0xF);  // Convert received RF messages to CAN data field
        data[2] = ((message >> 4) & 0xFF);  //
        data[3] = (message & 0xF);  //
    }
}
temp = millis();
if ((temp - time) > (MESSAGE_FREQUENCY - 1)) { // code in here will run every MESSAGE_FREQUENCY milliseconds
    time = millis();
    missedTransmissions++;

    if (missedTransmissions > (SAFETY_MODE * 20)) {
        // Expected messages from TX but didn't get any
        digitalWrite(LED_R, HIGH);
        byte sndStat = CAN0.sendMsgBuf(ADDR, 1, 8, sleepData); // Send predetermined message from Elmers.h file
        delay(10);
        digitalWrite(LED_R, LOW);
    }
    else {
        // Normal operation
        digitalWrite(LED_G, HIGH);
        digitalWrite(LED_R, LOW); // make sure the red LED is off
        byte sndStat = CAN0.sendMsgBuf(ADDR, 1, 8, data); // create CAN message send to controller (Address, eID = 1, 8 data bytes, data)
        delay(10);
        digitalWrite(LED_G, LOW);
    }
}
}
Appendix F

User Manual

Elmer’s Wireless Joystick
User Manual
# Table of Contents

Specifications ............................................................................................................................................. 61
Operation: ................................................................................................................................................ 65
  Full Power Operation, Non-Charging: ................................................................................................. 66
  Full Power Operation – Charging: ......................................................................................................... 67
  SOC Check State .................................................................................................................................. 67
  Sleep Mode – Not Charging .................................................................................................................. 67
  Sleep Mode – Charging ......................................................................................................................... 67
Troubleshooting ...................................................................................................................................... 68
  Transmitter Issues ............................................................................................................................... 68
  Receiver Issues ................................................................................................................................... 68
Parts List: .................................................................................................................................................. 70
Schematics .................................................................................................................................................. 71
Specifications

Table F-1: Proposed vs. Actual Performance Metrics

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Proposed Value</th>
<th>Actual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Tractor Cab to Grain Cart ~ 30 ft</td>
<td>Tractor Cab to Grain Cart ~ 30 ft</td>
</tr>
<tr>
<td>Battery Life (Continuous)</td>
<td>&gt; 4 Hours</td>
<td>&gt; 34 Hours</td>
</tr>
<tr>
<td>Battery Life (Standby)</td>
<td>&gt; 7 Days</td>
<td>&gt; 9 Days</td>
</tr>
<tr>
<td>Operating Delay</td>
<td>&lt; 500ms</td>
<td>29ms</td>
</tr>
<tr>
<td>Enclosure Size</td>
<td>≤ 355 cm³</td>
<td>35 cm³</td>
</tr>
<tr>
<td>Discharge Rate</td>
<td>&lt; 500mA</td>
<td>140mA (max)</td>
</tr>
</tbody>
</table>

The required range was a metric chosen by Elmer’s Manufacturing to ensure that the wireless signal could reach the grain cart’s receiver package from the wireless joystick package inside the tractor cab. This metric was tested at the Elmer’s Manufacturing facility.
Figure F-1: Measured Discharge Rate in Full Power Operation

The operational battery life was a metric chosen by Elmer’s Manufacturing to satisfy a day of harvest. Operational battery life was tested by allowing the module to transmit for an extended period of time. State of charge was continuously monitored throughout the test. Figure F-2 shows that a fully charged battery would last more than 34 hours.
Battery life in sleep mode was required to last more than one week. This metric was chosen by Elmer’s Manufacturing as it allowed the wireless joystick to go without charging for a few days without worrying about the battery depleting after each day. This feature was implemented by adding software that puts the transmitter to sleep. The power to the joystick is also cut off with an electrical switch. Testing was done similarly to the operational battery life; however, the modules were put to sleep and the joystick was unpowered. Testing showed that the sleep mode battery life was observed as greater than 9 days, which met the design metrics as shown in Figure F-2.
Operational delay was considered to ensure that the wireless design did not introduce noticeable delay between the joystick operation and the hydraulic functions. This metric was determined to be 29ms by measuring the time between a button press on the joystick and an actuator signal from the hydraulic controller.

Enclosure size was a metric specified by Elmer’s Manufacturing so that the wireless joystick could fit in a cup holder of the tractor cab. To meet this metric, the enclosure was designed around a reference provided by Elmer’s Manufacturing. The receiver enclosure was designed to be mounted to the grain cart.

To allow for operation during charging, the discharge rate was required to be less than the charging rate supplied by micro-USB. The charger supports 500mA of charging current, and the maximum discharge current was 140mA. Therefore, the design goal of charging capabilities was met.
Operation:

Figure F-4: Transmitter Enclosure Model. (a) Top View Inside Enclosure with Joystick Removed. (b) View of Exterior of Enclosure
Full Power Operation, Not Charging:

During this state, the green light blinks once approximately every 2 seconds. The transmitter is transmitting during this state. This will continue until one of four things occurs: First of all,

- If the battery is disconnected, the modules do not operate.
- No user input for 5 minutes
- Push button is pressed and held for 5 seconds
- Battery life below 10%

When the first condition occurs, no power is supplied to the modules, and no operation occurs. During the next two conditions, the transmitter enters low power mode and can be woken up with a pushbutton press. The last condition requires the battery to be charged above 10% to resume use.
Full Power Operation, Charging:

During this state, the light fades green. The transmitter is transmitting during this state. This state continues to operate like the non-charging state, however is not put to sleep by user inactivity. Once unplugged, the transmitter enters the non-charging state.

SOC Check State

This state can only be entered while the battery is not charging. During this state, the light is on for three seconds and displays a colour according to the state of charge. While the battery is above 66%, the light displays green, if below 33%, red, and if in between 66% and 33%, displays yellow. This state is entered when a user holds down the pushbutton switch for one second. Once this state finishes, the transmitter packages return to the non-charging operation state.

Sleep Mode, Not Charging

During this mode, the transmitter is in low power and not transmitting. The transmitter will remain in this mode until a user presses the pushbutton switch. When the pushbutton is pressed, the transmitter is woken up and resumes transmitting.

Sleep Mode, Charging

This mode behaves similarly to the Sleep (Not Charging) state. The transmitter is being charged in this state.
## Troubleshooting

### Transmitter Issues

Table F-2: Transmitter Issues and Solutions.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Green indicator light not blinking</strong></td>
<td>During normal operation, the green light should either blink once every 2 seconds (not charging) or fade in and out (while charging)</td>
</tr>
<tr>
<td></td>
<td>Verify battery disconnect switch in in the ON position. The red light on the battery charging module should be ON</td>
</tr>
<tr>
<td></td>
<td>Press the pushbutton to verify the transmitter has not entered sleep mode. The green light should now start blinking</td>
</tr>
<tr>
<td></td>
<td>Plug in the micro-USB charger. The transmitter shuts down when the battery is below 10% charge to prevent damage</td>
</tr>
<tr>
<td></td>
<td>Verify the 6-pin joystick connector is properly connected to the wiring inside the transmitter enclosure</td>
</tr>
<tr>
<td></td>
<td>Verify all 5 wires of the screw terminal on the circuit board are installed as shown in the schematics below and the screws are tight</td>
</tr>
<tr>
<td><strong>Battery not charging</strong></td>
<td>During normal battery charging, the green light will slowly fade in and out</td>
</tr>
<tr>
<td></td>
<td>Verify the micro-USB connector is properly inserted in the battery charger board. The blue light should be on whenever the micro-USB cable is plugged in and powered</td>
</tr>
<tr>
<td></td>
<td>Verify the physical disconnect switch is in the ON position. The battery does not charge while the switch is OFF</td>
</tr>
</tbody>
</table>
Receiver Issues

Table F-3: Receiver Issues and Solutions.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No lights on receiver</strong></td>
<td>Verify the 2-pin connector from the receiver is properly plugged into the grain cart wiring harness</td>
</tr>
<tr>
<td></td>
<td>Verify the grain cart has power supplied</td>
</tr>
<tr>
<td><strong>Red light on Receiver</strong></td>
<td>Verify the external antenna on the receiver is inserted and securely tightened</td>
</tr>
<tr>
<td></td>
<td>Verify the transmitter is transmitting (see above sections)</td>
</tr>
<tr>
<td><strong>Green Light on receiver but no auger movement</strong></td>
<td>Verify the 3-pin connector from the receiver is correctly inserted into the 3-pin connector on the grain cart wiring harness</td>
</tr>
<tr>
<td></td>
<td>Verify the 3 wires inside the receiver are correctly installed in the screw terminals as shown in the schematics below. Ensure the screws are tight</td>
</tr>
<tr>
<td></td>
<td>Verify the hydraulics on the grain cart are active</td>
</tr>
<tr>
<td></td>
<td>If available, use the Elmer’s Manufacturing PC Tools software to verify the joystick is detected and no errors are present.</td>
</tr>
</tbody>
</table>
Parts List:

The following parts list contains all the required components for both the transmitter and receiver packages.

Table F-4: Parts list.

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Manufacturer Part Number</th>
<th>Vendor Part Number</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>2</td>
<td>MSP430G2553</td>
<td>296-28431-1-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>Wireless Module</td>
<td>1</td>
<td>WRL-00691</td>
<td>0293-SPF</td>
<td>Canada Robotix</td>
</tr>
<tr>
<td>Wireless Module w/ External Antenna</td>
<td>1</td>
<td>WRL-00705</td>
<td>1568-1296-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>CAN Bus Transceiver Module</td>
<td>2</td>
<td>MIKROE-986</td>
<td>1471-1324-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>Lithium-Ion Polymer (LiPo) Battery</td>
<td>1</td>
<td>0233-PRT</td>
<td>0233-PRT</td>
<td>Canada Robotix</td>
</tr>
<tr>
<td>Battery Babysitter LIPO Battery</td>
<td>1</td>
<td>PRT-13777</td>
<td>1568-1404-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>20W Adjustable DC-DC Buck Converter</td>
<td>1</td>
<td>DFR0379</td>
<td>1738-1293-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>DC-DC Boost Converter</td>
<td>1</td>
<td>DFR0123</td>
<td>1738-1144-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>Linear Regulator</td>
<td>2</td>
<td>MCP1825S-3302E/AB</td>
<td>MCP1825S-3302E/AB-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>N-Channel MOSFET</td>
<td>1</td>
<td>IRL40B209</td>
<td>IRL40B209-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>Push button Switch</td>
<td>1</td>
<td>3-1825910-1</td>
<td>450-1643-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>2.45GHz RP-SMA Antenna</td>
<td>1</td>
<td>W1027</td>
<td>553-1310-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>3-Position Screw Terminal</td>
<td>1</td>
<td>1725669</td>
<td>277-1274-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>5-Position Screw Terminal</td>
<td>1</td>
<td>OSTVN05A150</td>
<td>ED10564-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>RGB LED</td>
<td>2</td>
<td>WP154A4SUREQBFZGW</td>
<td>754-1492-ND</td>
<td>Digikey</td>
</tr>
<tr>
<td>PCB (Price Approximated)</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
<td>PCBWay</td>
</tr>
<tr>
<td>6 Pin Plug Deutsche Connector</td>
<td>1</td>
<td>DTM06-6S</td>
<td>571-DM06-6S</td>
<td>Mouser</td>
</tr>
<tr>
<td>6 Pin Connector Wedge Lock</td>
<td>1</td>
<td>WM-6S</td>
<td>571-WM-6S</td>
<td>Mouser</td>
</tr>
<tr>
<td>Deutsche Socket Terminals (Loose)</td>
<td>6</td>
<td>1062-20-0222 (Loose Piece)</td>
<td>571-1062-20-0222-LP</td>
<td>Mouser</td>
</tr>
<tr>
<td>2 Pin Receptacle Deutsche Connector</td>
<td>1</td>
<td>DT04-2P</td>
<td>571-DT04-2P</td>
<td>Mouser</td>
</tr>
<tr>
<td>2 Pin Connector Wedge Lock</td>
<td>1</td>
<td>W2-P</td>
<td>571-W2-P</td>
<td>Mouser</td>
</tr>
<tr>
<td>3 Pin Receptacle Deutsche Connector</td>
<td>1</td>
<td>DT04-3P</td>
<td>571-DT04-3P</td>
<td>Mouser</td>
</tr>
<tr>
<td>3 Pin Connector Wedge Lock</td>
<td>1</td>
<td>W3P-1939</td>
<td>571-W3P-1939</td>
<td>Mouser</td>
</tr>
<tr>
<td>Deutsche Pin Terminals (Loose)</td>
<td>5</td>
<td>1060-16-0622 (Loose Piece)</td>
<td>571-1060-16-0622-LP</td>
<td>Mouser</td>
</tr>
</tbody>
</table>
Schematics

As shown below, is the wiring schematic for all modules used in the transmitter and receiver design.

![Schematic Diagrams]

Figure F-6: Schematic of the wireless package (part 1 of 2). Continued in Figure F-7.
Figure F-7: Schematic of the wireless package (part 2 of 2).
Appendix G

System Testing

Due to the file size requirement of the final report full the test data could not be included in this Appendix. These can be accessed by downloading the contents of the USB enclosed with this report. Should the reader of this document require access to this USB please contact the University of Manitoba Electrical and Computer Engineering Department.

G.1 CAN Test

These test data were gathered over 3 separate 2.5-hour periods. A byte array of length 8 was randomly generated by the receiver and sent to the transmitter via the CAN/SPI converter. The list of sent messages and the list of received messages were placed side by side and compared in an Excel spreadsheet. The sheet contents are shown here. It was discovered after the first period that long numbers are truncated in excel. To avoid this, the messages were copied as text instead of numbers in the subsequent tests.

G.2 Wireless Test

These test data were gathered over a single 6-hour period. An integer between 0 and 10 000 000 was randomly generated by the transmitter and sent to the receiver via the RF module. The list of sent messages and the list of received messages were placed side by side and compared in an Excel spreadsheet. The sheet contents are shown here.

G.3 Full Power Battery Test

These test data were gathered over a single 45-hour period. The data near the start was invalid due to an incorrect setup. The data near the end is not used since the software battery limit has been put in place. The data was cropped to show only valid data, and saved to an excel spreadsheet. The sheet contents are shown here.
G.4 Standby Battery Test

These test data were gathered over a single 56-hour period. The test was intended to run longer, but a power outage in the building ended the test earlier than planned. The state of charge and current were recorded at 5 second intervals, and saved to an excel spreadsheet. The sheet contents are shown here.