University of Notre Dame 2021-2022

Notre Dame Rocketry Team
Launch Vehicle Identification System (LVIS)

EE Senior Design 2022
Final Report

Len Pieroni
Stephen Bolster

Table of Contents

Introduction

Every year the Notre Dame Rocketry Team (NDRT) competes in the NASA Student Launch Initiative. It is open to high school and college teams who design, build, and launch a launch vehicle to meet NASA's mission requirements. Over the course of the year, each team participates in three design reviews: Preliminary Design Review, Critical Design Review, and Launch Readiness Review. The purpose of these reviews is to document the design process and receive feedback from NASA engineers on safety, mission performance, etc. The team must demonstrate the safety of the launch vehicle with at least one successful vehicle demonstration flight and pass the Launch Readiness Review to be eligible for the final competition in Huntsville, AL. Teams are scored based on four written reports, three design reviews, competition launch performance, safety, and other factors.

The mission description for this year is found in the 2022 NASA Student Launch Handbook and is quoted below:

> *"Teams shall design a payload capable of autonomously locating the launch vehicle upon landing by identifying the launch vehicle's grid position on an aerial image of the launch site without the use of a global positioning system (GPS). The method(s)/design(s) utilized to complete the payload mission will be at the teams' discretion and will be permitted so long as the designs are deemed safe, obey FAA and legal requirements, and adhere to the intent of the challenge."*

NDRT considered several different methods to complete this mission (documented in the PDR report linked on this project's website), and eventually settled on a non-deploying inertial navigation system. Nicknamed LVIS, or Launch Vehicle Identification System, the payload carried a triple-redundant system of Raspberry Pi's, high-G inertial measurement units (IMUs), and low-G IMUs. The Raspberry Pi's ran a program that continuously monitored the IMUs while waiting for launch. After launch detection, the initial orientation was recorded and the IMU data was continuously written to a data file until landing detection. Once landed, an algorithm was run on the saved data to integrate the acceleration and rotation from launch to landing and calculate a total displacement. This displacement was converted into a grid number corresponding to its position on the launch field. The grid number was then transmitted by the payload to the ground station. Accuracy of the transmitted grid number was verified by GPS.

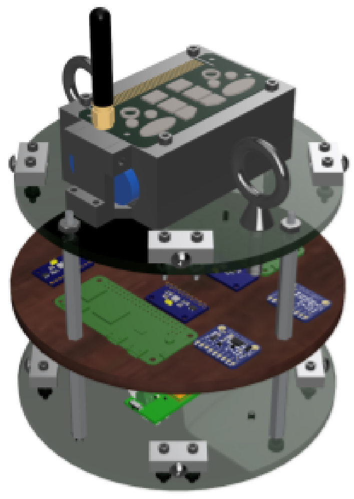


Figure 1: Launch Vehicle Identification System CAD model

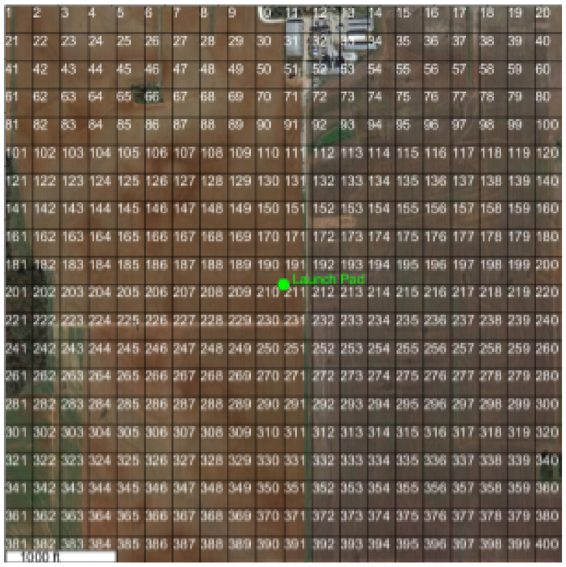


Figure 2: Gridded Launch Field Image

In close collaboration with the NDRT Payload squad, this EE Senior Design group built the power supply and wireless transmission system for both LVIS and the ground station. These two systems were integrated into a single custom printed circuit board in order to minimize their total mass and volume. Required functionality included distributing power from the battery to each of the payload's active components, receiving messages from the inertial navigation system's master Pi, transmitting the messages from the launch vehicle to the ground station, and displaying them at the ground station. The system also needed to comply with relevant constraints on frequency and transmission power set by NASA and the FCC, as well as meet range and reliability requirements derived from the expected launch conditions. Robustness and reliability are the highest priorities for this project due to the extreme conditions in which it operates and its functionality being essential for mission success.

Detailed System Requirements

NASA Requirements

- "2.23.8 Transmissions from onboard transmitters, which are active at any point prior to landing, will not exceed 250 mW of power (per transmitter).
  - Operates at 50mW, can increase but not beyond 150mW
- "2.23.9 Transmitters will not create excessive interference. Teams will utilize unique frequencies, handshake/passcode systems, or other means to mitigate interference caused to or received from other teams"
  - Uses LoRa channel 17 (905.7 MHz) to mitigate interference
- "2.7 The launch vehicle and payload will be capable of remaining in launch-ready configuration on the pad for a minimum of 2 hours without losing the functionality of any critical on-board components, although the capability to withstand longer delays is highly encouraged"

Hardware Requirements

- Total payload mass does not exceed 90 oz or 2.551 kg.
- Power delivery system delivers sufficient power to each peripheral Raspberry Pi, microcontroller, and transceiver.
- Embedded intelligence interfaces with INS, ground station, and transceiver. It passes a message received on one interface through to the other.
- Operates in outdoor temperatures between 20°F and 100°F.

- Withstands acceleration on the order of 40 Gs.
- Transmits within ISM Frequency Bands.
    - Operates at 905.7MHz (LoRa channel 17), inside 915MHz band
- Receives transmission at distances of up to 2500 feet.
    - Requires Line of Sight between payload and ground station

Detailed Project Description

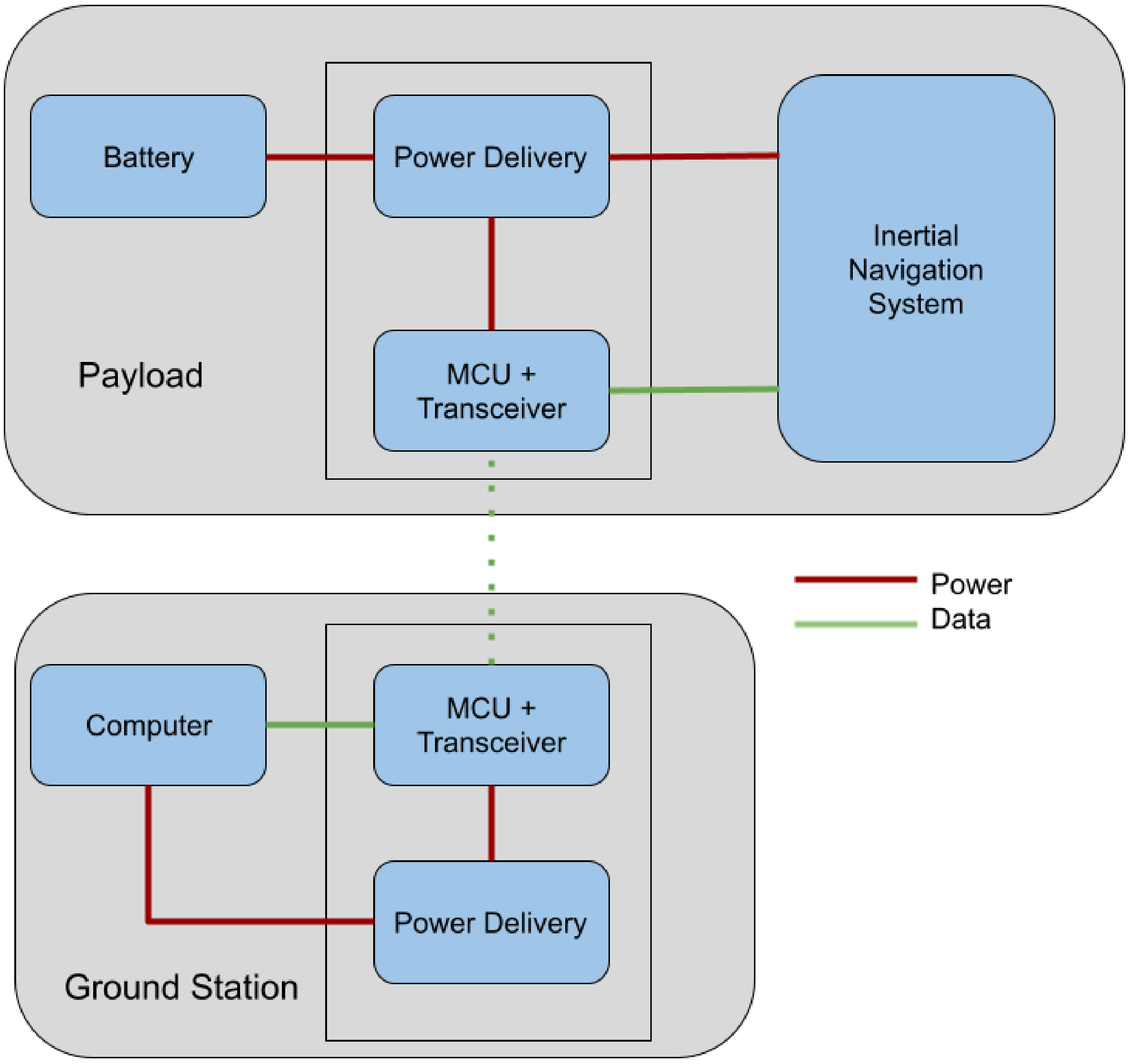
System Theory of Operation

The system consists of two identical boards. One is located inside the launch vehicle and powered by a 7.4V battery. The other board is placed at the ground station and powered via USB by a connected laptop. A serial interface enables data to be passed to the transmitter unit inside the launch vehicle and a USB-to-serial bridge enables data to be passed between the ground station's receiver unit and the connected laptop. When a packet is passed to the transmitter, it waits for the end of the packet (denoted by a '\r' or '\n' character for compatibility) and then transmits the packet to the ground station using channel 17 (905.7 MHz) of the 915MHz LoRa frequency band. The RFM95W transceiver integrated circuit used on both boards is configured for variable packet lengths so any packet small enough to fit in the 64-byte transmit and receive buffers can be sent.

The transmitter unit, located in the launch vehicle, also provides power for the payload's sensor suite. It regulates the 7.4V of the connected battery down to 5V and can provide up to 2A total on the three output connectors. Two connectors are two-pin, providing only 5V and ground connections, and the third is a four-pin connector to provide 5V, ground, and serial Tx and Rx connections. The third connector is the one that carries data between the unit and the payload’s sensors. These three connectors and the voltage regulation electronics are also present on the ground station’s board. On the receiver, the voltage regulator is disconnected from the 5V plane with a jumper clip and the power connectors are unused.

The wireless communication subsystem consists of a 3.3V LDO regulator, a PIC32MX microcontroller, an RFM95W LoRa transceiver module, and a CP2104 USB-to-serial bridge as well as a Micro USB connector. The microcontroller communicates with the transceiver over SPI using the module’s single read/write protocol, in which each register must be addressed individually and separated by a HIGH signal on the Chip Select line. The interface between the microcontroller and the CP2104 uses UART serial and the CP2104 communicates with a laptop with a Micro USB cable and the USB protocol.

System Block Diagram



Detailed Design/Operation of Power Supply

The ground station does not use the unit's power supply subsystem. On the launch vehicle's unit, the power supply provides 5V power to the INS from a 7.4V battery. A buck converter rated for 2A output lowers the 7.4V to 5V by switching the supply voltage on and off rapidly, providing higher efficiency than a linear LDO regulator. An external LC filter smooths the output of the regulator and mitigates noise. The 5V output goes through a jumper clip to the board’s 5V plane, connecting to the 5V pin on each of the three power connectors as well as the 3.3V regulator that powers the wireless transmission system. The alternate position of the jumper clip disconnects the buck converter’s output from the 5V plane and instead connects the 5V pin on the Micro USB connector to the plane, preventing damage caused by connecting the board to both a battery and a laptop simultaneously.

The primary Pi that will be communicating with the wireless transmission system connects to the four-pin connector on the board and the other two Pis use the two-pin connectors. All sensors draw power from their associated Pis.

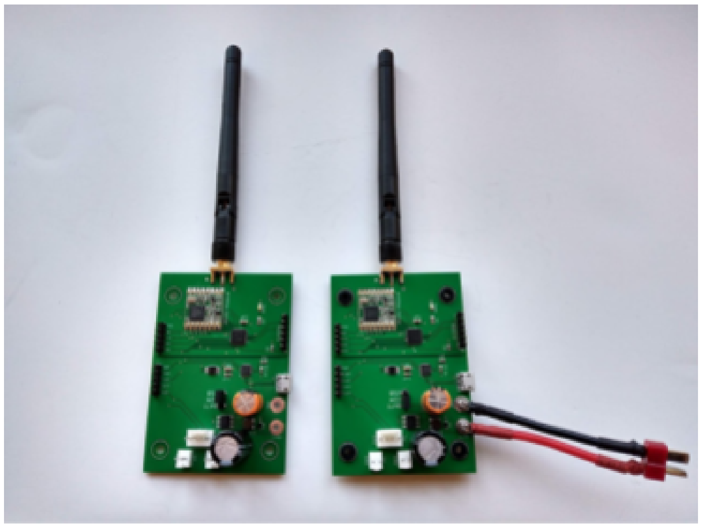


Figure 3: Assembled Receiver (left) and Transmitter (right) boards

Detailed Design/Operation of Wireless Transmission System

Both the ground station and the launch vehicle units use the wireless transmission subsystem. Inside the launch vehicle, one Raspberry Pi is designated as the primary computer, the one which communicates with the wireless transmission system via UART. This Pi creates packets to send to the ground station. These packets must be no more than 63 characters long and must be terminated with a ‘\r’ or ‘\n’ character. The packet is then sent over UART to the wireless transmitter, where the microcontroller counts the number of bytes in the packet and sends the radio module the packet length and the packet before putting the module into transmit mode. The ground station receives the packet and the onboard microcontroller reads it from the transceiver chip over SPI. Once the packet is read, the microcontroller writes it to the connected laptop using UART and the USB-to-serial bridge.

System Integration Testing

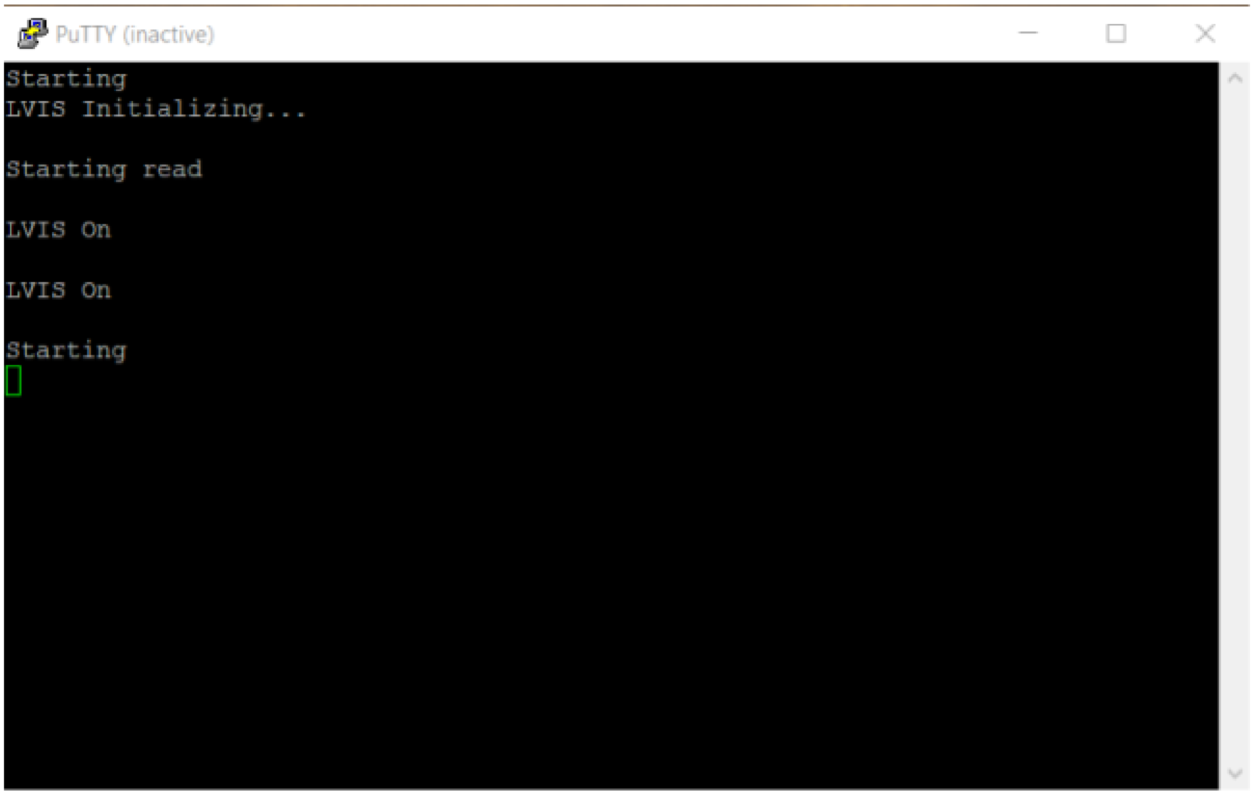
Power Supply Test: *Connect the battery to the input terminal and check the output voltages to the Raspberry Pi power connectors and the voltage pins of the microcontroller / transceiver module.*

With the battery plugged in, 5V was measured on each of the Raspberry Pi connectors and 3.3V was measured on the microcontroller / transceiver supply pins. Measurement of the expected voltages indicated that the power supply was operating nominally.

Transmission Range Test: *Separate the transmitter and receiver by at least 2500 ft (~0.5 mi) then begin transmission and observe packet reception rate.*

This test was conducted by placing the transmitter and receiver at opposite ends of South Quad on campus. Line-of-sight transmission success rate was 100% at maximum range with the transmitter inside the launch vehicle body tube.

LVIS Integration Test: *Demonstrate full payload functionality by connecting the wireless transmission system and inertial navigation system and verifying that messages sent from the master Pi are received by the ground station.*



```
Starting
LVIS Initializing...

Starting read

LVIS On

LVIS On

Starting
```

Figure 4: Screenshot of the ground station terminal after the pre-launch transmission test

This test was successfully completed prior to each launch during the rehearsal and at the launch field. Observation of the LVIS boot-up message at the ground station was required before the payload could be inserted into the launch vehicle.

Users Manual

Ground Station Setup

1. The ground station requires installation of PuTTY, a free terminal client for Windows.
2. Connect the receiver board to a PC via a micro-usb cable and ensure that the green power LED turns on.

3. Open Device Manager and verify that “CP210x USB to UART bridge” appears under the “Ports (COM & LPT)” dropdown.
4. Note the COM port number of this device.
5. Open PuTTY and select “Serial” connection type in the main window.
6. In the Serial line box, type in the “COMx” port noted during installation.
7. Type “57600” in the Speed box.
8. Click “Open” at the bottom of the window and a blank terminal window should appear.

Transmitter Setup

1. Fasten the battery connector to the transmission board by tightening an M2 screw through each of the ring connectors. Make sure to verify the correct polarity.
2. Pass four M2 screws through the mounting holes of the transmission board and the battery mounting block, and secure them with nuts on the underside of the top payload bulkhead.
3. Connect the three Raspberry Pi cables to their respective molex connectors on the transmission board.
4. Place the power supply jumper pin in the position labeled “BATT”.

Testing if the System is Working

Once powered on, the transmitter should immediately send the message “Starting” to the ground station. About two minutes later, the message “LVIS Initializing…” should be received by the ground station followed by a sequence of startup messages. These messages indicate LVIS’s current state. While it is waiting for launch, LVIS will periodically (~10s intervals) transmit “LVIS On” to indicate that it is still in the active state. If all of these transmissions are received, then LVIS is ready for liftoff.

Troubleshooting

“Starting” message not received at power on.

Check that the transmitter and receiver are both plugged in and the green power LED is on. If the transmitter is plugged in but not powered on, ensure that the jumper pin is in the “BATT” position. If the jumper pin is correct and the board still doesn’t power on, check that the battery voltage is not below 6.4V. If the battery voltage is near or below this level, recharge or replace the battery immediately.

“Starting” message received but no subsequent messages

Reboot the transmitter. Ensure that each string sent to the transmitter over UART does not exceed 63 characters and is terminated by a ‘\n’ character. Do not try to transmit messages more frequently than every 100ms to allow time for each transmission to send without overflowing the buffer. Ensure all characters are encoded in UTF8 format.

Changing the operating frequency

Open “main.c” in MPLAB X. On line 6, change FREQUENCY to the desired frequency. On line 301, make tx = 1. Connect power and the PicKit to the transmitter board and upload the script. Change line 301 to “tx = 0”. Connect power and the PicKit to the receiver board and upload the script. The system should now operate at the frequency it was changed to. Verify that the “Starting” message is still received at power on.

To-Market Design Changes

- Implement an acknowledgement signal sent by the receiver unit to the transmitter so as to improve the reliability of data transmission

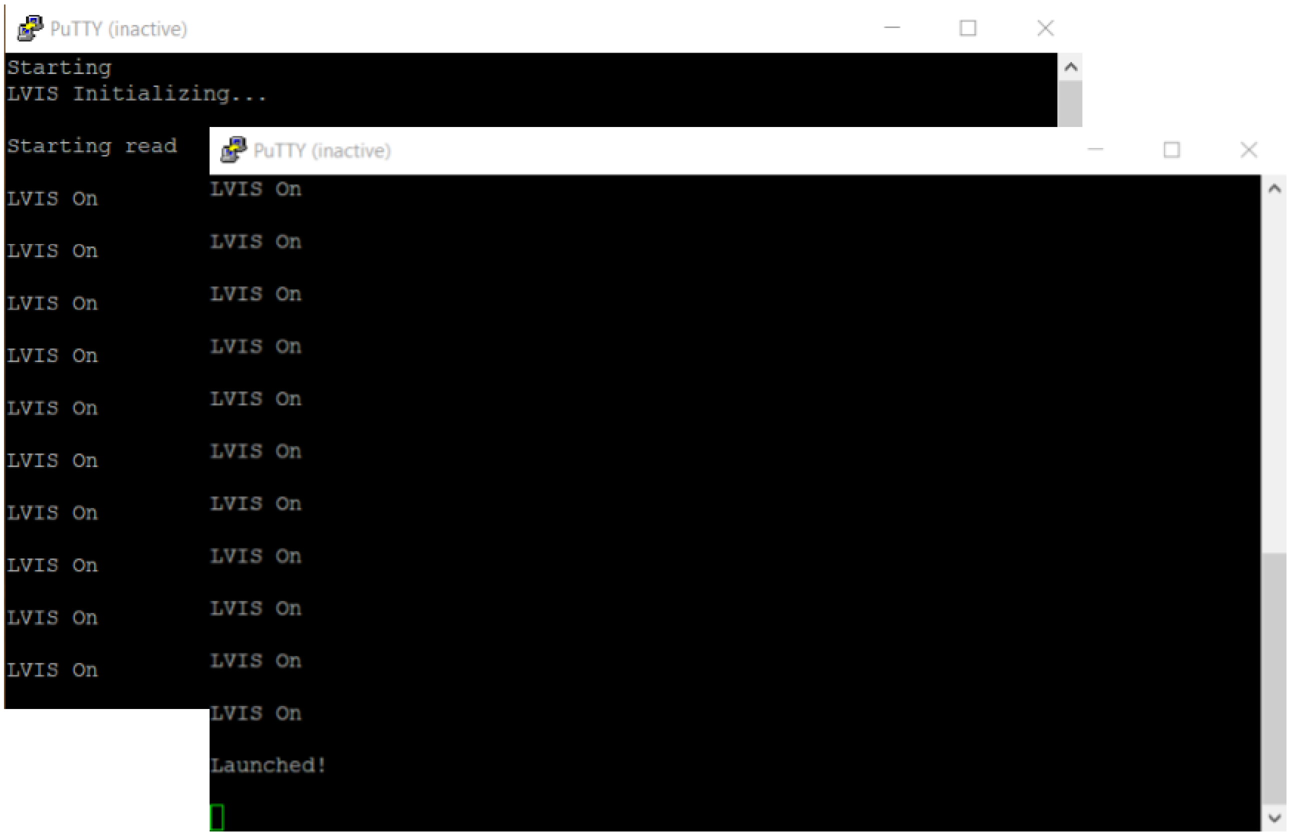
- Implement a method, such as a checksum, to verify the data received by the ground station to ensure that a byte is not corrupted or lost
- Add a way to transmit commands from the ground station to the payload, such as a reset
- Add a DIP switch to allow the board to be used as a transmitter or a receiver without reprogramming

Conclusions

The team designed and built a single board that incorporates both a 5V power supply and a wireless transceiver and that can function as both the payload transmitter and the ground station receiver. Both subsystems perform very well but have room for improvement. For example, the wireless transceiver could be programmed to enable communication from the ground station to the launch vehicle, which would allow the ground station attendant to reset the system remotely and would enable the use of an acknowledgement signal to notify the launch vehicle's transmitter that its message was received properly.

While the launch vehicle sat on the launch rail in Huntsville, the ground station reported that a launch event had been detected before takeoff actually occurred. The launch of a nearby rocket or a strong gust of wind most likely shook the rocket and caused LVIS to detect a launch event prematurely, which would have caused it to fail because LVIS would have measured a launch and a landing before the rocket's flight began instead of measuring the accurate trajectory of the vehicle. Fortunately, the ground station attendant noticed that a launch event detection was incorrectly reported and was able to notify the team's launch supervisors. They were able to power down and restart LVIS, fixing the error and restoring the payload to full functionality.

During the Huntsville flight, the system worked as intended. It reported the correct launch event after the previously discussed reset and LVIS remained powered for the duration of the flight. Unfortunately, the wireless transmission subsystem failed after landing. The launch vehicle landed on the opposite side of a small ridge from the ground station so the line of sight required for successful communication was blocked by dirt. LVIS measured the landing site correctly and the transmitter sent the coordinates, but the ground station was unable to receive the data.



```
Starting
LVIS Initializing...

Starting read

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On
```

```
LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

LVIS On

Launched!
```

Figure 5: Screenshots of the ground station terminal after the April 23 launch in Huntsville, AL

Figure 6: Launch vehicle liftoff

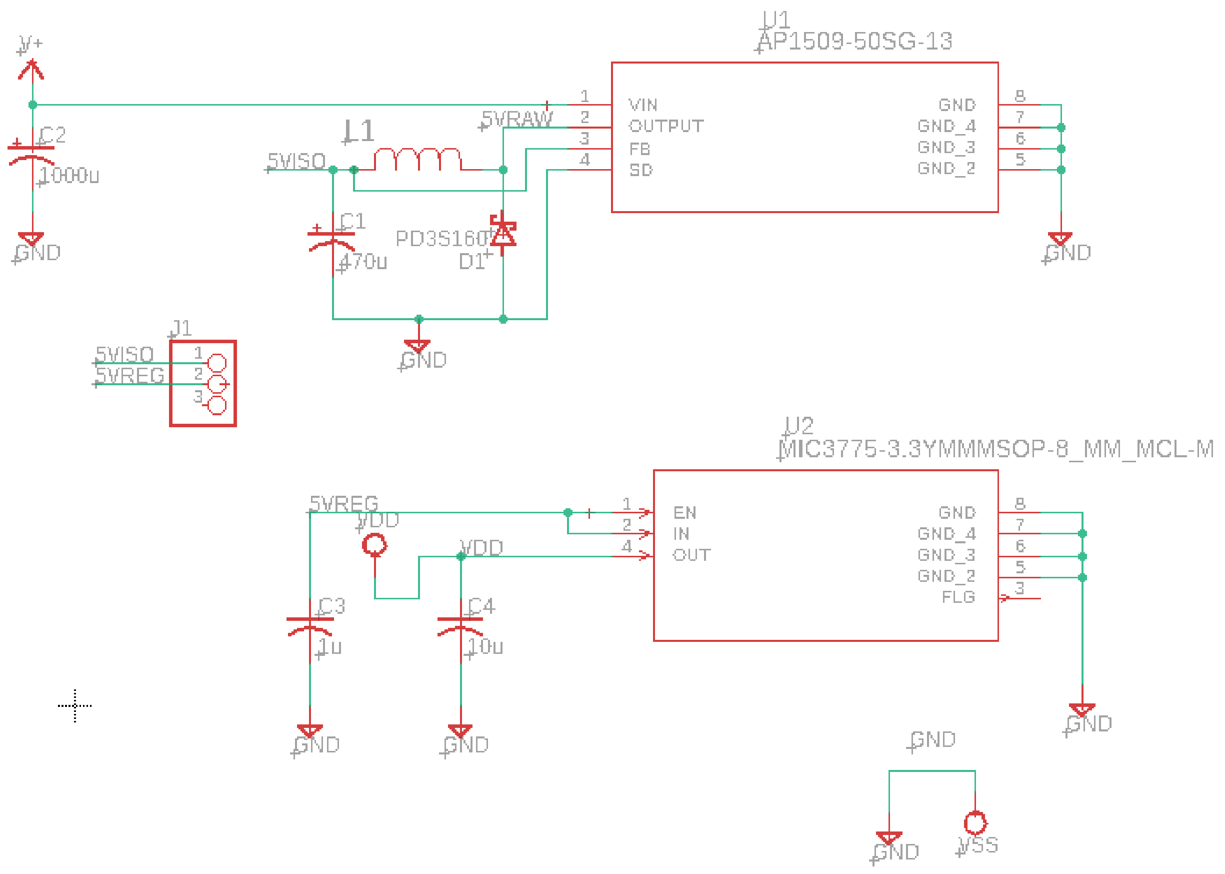
Appendices

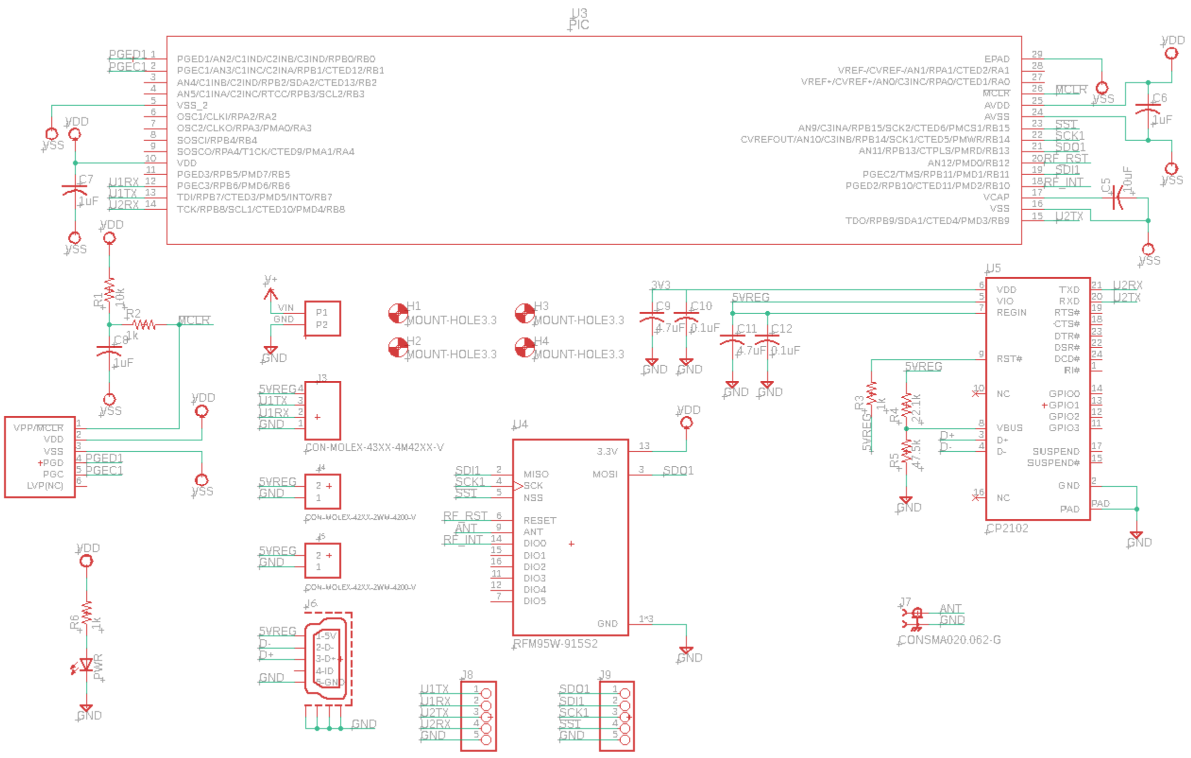
Bill of Materials

1/19/2022 Order					
Part Number	**Name**	**Description**	**Unit Price**	**Quantity**	**Total Price**
576-2280-ND	MIC3775-3.3YMM	3.3V LDO Regulator	$2.07	4	$8.28
296-49468-1-ND	TPS613222ADBVT	5V Boost Regulator	$1.16	8	$9.28
PIC32MX110F016B-I/ML-ND	PIC32MX110F016B-I/ML	PIC32 Microcontroller	$2.67	4	$10.68
RFM95W-915S2-ND	RFM95W-915S2	RF Transceiver	$13.44	4	$53.76
343-CONSMA020.062-G-ND	343-CONSMA020.062-G	SMA Antenna Connector	$2.96	4	$11.84
2151-RST-W1B6-10808-22M-FY-001-ND	2151-RST-W1B6-10808-22M-FY-001	Antenna	$4.64	2	$9.28
WM1397CT-ND	476540001	Micro USB Connector	$0.87	4	$3.48
2/2/2022 Order					
Part Number	**Name**	**Description**	**Unit Price**	**Quantity**	**Total Price**
AP1509-50SGDICT-ND	AP1509-50SG-13	5V Buck Regulator	1.45	4	5.8
541-1405-1-ND	IFSC1515AHER4R7M01	Inductor	0.61	5	3.05
495-5997-ND	B41858C4477M000	Capacitor (Buck output)	0.84	5	4.2
1189-3726-ND	25ZL1000MEFC12.5X20	Capacitor (Buck input)	0.83	5	4.15
PD3S230LQ-7DICT-ND	PD3S230LQ-7	Rectifier Diode	0.64	5	3.2
F3157CT-ND	SP0504BAHTG	ESD Diode	0.95	5	4.75
S9002-ND	SSC02SYAN	Jumper connector	0.21	6	1.26

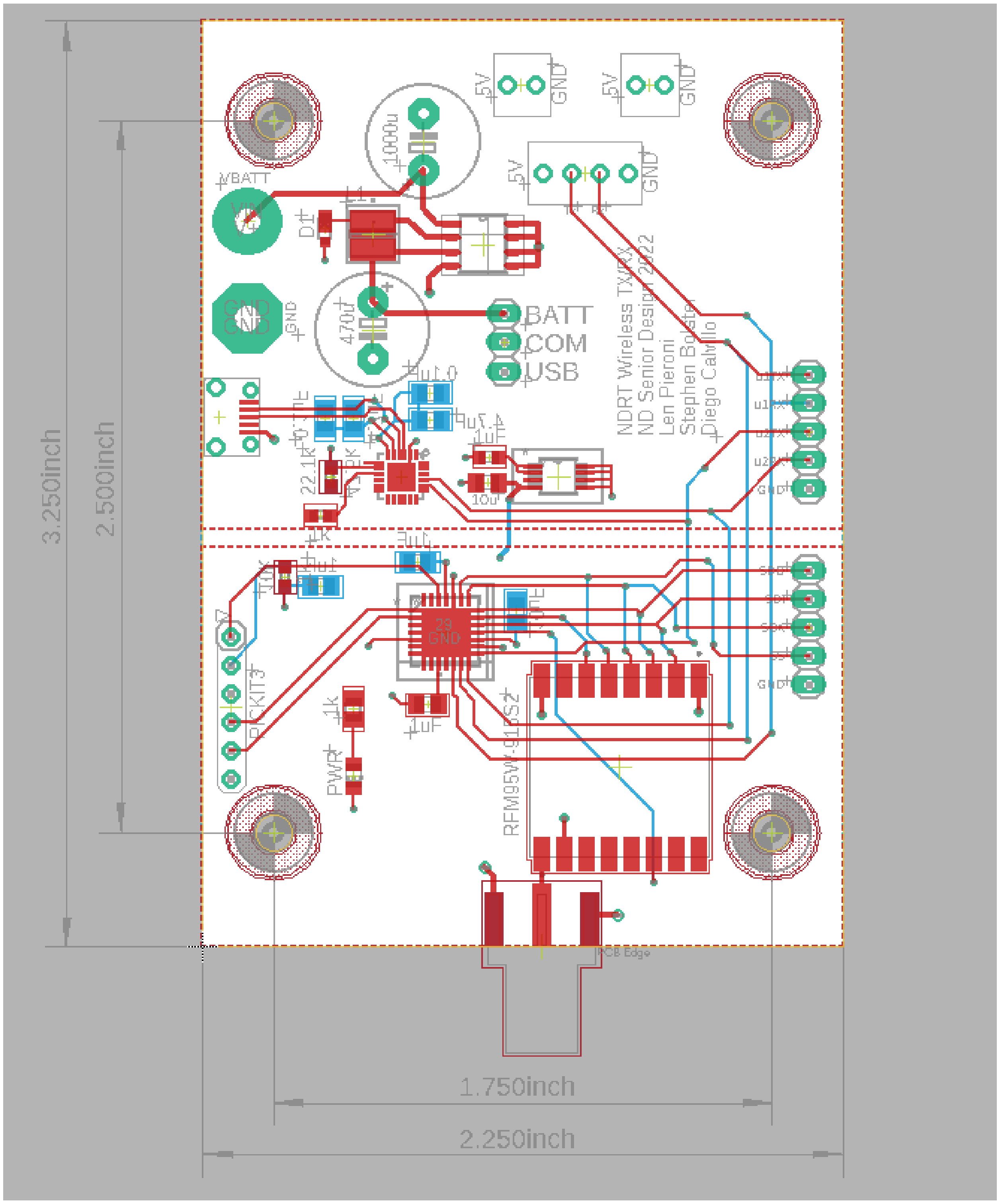
23-0190700005-ND	190700005	Ring terminals	0.23	6	1.38
PCB Order (PCBWay)		PCB	5	5	46.9
PCB Order (Osh Park)		PCB		3	44.65
				Total:	**225.94**

Hardware Schematic




U3
PIC
PGED1/AN2/C1IND/C2INB/C3IND/RPB0/RB0
PGEC1/AN3/C1INC/C2INA/RPB1/CTED12/RB1
AN4/C1INB/C2IND/RPB2/SDA2/CTED13/RB2
AN5/C1INA/C2INC/RTCC/RPB3/SCL2/RB3
VSS_2
OSC1/CLKI/RPA2/RA2
OSC2/CLKO/RPA3/PMA0/RA3
SOSCI/RPB4/RB4
SOSCO/RPA4/T1CK/CTED9/PMA1/RA4
VDD
PGED3/RPB5/PMD7/RB5
PGEC3/RPB6/PMD6/RB6
TDI/RPB7/CTED3/PMD5/INT0/RB7
TCK/RPB8/SCL1/CTED10/PMD4/RB8
EPAD
VREF-/CVREF-/AN1/RPA1/CTED2/RA1
VREF+/CVREF+/AN0/C3INC/RPA0/CTED1/RA0
MCLR
AVDD
AVSS
AN9/C3INA/RPB15/SCK2/CTED6/PMCS1/RB15
CVREFOUT/AN10/C3INB/RPB14/SCK1/CTED5/PMWR/RB14
AN11/RPB13/CTPLS/PMRD/RB13
AN12/PMD0/RB12
PGEC2/TMS/RPB11/PMD1/RB11
PGED2/RPB10/CTED11/PMD2/RB10
VCAP
VSS
TDO/RPB9/SDA1/CTED4/PMD3/RB9
PGED1
PGEC1
U1RX
U1TX
U2RX
MCLR
SS1
SCK1
SDO1
RF_RST
SDI1
RF_INT
U2TX
VDD
VSS
C7
1uF
C6
1uF
C5
10uF
R1
10k
R2
1k
C8
1uF
VPP/MCLR
VDD
VSS
PGD
PGC
LVP(NC)
V+
VIN
GND
P1
P2
H1
MOUNT-HOLE3.3
H2
MOUNT-HOLE3.3
H3
MOUNT-HOLE3.3
H4
MOUNT-HOLE3.3
J3
5VREG
U1TX
U1RX
GND
CON-MOLEX-43XX-4M42XX-V
J4
CON-MOLEX-42XX-2WM-4200-V
J5
CON-MOLEX-42XX-2WM-4200-V
J6
5VREG
D-
D+
GND
1-5V
2-D-
3-D+
4-ID
5-GND
U4
3.3V
MISO
MOSI
SCK
NSS
RESET
ANT
DIO0
DIO1
DIO2
DIO3
DIO4
DIO5
RFM95W-915S2
SDI1
SCK1
SS1
RF_RST
ANT
RF_INT
SDO1
3V3
C9
4.7uF
C10
0.1uF
C11
4.7uF
C12
0.1uF
5VREG
U5
VDD
VIO
REGIN
RST#
NC
VBUS
D+
D-
TXD
RXD
RTS#
CTS#
DTR#
DSR#
DCD#
RI#
GPIO0
GPIO1
GPIO2
GPIO3
SUSPEND
SUSPEND#
GND
PAD
CP2102
U2RX
U2TX
R3
1k
R4
22.1k
R5
47.5k
J7
ANT
GND
CONSMA020.062-G
R6
1k
PWR
J8
U1TX
U1RX
U2TX
U2RX
GND
J9
SDO1
SDI1
SCK1
SS1
GND

Board Layout



Software

regList.c

```
// registers
#define REG_FIFO                 0x00
#define REG_OP_MODE              0x01
#define REG_FRF_MSB              0x06
#define REG_FRF_MID              0x07
#define REG_FRF_LSB              0x08
#define REG_PA_CONFIG            0x09
#define REG_OCP                  0x0b
#define REG_LNA                  0x0c
#define REG_FIFO_ADDR_PTR        0x0d
#define REG_FIFO_TX_BASE_ADDR    0x0e
#define REG_FIFO_RX_BASE_ADDR    0x0f
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS            0x12
#define REG_RX_NB_BYTES          0x13
#define REG_PKT_SNR_VALUE        0x19
#define REG_PKT_RSSI_VALUE       0x1a
#define REG_RSSI_VALUE           0x1b
#define REG_MODEM_CONFIG_1       0x1d
#define REG_MODEM_CONFIG_2       0x1e
#define REG_RX_TIMEOUT           0x1f
#define REG_PREAMBLE_MSB         0x20
#define REG_PREAMBLE_LSB         0x21
#define REG_PAYLOAD_LENGTH       0x22
#define REG_MODEM_CONFIG_3       0x26
#define REG_FREQ_ERROR_MSB       0x28
#define REG_FREQ_ERROR_MID       0x29
#define REG_FREQ_ERROR_LSB       0x2a
#define REG_RSSI_WIDEBAND        0x2c
#define REG_DETECTION_OPTIMIZE   0x31
#define REG_INVERTIQ             0x33
#define REG_DETECTION_THRESHOLD  0x37
#define REG_SYNC_WORD            0x39
#define REG_INVERTIQ2            0x3b
#define REG_DIO_MAPPING_1        0x40
#define REG_VERSION              0x42
#define REG_PA_DAC               0x4d

// modes
```

```
#define MODE_LONG_RANGE          0x80
#define MODE_SHARED_REG        0x40
#define MODE_SLEEP               0x00
#define MODE_STDBY               0x01
#define MODE_TX                  0x03
#define MODE_RX_CONTINUOUS       0x05
#define MODE_RX_SINGLE           0x06

// masks
#define MASK_RX_TIMEOUT        0x80
#define MASK_RX_DONE           0x40
#define MASK_PAYLOAD_CRC_ERROR   0x20
#define MASK_VALID_HEADER        0x10
#define MASK_TX_DONE           0x08
#define MASK_CAD_DONE          0x04
#define MASK_FHSS_CHANGE_CHANNEL 0x02
#define MASK_CAD_DETECTED      0x01
```

main.c

```
#include <xc.h>
#include <stdio.h>
#include "SDlib16_delay.h"
#include "regList.c"

#define FREQUENCY 905700000     // Channel 17

// DEVCFG3
#pragma config USERID = 0xFFFF           // Enter Hexadecimal value (Enter
Hexadecimal value)
#pragma config PMDL1WAY = ON             // Peripheral Module Disable
Configuration (Allow only one reconfiguration)
#pragma config IOL1WAY = ON              // Peripheral Pin Select
Configuration (Allow only one reconfiguration)

// DEVCFG2
#pragma config FPLLIDIV = DIV_2         // PLL Input Divider (2x Divider)
#pragma config FPLLMUL = MUL_20          // PLL Multiplier (20x Multiplier)
#pragma config FPLLODIV = DIV_2         // System PLL Output Clock Divider
(PLL Divide by 2)

// DEVCFG1
#pragma config FNOSC = FRCPLL            // Oscillator Selection Bits (Fast
RC Osc with PLL)
#pragma config FSOSCEN = OFF             // Secondary Oscillator Enable
(Disabled)
#pragma config IESO = ON                 // Internal/External Switch Over
(Enabled)
#pragma config POSCMOD = OFF             // Primary Oscillator Configuration
(Primary osc disabled)
#pragma config OSCIOFNC = OFF            // CLKO Output Signal Active on the
OSCO Pin (Disabled)
#pragma config FPBDIV = DIV_4            // Peripheral Clock Divisor (Pb_Clk
is Sys_Clk/4)
#pragma config FCKSM = CSECME            // Clock Switching and Monitor
Selection (Clock Switch Enable, FSCM Enabled)
#pragma config WDTPS = PS4096            // Watchdog Timer Postscaler
(1:4096)
#pragma config WINDIS = OFF              // Watchdog Timer Window Enable
(Watchdog Timer is in Non-Window Mode)
#pragma config FWDTEN = OFF              // Watchdog Timer Enable (WDT
Disabled (SWDTEN Bit Controls))
#pragma config FWDTWINSZ = WINSZ_25     // Watchdog Timer Window Size
(Window Size is 25%)

// DEVCFG0
#pragma config JTAGEN = OFF             // JTAG Enable (JTAG Disabled)
#pragma config ICESEL = ICS_PGx1        // ICE/ICD Comm Channel Select
(Communicate on PGEC1/PGED1)
#pragma config PWP = OFF                // Program Flash Write Protect
(Disable)
#pragma config BWP = OFF                // Boot Flash Write Protect bit
(Protection Disabled)
#pragma config CP = OFF                 // Code Protect (Protection
Disabled)



char uart1_init(unsigned int baud) {
    U1RXR = 0b0001; // RB6 is U1RX
    RPB7R = 0b0001; // RB7 is U1TX
    U1MODE = 0x00000088;
    U1BRG = 10000000ul / (4*baud) - 1;
    U1STAbits.UTXEN = 1;
    U1STAbits.URXEN = 1;
    U1MODEbits.ON = 1;
    return 0;
}

char uart2_init(unsigned int baud) {
    U2RXR = 0b0100; // RB8 is U2RX
    RPB9R = 0b0010; // RB9 is U2TX
    U2MODE = 0x00000088;
    U2BRG = 10000000ul / (4*baud) - 1;
    U2STAbits.UTXEN = 1;
    U2STAbits.URXEN = 1;
    U2MODEbits.ON = 1;
    return 0;
}

char uart1_write(char tx) {
    while(U1STAbits.UTXBF); // Block while transmit buffer is full
    U1TXREG = tx;
    return tx;
```

```
}

char uart2_write(char tx) {
    while(U2STAbits.UTXBF); // Block while transmit buffer is full
    U2TXREG = tx;
    return tx;
}

char write_uart(char moduleNum, char tx) {
    switch(moduleNum) {
      case 1:
          return uart1_write(tx);
          break;
      case 2:
          return uart2_write(tx);
          break;
      default:
          return -8; // Return -8 if moduleNum is not valid
    }
}

char uart1_read() {
    if(U1STAbits.URXDA)
        return U1RXREG;
    return -7; // Return -7 if receive buffer is empty
}

char uart2_read() {
    if(U2STAbits.URXDA)
        return U2RXREG;
    return -7; // Return -7 if receive buffer is empty
}

char read_uart(char moduleNum) {
    switch (moduleNum){
      case 1:
          return uart1_read();
          break;
      case 2:
          return uart2_read();
          break;
      default:
          return -8; //Return -8 if moduleNum is not valid
    }
}

char uart1_wait_for_read() {
    while(!U1STAbits.URXDA);
    return U1RXREG;
}

char uart2_wait_for_read() {
    while(!U2STAbits.URXDA);
    return U2RXREG;
}

char wait_for_read_uart(char moduleNum) {
    switch(moduleNum) {
      case 1:
          return uart1_wait_for_read();
          break;
      case 2:
          return uart2_wait_for_read();
          break;
      default:
          return -8;
    }
}

char spi_init(unsigned int sck) {
//    SPI1CON = 0x100120AD; // 8-bit words, not Framed mode, with SS
    SPI1CONbits.ON = 0;
    SPI1CONbits.CKE = 1;
    SPI1CONbits.CKP = 0;
    SPI1CONbits.MSTEN = 1;
    SPI1CONbits.ENHBUF = 0;
    SPI1BRG = 10000000ul / (2*sck) - 1;
    RPB13R = 0x3; // RB13 is SDO1
    SDI1R = 0x3; // RB11 is SDI1
    // RB15 is SS1 (active low)
    SPI1CONbits.ON = 1;
    return 0;
}

char read_spi() {
    while(SPI1STATbits.SPIRBE); // Block while waiting for receive
    delay_us(5);
    return SPI1BUF;
    return -7; // If receive buffer is empty, return -7 (illegal receive
value)
}

char write_spi(char tx) {
    while(SPI1STATbits.SPITBF) { // While transmit buffer is full, wait
until space opens
      delay_us(1);
    }
    delay_us(5);
    return SPI1BUF = tx;
}

void write_reg_spi(char addr, char val) {
    LATBbits.LATB15 = 0;
    delay_us(5);
    write_spi(0x80 | addr);
    read_spi();
    write_spi(val);
    read_spi();
    delay_us(5);
    LATBbits.LATB15 = 1;
    delay_us(20);
}

char read_reg_spi(char addr) {
    LATBbits.LATB15 = 0;
    delay_us(5);
    write_spi(addr);
    read_spi();
    write_spi(0x00);
    char val = read_spi();
    delay_us(5);
    LATBbits.LATB15 = 1;
    delay_us(20);
    return val;
}

char write_reg_burst(char addr, char* vals, char len) {
    char i;
    LATBbits.LATB15 = 0;
    delay_us(5);
    write_spi(0x80 | addr);
    for(i = 0; i < len; i++) {
      delay_us(15);
      write_spi(vals[i]);
      read_spi();
    }
    LATBbits.LATB15 = 1;
    return len;
}

char* read_reg_burst(char addr, char* vals, char len) {
    LATBbits.LATB15 = 0;
    delay_us(5);
    write_spi(addr);
    read_spi();
    for(char i = 0; i < len; i++) {
      write_spi(0x00);
      vals[i] = read_spi();
    }
    delay_us(5);
    LATBbits.LATB15 = 1;
}

int radio_config() {
    uint64_t frf = ((uint64_t)FREQUENCY << 19) / 32000000;

    uint8_t frf_msb = (uint8_t)(frf >> 16);
    uint8_t frf_mid = (uint8_t)(frf >> 8);
    uint8_t frf_lsb = (uint8_t)(frf >> 0);


    read_reg_spi(REG_VERSION);
    write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_SLEEP)); //Set
RegOpMode to sleep
    delay_us(50);
    read_reg_spi(REG_OP_MODE);
    delay_us(50);
//    write_reg_spi(REG_FRF_MSB, 0xE4);
//    write_reg_spi(REG_FRF_MID, 0xC0);
//    write_reg_spi(REG_FRF_LSB, 0x00);
    write_reg_spi(REG_FRF_MSB, frf_msb);
    write_reg_spi(REG_FRF_MID, frf_mid);
    write_reg_spi(REG_FRF_LSB, frf_lsb);
    write_reg_spi(REG_FIFO_TX_BASE_ADDR, 0x00);
    write_reg_spi(REG_FIFO_RX_BASE_ADDR, 0x00);
    short lnaGain = read_reg_spi(REG_LNA) & 0xE0;
    write_reg_spi(REG_LNA, (lnaGain | 0x03));
    write_reg_spi(REG_MODEM_CONFIG_3, 0x04);
    write_reg_spi(REG_PA_DAC, 0x84);
    write_reg_spi(REG_OCP, 0x2B);
    write_reg_spi(REG_PA_CONFIG, 0x8F);
    write_reg_spi(REG_RX_TIMEOUT, 0xFF);
    write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_STDBY));
    delay_ms(5);
    return 0;
}

void rf_block_while_tx() {
    while(!(read_reg_spi(REG_IRQ_FLAGS)) & MASK_TX_DONE)
      delay_us(100);
    write_reg_spi(REG_IRQ_FLAGS, MASK_TX_DONE);
    return;
}

char begin_packet() {
    write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_STDBY));
    write_reg_spi(REG_MODEM_CONFIG_1, 0x72);
    write_reg_spi(REG_FIFO_ADDR_PTR, 0x00);
    write_reg_spi(REG_PAYLOAD_LENGTH, 0x00);
    return 0;
}

void add_data_to_packet(char data) {
    write_reg_spi(REG_FIFO, data);
//    *ptrPayloadLen++; // Pointer is used so the payload length is
automatically incremented by the function
//    return *ptrPayloadLen;
}

char end_packet(char payloadLen) {
    write_reg_spi(REG_PAYLOAD_LENGTH, payloadLen);
    write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_TX));
    return payloadLen;
}
```

```
char send_packet(char* payload, char payloadLen) {
    int i = 0;
    begin_packet();
//    write_reg_burst(REG_FIFO, payload, payloadLen);
    for(i = 0; i < payloadLen; i++) {
      add_data_to_packet(payload[i]);
    }
    return end_packet(payloadLen);
}

char rf_block_until_rx() {
    while(!(read_reg_spi(REG_IRQ_FLAGS) & (MASK_RX_TIMEOUT |
MASK_RX_DONE)))
      delay_us(50);
    char irqFlags = (read_reg_spi(REG_IRQ_FLAGS));
    write_reg_spi(REG_IRQ_FLAGS, (MASK_RX_TIMEOUT | MASK_RX_DONE |
MASK_PAYLOAD_CRC_ERROR | MASK_VALID_HEADER));
    return irqFlags;
}

int main() {
    unsigned char payloadLen = 0;
    unsigned char payload[64];

    // RUN_TIME DEBUGGING VARIABLES

    char tx = 1; // If tx == 1, transmitter mode enabled
    char uart_debug = 2;

    delay_ms(500);
    ANSELA = 0;
    ANSELB = 0;
    TRISBbits.TRISB15 = 0;
    LATBbits.LATB15 = 1;
    spi_init(100000);
    uart1_init(57600);
    uart2_init(57600);
    unsigned char row = '0';
    unsigned char col = 'A';
    unsigned char i = 0;
    radio_config();

    delay_ms(250);
    uart2_write('B');
    uart2_write('o');
    uart2_write('o');
    uart2_write('t');
    uart2_write('i');
    uart2_write('n');
    uart2_write('g');
    uart2_write('\r');
    uart2_write('\n');
    delay_ms(25);

    if(tx) {
      payload[0] = 'S';
      payload[1] = 't';
      payload[2] = 'a';
      payload[3] = 'r';
      payload[4] = 't';
      payload[5] = 'i';
      payload[6] = 'n';
      payload[7] = 'g';
      payload[8] = '\r';
      payload[9] = '\n';
      payloadLen = 10;
      send_packet(payload, payloadLen);
      rf_block_while_tx();
    }

    while(tx) {
      payloadLen = 0;
//    payload[0] = col++;
//    payload[1] = row++;
//    payload[2] = '\r';
//    payload[3] = '\n';
//    payloadLen = 4;
//    if(col > 'Z')
//        col = 'A';
//    if(row > '9')
//        row = '0';
      do {
          payload[payloadLen] = write_uart(uart_debug,
wait_for_read_uart(1));
          payloadLen++;
      } while(payload[payloadLen-1] != '\n'
          && payload[payloadLen-1] != '\r'
          && payloadLen < 64);
      if(payload[payloadLen-1] == '\r' && payloadLen < 63) {
          payload[payloadLen] = '\n';
          write_uart(uart_debug, '\n');
          payloadLen++;
      }


//    payload[0] = 'H';
//    payload[1] = 'e';
//    payload[2] = 'l';
//    payload[3] = 'l';
//    payload[4] = 'o';
//    payload[5] = '\n';
//    payloadLen = 6;
      send_packet(payload, payloadLen);
      rf_block_while_tx();
      for(i = 0; i < payloadLen; i++) {
          write_uart(uart_debug, payload[i]);
      }
      delay_ms(250);
    }

    while(!tx) {
      write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_RX_SINGLE));
      write_reg_spi(REG_FIFO_ADDR_PTR, 0x00);
      unsigned char irqFlags = rf_block_until_rx();
      switch(irqFlags) {
          case (MASK_RX_DONE | MASK_VALID_HEADER):
            payloadLen = read_reg_spi(REG_RX_NB_BYTES);
            read_reg_burst(REG_FIFO, payload, payloadLen);
            write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_STDBY));
//          write_uart(uart_debug, '\r');
//          write_uart(uart_debug, '\n');
            for(i = 0; i < payloadLen; i++)
                write_uart(uart_debug, payload[i]);
            break;
          case (MASK_RX_DONE | MASK_PAYLOAD_CRC_ERROR | MASK_VALID_HEADER):
            write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_STDBY));
            break;
          case (MASK_RX_TIMEOUT):
            write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_STDBY));
            break;
          default:
            write_reg_spi(REG_OP_MODE, (MODE_LONG_RANGE | MODE_STDBY));
      }
      delay_us(25);
    }

    return (EXIT_SUCCESS);
}
```

Data Sheets

Diodes Incorporated AP1509-50 5V Buck Converter Datasheet:
https://www.diodes.com/assets/Datasheets/AP1509.pdf

Microchip MIC3775 3.3V LDO Datasheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/MIC3775-750mA-microCap-Low-Voltage-Low-Dropout-Regulator-DS20006045A.pdf

Silicon Labs CP2104 USB to UART bridge Datasheet:
https://www.silabs.com/documents/public/data-sheets/cp2104.pdf

Microchip PIC32MX110F016B-I/ML microcontroller Datasheet:
https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets/PIC32MX1XX2XX283644-PIN_Datasheet_DS60001168L.pdf