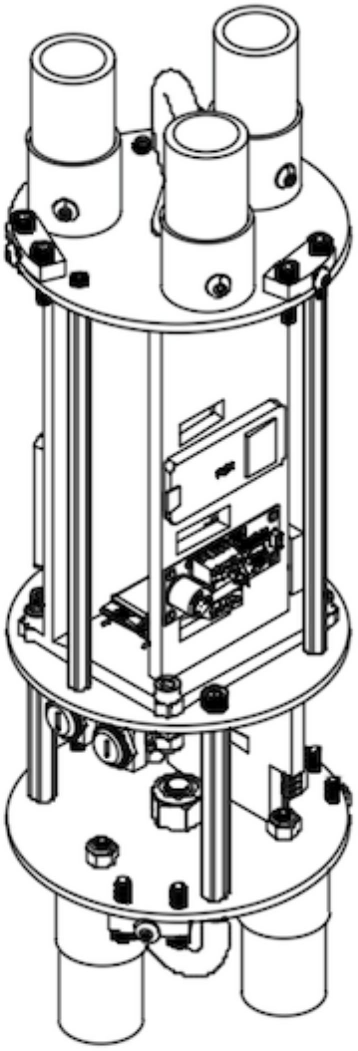
P.E.M. Operational Safety Document

Prepared by the ND Rocketry Electronics Team



February 25, 2024

Overview of P.E.M.

PIEZO IS NOT IN A RELIABLE FUNCTIONING STATE

Initialization Process

- **Battery**: If unsure about how to plug in battery, refer to a member of the Notre Dame Rocketry Electrical Team. **DO NOT UNDER ANY CIRCUMSTANCES GUESS HOW TO CONNECT THE BATTERIES**.

- **Switches**: Make sure switches are in an OFF position. Switches are in an OFF position if and only if you DO NOT HEAR the piezo. If at any point the piezo on the module is OFF while the switch is in the ON position flip the switch to OFF immediately and go into debugging. The bottom switch controls the computer battery while the top switch controls the detonation and Eggtimer battery.

- **Headers**: There are two headers located on the module, one is to the piezo one is to the explosive detonation module. The explosive one is clearly labeled as **WARNING: CHARGE BYPASS**. This header should only be on when doing a ground ejection test. This is critical as this is a safety bypass which means when it is integrated into the rocket this safety measure will be disabled. Please see safety section for more information.

- **Piezo**: Anytime anyone is around the module, the piezo should be on. Given current restraints for sub-scale PEM, this system will be quieter than usual. However it is still audible. If at any point the piezo turns on while the switch is OFF, or the switch is ON but the piezo is OFF, immediately cut power and investigate. The piezo should only turn off in flight after the LoRa check has been passed. This is safety critical, see the safety section for more information.

- **Serial Communication**: The system initiates serial communication at a baud rate of 115200. This step is crucial for enabling data transmission between the microcontroller and the computer for diagnostics and monitoring. This is only set if you are directly connecting to the module.

- **Radio Module Initialization**: The radio module is initialized to facilitate wireless communication. This is vital for receiving instructions and transmitting status updates. Its transceiver is set to these specifications:

```
#define RF95_FREQ 915.0        // LoRa Frequency (MHz)
#define TX_POWER 23            // Transmit Power (LoRa, dBm)
#define BANDWIDTH 125          // Bandwidth (LoRa, kHz)
#define SPREADING_FACTOR 12    // Spreading Factor (LoRa)
#define CODING_RATE 8          // Coding Rate (LoRa, 4/x)
#define PREAMBLE_LENGTH 12     // Preamble Length (LoRa)
```

- **Pin Configuration**: Pins are configured to specific modes (input or output) to control and monitor various components such as sensors, indicators, and actuators.

- **Altimeter Sensor Initialization**: The MPL3115A2 altimeter sensor is started. If the sensor fails to initialize, the system enters an infinite loop, signaling a critical setup error. View OPCodes to see this if unable to get to switches.

Main Operation

1. **Setting Target Altitude**: The system calculates and sets a target altitude, which is 200 feet above the current altitude measured by the altimeter. This altitude is crucial for determining when the system should activate its payload.

2. **Receiving Messages**: The system enters a loop where it continuously listens for incoming messages. Special attention is given to a specific target string (EXIT_STRING), which, when received, triggers the next phase of the operation.

3. **Continuous Pin Monitoring**: Post receiving the target string, the system frequently checks the state of the CHECK_PIN. The CHECK_PIN is the Eggtimer signal that the target set in its configuration has been reached. If this pin is in the HIGH state, the WRITE_PIN is set to HIGH, and the NeoPixel turns red, indicating the system's readiness for further action.

4. **Altitude Check**: Concurrently, the system monitors if the preset target altitude has been reached. This is a critical condition for the next step in the system's operational sequence. This is implemented on the second relay as a way to make sure that if the operational system of the CHECK_PIN fails while the system is on, it still needs to be above a certain height to activate.

5. **System Activation**: Upon satisfying both the message reception and altitude conditions, the system engages its primary function, signified by transmitting a predefined message. This step marks the completion of the system's main objectives.

Debugging Indicators with NeoPixel

- **Blue Light**: Indicates that the system is in the startup phase. This is set during the setup function.

- **Yellow Light**: Shows that the system is awaiting an incoming message. This status is set at the beginning of the main loop.

- **Green Light**: Signifies that the target message has been successfully received. This indicates a successful reception of the crucial command.

- **Red Light**: Activated when the CHECK_PIN is detected as HIGH, indicating that a specific condition has been met for system operation.

- **Purple Light**: Represents a state where the CHECK_PIN is not in the HIGH state, showing that the system is in a standby or waiting mode.

Safety Suggestions

The following safety suggestions are provided as guidelines and should be considered complementary to the directives of the program manager and the recovery lead. They are not intended to supersede any official safety protocols.

General Safety Measures

1. **Pre- and Post-Testing of the Module**: Conduct comprehensive tests of the module to ensure it activates only at the intended time. Perform functionality checks before and after ground ejection testing using a minimal resistance resistor, like 1 Ohm, as a stand-in.

2. **Proximity to the Rocket**: Avoid being in front of the rocket during the implementation phase, irrespective of switch positions or code status. Do not position any part of the body, particularly the head, near the exposed gunpowder capsules, unless necessary.

3. **Restricted Area Near Recovery Section**: Limit access to the area around the rocket's recovery section to essential personnel only. Even with stringent safety measures, unforeseen malfunctions can occur.

4. **Handling Gunpowder Charges**: When transporting gunpowder charge capsules, always aim them away from people and into an open area. Never point the capsules at individuals or objects, including oneself.

5. **Electrical Safety**: Regularly inspect all electrical connections for signs of wear, corrosion, or damage. Ensure that all wiring is properly insulated and secured to prevent accidental short circuits.

6. **Environmental Awareness**: Be cognizant of the surrounding environment during testing and integration. Avoid operating near flammable materials, in extreme temperatures, or in wet conditions to mitigate additional risks. If any components get wet or are otherwise exposed to adverse environmental conditions, do not assume they are in working order anymore and they need to be tested.

Guidelines for Integrationists

1. **Manual Integration Safety**: Strictly adhere to specialized safety guidelines during manual integration of the module. This is crucial to minimize risks associated with accidental activation or errors, which can have severe consequences.

2. **Safe Handling Practices**: Avoid placing hands near the explosive capsule during integration. Instead, focus on handling the module by its anchor points. This reduces the risk of accidental detonation, which can be catastrophic.

3. **Personal Protective Equipment (PPE)**: Always wear gloves, long-sleeve shirts, and eye protection during integration, particularly for tasks like claying. Claying involves direct contact with potentially hazardous materials and is a critical stage where utmost caution is necessary to prevent any direct exposure or mishandling.

4. **Solo Integration Work for Claying**: Conduct sensitive integration tasks such as claying without any concurrent work on the rocket. Claying is one of the most hazardous stages due to the direct handling near the explosive or sensitive components. Therefore, it is imperative that this task be performed quickly, meticulously, and, ideally, in isolation to reduce the risk of electrical shorts or inadvertent activation.

5. **Adherence to Safety Protocols**: Diligently commit to following all safety guidelines. While complete adherence may be challenging, it is essential for the immediate and long-term safety of everyone involved with the PEM system. The dangers inherent in rocket integration, particularly during claying, demand constant vigilance and strict adherence to safety protocols.

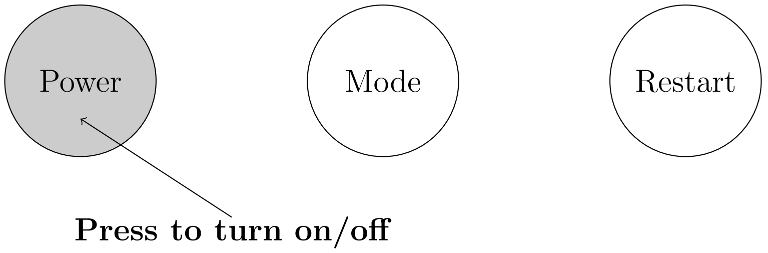
6. **Communication and Coordination**: Maintain clear and constant communication with all team members during integration and testing. It is vital that everyone is aware of their roles and the current operational status to prevent accidents due to miscommunication. No one should be in proximity to the rocket during critical stages like claying, except for those directly involved in the task. At certain stages, only one person should be working on the rocket to ensure utmost safety.

How To Activate Detonation Module

LoRa: This section details how to activate the detonator module to send a LoRa string.

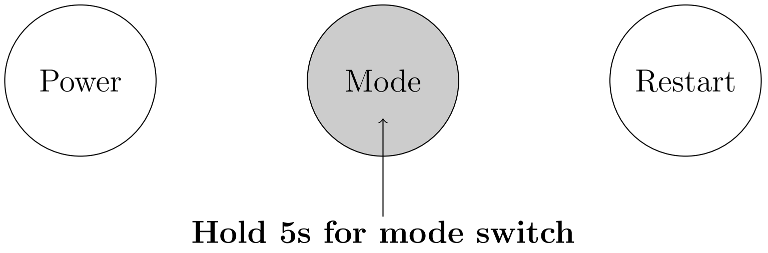
Powering On and Off

To power on the detonation module, press the 'Power' button located on the left side of the module. This button is specifically designed for toggling the power state of the module. A single press will activate the module, indicated by the activation of the OLED display and the illumination of the status LED. To power off the module, press the same 'Power' button again. The module will shut down, turning off the display and status indicators. It is crucial to ensure that the module is powered off when not in use to conserve energy and maintain safety.



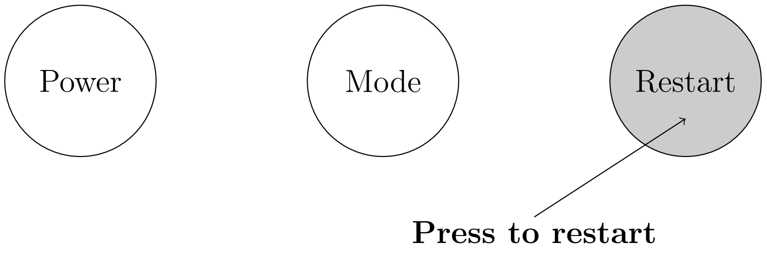
Switching Modes

The detonation module is equipped with a 'Mode' button, located in the center, to toggle between different operational states. To switch modes, hold down the 'Mode' button for at least 5 seconds. This action initiates the transition from the current mode to the next. The module typically operates in two primary modes: 'Receive' and 'Transmit'. In 'Receive' mode, the module is prepared to accept incoming signals, whereas in 'Transmit' mode, it is ready to send the LoRa string to initiate the rocketry system's separation. It's crucial to ensure that the module is in the correct mode before proceeding with any operation to avoid unintended activations. **It starts in RECIEVE mode.**



Restarting the Module

The 'Restart' button, located on the right side of the detonation module, serves as a crucial control for resetting the system. In scenarios where the module encounters an operational anomaly or when the OLED display fails to turn on, pressing the 'Restart' button effectively reboots the system. This action clears any temporary glitches and reinitializes the module's software, ensuring that the system returns to a stable state. To execute a restart, simply press the 'Restart' button. This process may be repeated as necessary until the system resumes normal function. It is a straightforward yet vital procedure for maintaining the module's reliability.



Eggtimer: This indicates how to initialize the Eggtimer module. Please refer to the manufactures PDF for further information. The password to the Eggtimer is on the recovery leads phone.

1. **Purpose of Testing**: Test the deployment channels with specific battery and igniter combinations to ensure reliable operation during flight, preventing issues like Quasar reset or igniter failure.
2. **Deployment Channel Testing Protocol**:
   (a) Always test new battery and igniter combinations before flight to confirm compatibility and operational reliability.
   (b) Safety Precaution: Do not test with live charges initially. Maintain a safe distance (recommended 30 feet) from the rocket during deployment testing.
   (c) Accessing Test Mode: Use a special URL (`192.168.4.1/test`) to access the test page, distinct from normal operational pages to prevent accidental activation.
   (d) Test Page Interface: The page displays options to select deployment channels, enter a validation code, and initiate the test.
   (e) Channel Settings: Inherits settings from Global Settings, including igniter mode, firing time, and servo deployment configurations.
   (f) Test Process: Enter validation code, click TEST, observe countdown, and deployment device is triggered.
   (g) Abort Procedure: Close the browser before countdown completion to abort the test. Check the Status Page afterward for confirmation.
3. **Caution**: Never test with just bare wire. Use a resistive load like an ematch, igniter, or fine nichrome wire. The Quasar includes safety features like current limiting, but improper testing can still cause damage.

Rocket Ejection Event Failure

In the event of a ground ejection test failure or a failure to deploy during launch, immediate and specific actions are required. UNDER NO CIRCUMSTANCES SHOULD ANYONE APPROACH THE ROCKET UNTIL YOU HAVE THOROUGHLY READ AND UNDERSTOOD THIS SECTION.

1. **Recovery Lead's Permission**: If the recovery lead has given explicit permission, a designated member of the recovery team may approach the rocket. This approach must be done strictly from the rear of the rocket. The battery switch, previously identified to all team members, should be flipped first. This action is reserved for extreme circumstances and only after confirmation that it is safe by both the recovery lead and program manager. **DO NOT, UNDER ANY CIRCUMSTANCES, ACT INDEPENDENTLY TO APPROACH THE SWITCHES. FOLLOW THE DETAILED STEPS IN THIS DOCUMENT UNLESS OTHERWISE DIRECTED BY AUTHORISED PERSONNEL.**

2. **Piezo Alarm Monitoring**: The team must immediately quiet down and listen for the piezo alarm. It is crucial that no one touches the rocket at this stage. If the piezo alarm is active, it indicates that it is safe to approach the rocket for switch operation and unintegration.

3. **OpCode Mode Activation**: If the piezo alarm is inactive or has stopped, immediately transition to OpCode mode. This involves flashing the code from the Github repository to the payload GPS receiving module. Once done, open a serial port in Arduino and prepare the OpCode table for reference.

4. **LoRa Reception Check**: Send the opcode to check if the LoRa signal was received. If it returns '1', proceed to the next step. If it returns '0', it is safe to approach the rocket.

5. **Eggtimer Pin Check**: Send the opcodes (there are two) to check the Eggtimer Control Pin and Eggtimer Activate Pin. If any of these return '1', continue to the next step. If they return '0', it is safe to approach the rocket.

6. **Altitude Activate Pin Check**: Send the opcode to check the Altitude Activate Pin. If it returns '1', proceed to the next step. If it returns '0', it is safe to approach the rocket. **Note: This is only for failure to deploy after landing.**

7. **Writing Pins to Low**: At this stage, send the opcode that writes all pins to a low state. Then, move on to the next step.

8. **Charge Status and RP2040 Restart**: At this point, the charge is live and has not detonated. No one should approach the rocket. Attempt to restart the RP2040 by sending the appropriate opcode.

9. **Repeating Steps**: Cycle through the above steps as necessary. Break out and proceed to approach the rocket only if it is deemed safe at any point during this process.

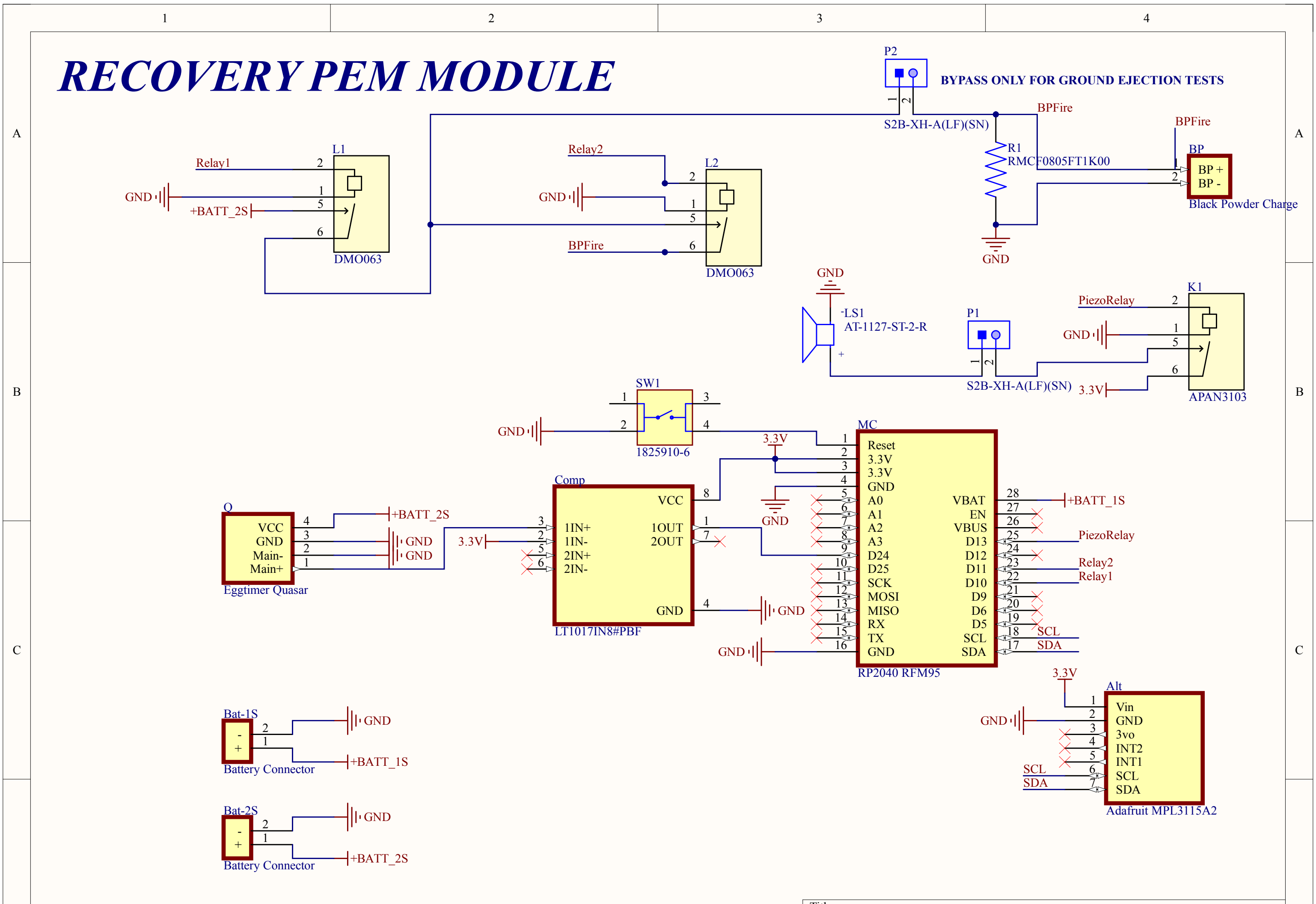
WARNING: The importance of adhering to these steps cannot be overstated. The safety of the team and the integrity of the rocket depend on strict compliance with these procedures.

OPCODE

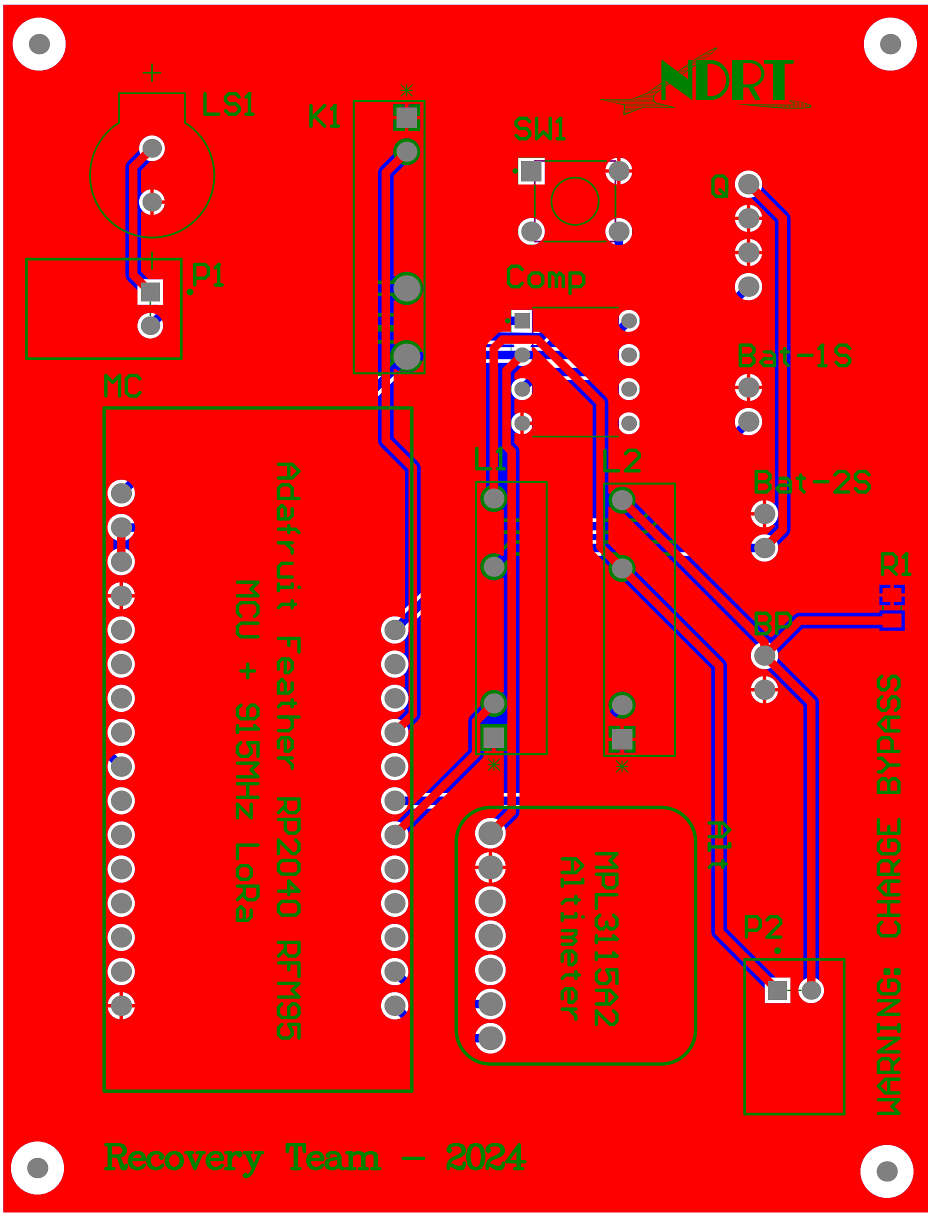
OPCODE	**Serial Input**	**Description**
0x01	OPCODE:01	Responds with the state of ’altitudeTargetSet’.
0x02	OPCODE:02	Responds with the current trigger altitude.
0x03	OPCODE:03	Checks and responds with the state of the ’Eggtimer Check Pin’.
0x04	OPCODE:04	Checks and responds with the state of the ’Eggtimer Write Pin’.
0x05	OPCODE:05	Checks and responds with the state of the ’Altimeter Write Pin’.
0x06	OPCODE:06	Responds with the LoRA received state (true/false).
0x07	OPCODE:07	Sets the ’Eggtimer Write Pin’ to LOW and confirms action.
0x08	OPCODE:08	Sets the ’Altimeter Write Pin’ to LOW and confirms action.
0x09	OPCODE:09	Increments and responds with a heartbeat count.
0x0A	OPCODE:0A	[Reserved for future use or specific implementation.]
0x0B	OPCODE:0B	Triggers a system reset of the RP2040.

Note: If you fail to receive any response either the LoRa module antenna has failed or the RP2040 has failed in some capacity. In this instance it is up to the program manager and team lead on how to move forward.

RECOVERY PEM MODULE



Title Recovery PEM Module		
Size A	Number 002	Revision
Date: 12/03/2023		Sheet of
File: C:\Users\..\RecoveryMain.SchDoc		Drawn By:


NDRT
+
LS1
K1
*
SW1
Q
P1
Comp
MC
Bat-1S
L1
L2
Bat-2S
R1
BP
Adafruit Feather RP2040 RFM95
MCU + 915MHz LoRa
MPL3115A2
Altimeter
Alt
P2
WARNING: CHARGE BYPASS
Recovery Team - 2024

Commented Code

Main

```

/*
__/\\\\\\\\\\\\\____/\\\\\\\\\\\\\\\__/\\\\____________/\\\\_
 _\/\\\/////////\\\_\/\\\///////////__\/\\\\\\________/\\\\\\_
  _\/\\\_______\/\\\_\/\\\_____________\/\\\//\\\____/\\\//\\\_
   _\/\\\\\\\\\\\\\/__\/\\\\\\\\\\\_____\/\\\\///\\\/\\\/_\/\\\_
    _\/\\\/////////____\/\\\///////______\/\\\__\///\\\/___\/\\\_
     _\/\\\_____________\/\\\_____________\/\\\____\///_____\/\\\_
      _\/\\\_____________\/\\\_____________\/\\\_____________\/\\\_
       _\/\\\_____________\/\\\\\\\\\\\\\\\_\/\\\_____________\/\\\_
        _\///______________\///////////////__\///______________\///__
*/

#include "config.h"  // Include configuration header file

Adafruit_MPL3115A2 altimeter = Adafruit_MPL3115A2();  // Initialize altimeter object

void setup() {
  Serial.begin(115200);  // Start serial communication at 115200 baud rate

  initializeRadio();  // Initialize the radio module

  // Set pin modes for various functionalities
  pinMode(CHECK_PIN, INPUT);  // Set CHECK_PIN as input
  pinMode(WRITE_PIN, OUTPUT);  // Set WRITE_PIN as output
  pinMode(ALTITUDE_READY_PIN, OUTPUT);  // Set ALTITUDE_READY_PIN as output, used to indicate altitude condition met
  pinMode(PIEZO_PIN, OUTPUT);  // Set PIEZO_PIN as output for piezo buzzer

  // Initialize NeoPixel LED
  pixels.begin();  // Initialize NeoPixel
  pixels.show();  // Turn off all pixels (initial state)

  // Initialize MPL3115A2 altimeter sensor
  if (!altimeter.begin()) {
    Serial.println("Could␣not␣find␣a␣valid␣MPL3115A2␣sensor,␣check␣wiring!");
    while (1);  // Infinite loop if sensor not found
  }
}
```

```

void loop() {
  String message;  // To store received message
  bool received = false;  // Flag to indicate message receipt
  bool continueChecking = false;  // Flag for continuous checking of CHECK_PIN
  bool altitudeTargetSet = false;  // Flag to indicate if target altitude is set
  float triggerAltitude = 0.0;  // Variable to store target altitude

  // Set target altitude if not already set
  if (!altitudeTargetSet) {
    triggerAltitude = altimeter.getAltitude() + 200.0;  // Set target altitude 200 feet above initial altitude
    altitudeTargetSet = true;  // Mark target altitude as set
    Serial.print("Target Altitude Set: ");
    Serial.println(triggerAltitude);
  }

  // Continuously check for received messages
  while (!received) {
    setNeoPixelColor(pixels.Color(255, 255, 0));  // Set NeoPixel to yellow, indicating waiting for message
    beepPiezo();  // Activate piezo buzzer
    received = checkForReceivedMessage(message);  // Check for received message
    if (received) {
      // If specific target string is received
      if (message == TARGET_STRING) {
        continueChecking = true;  // Enable continuous checking
        setNeoPixelColor(pixels.Color(0, 255, 0));  // Set NeoPixel to green, indicating target string received
        Serial.println("Target string received, starting continuous pin check.");
        break;  // Break the loop as target string is received
      } else {
        // Reset if received string is not target string
        received = false;
        message = "";
      }
    }

    checkForOpcode(received, continueChecking, altitudeTargetSet, triggerAltitude);

  }

  // Continuous checking of CHECK_PIN after receiving target message
  while (continueChecking) {
    if (digitalRead(CHECK_PIN) == HIGH) {
      digitalWrite(WRITE_PIN, HIGH);  // Set WRITE_PIN high
      setNeoPixelColor(pixels.Color(255, 0, 0));  // Set NeoPixel to red, indicating CHECK_PIN is high
      break;  // Exit loop once pin is written high
    } else {
      setNeoPixelColor(pixels.Color(128, 0, 128));  // Set NeoPixel to purple, indicating CHECK_PIN is not high
    }
      checkForOpcode(received, continueChecking, altitudeTargetSet, triggerAltitude);  // Check for opcodes
  }

  // Loop until target altitude is reached
  while (1) {
    bool altitudeReached = checkAndSetAltitude(triggerAltitude);  // Check if target altitude is reached
    if (altitudeReached) {
      checkForOpcode(received, continueChecking, altitudeTargetSet, triggerAltitude);
      Serial.println("Target␣altitude␣reached,␣exiting␣loop.");
      break;  // Exit loop once target altitude is reached
    }
  }

  setNeoPixelColor(pixels.Color(0, 0, 255));  // Set NeoPixel color to blue

  // Transmit that system has fired
  while (1) {
    checkForOpcode(received, continueChecking, altitudeTargetSet, triggerAltitude);
    transmitMessage(TRANSMIT_STRING);  // Transmit a predefined message
  }
}

void setNeoPixelColor(uint32_t color) {
  // Function to set color of NeoPixel
  for (int i = 0; i < NUMPIXELS; i++) {
    pixels.setPixelColor(i, color);  // Set color for each pixel
  }
  pixels.show();  // Apply the color change
}

/*
  NeoPixel color indications:
  - Blue: System is starting up. Set in the setup() function.
  - Yellow: System waiting for a message. Set at the beginning of the loop.
  - Green: Target message received. Set when expected message detected.
  - Red: CHECK_PIN is HIGH. Set in the continuous checking loop.
  - Purple: CHECK_PIN is LOW or not HIGH. Set in the continuous checking loop.
*/
```

Listing 1: config.h

Config

```

#ifndef CONFIG_H
#define CONFIG_H

//*******************************************//
//                Libraries                  //
//*******************************************//
#include <Arduino.h>
#include <RH_RF95.h>
#include <Adafruit_NeoPixel.h>
#include <Wire.h>
#include <Adafruit_MPL3115A2.h>

//*******************************************//
//             LoRa Settings                 //
//*******************************************//
#define RF95_FREQ 915.0                // LoRa Frequency (MHz)
#define RFM95_CS 16                    // Chip Select pin
#define RFM95_INT 21                   // Interrupt pin
#define RFM95_RST 17                   // Reset pin
#define TX_POWER 23                    // Transmit Power (LoRa, dBm)
#define BANDWIDTH 125                  // Bandwidth (LoRa, kHz)
#define SPREADING_FACTOR 12            // Spreading Factor (LoRa)
#define CODING_RATE 8                  // Coding Rate (LoRa, 4/x)
#define PREAMBLE_LENGTH 12             // Preamble Length (LoRa)
#define TARGET_STRING "EXIT_STRING"    // The string upon which to exit the loop
#define TRANSMIT_STRING "YourMessage"  // Replace with your desired default transmit message

//*******************************************//
//             Ignition Settings             //
//*******************************************//
#define CHECK_PIN 2  // Define the pin number to check
#define WRITE_PIN 10  // Define the pin number to write high
#define ALTITUDE_READY_PIN 11  // Change to a suitable pin number

//*******************************************//
//             Neopixel Settings             //
```

```
//*****************************************//
#define NUMPIXELS 1  // Number of NeoPixels
#define PIEZO_PIN 13 // Define the pin number for the piezo buzzer


//*****************************************//
//            Function Declarations          //
//*****************************************//
void initializeRadio();
bool checkForReceivedMessage(String &message);
void transmitMessage(const String &message);
bool checkAndSetAltitude(float triggerAltitude);
void checkForOpcode(bool &received, bool &continueChecking, bool &altitudeTargetSet, float &triggerAltitude);
void beepPiezo();


//*****************************************//
//                  Extern                   //
//*****************************************//
extern RH_RF95 rf95;
extern Adafruit_NeoPixel pixels;
extern Adafruit_MPL3115A2 altimeter;


#endif  // CONFIG_H
```

Listing 2: config.h

Functions

```
#include "config.h"

// Define the NeoPixel and RF95 instances in the global scope
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
RH_RF95 rf95(RFM95_CS, RFM95_INT);

void initializeRadio() {
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  // Manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);

  while (!rf95.init()) {
    Serial.println("LoRa␣failed␣to␣initialize!");
    while (1);
  }
  Serial.println("LoRa␣radio␣init␣OK!");

  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency␣failed!");
    while (1);
  }

  // Configure LoRa transmitter settings
  rf95.setTxPower(TX_POWER, false);
  rf95.setSignalBandwidth(BANDWIDTH * 1000);  // Convert kHz to Hz
  rf95.setSpreadingFactor(SPREADING_FACTOR);
  rf95.setCodingRate4(CODING_RATE);
  rf95.setPreambleLength(PREAMBLE_LENGTH);
}

bool checkForReceivedMessage(String &message) {
  if (rf95.available()) {
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);

    if (rf95.recv(buf, &len)) {
      digitalWrite(LED_BUILTIN, HIGH);
      message = String((char *)buf);
      Serial.print("Got:␣");
      Serial.println(message);
      digitalWrite(LED_BUILTIN, LOW);
      return true;
    } else {
      Serial.println("Receive␣Failed.");
    }
  }
  return false;
}


bool checkAndSetAltitude(float triggerAltitude) {
  float altitude = altimeter.getAltitude();  // Read current altitude
  Serial.print("Current␣Altitude:␣");
  Serial.println(altitude);

  if (altitude >= triggerAltitude) {
    digitalWrite(ALTITUDE_READY_PIN, HIGH);  // Set the pin high
    Serial.println("Altitude␣condition␣met,␣setting␣pin␣high.");
    return true;  // Return true when target altitude is reached
  }

  return false;  // Return false otherwise
```

```
}

void transmitMessage(const String &message) {
  Serial.print("Transmitting:␣");
  Serial.println(message);

  // Convert the message to a byte array for transmission
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  message.getBytes(buf, RH_RF95_MAX_MESSAGE_LEN);
  size_t len = message.length();

  // Send the message
  if (rf95.send(buf, len)) {
    Serial.println("Message␣sent␣successfully");
  } else {
    Serial.println("Message␣failed␣to␣send");
  }
}

// Pizeo Checking

void beepPiezo() {
  digitalWrite(PIEZO_PIN, HIGH); // Turn piezo on
  delay(10); // Delay for 100 milliseconds
  digitalWrite(PIEZO_PIN, LOW); // Turn piezo off
}

// OP Code Checking, Will Return Value
void checkForOpcode(bool &received, bool &continueChecking, bool
   &altitudeTargetSet, float &triggerAltitude) {
    if (rf95.available()) {
        uint8_t buffer[RH_RF95_MAX_MESSAGE_LEN];
        uint8_t length = sizeof(buffer);

        if (rf95.recv(buffer, &length)) {
            String message = String((char *)buffer);

            if (message.startsWith("OPCODE:")) {
                uint8_t opcode = strtol(message.substring(7).
                   c_str(), NULL, 16);

                switch (opcode) {
                    case 0x01:
                        transmitMessage("altitudeTargetSet:␣" +
                           String(altitudeTargetSet));
                        break;
                    case 0x02:
                        transmitMessage("Trigger␣Altitude:␣" +
                           String(triggerAltitude));
                        break;
                    case 0x03:
                        transmitMessage("Eggtimer␣Check␣Pin␣State:␣" + String(digitalRead(CHECK_PIN)));
                        break;
                    case 0x04:
                        transmitMessage("Eggtimer␣Write␣Pin␣State:␣" + String(digitalRead(WRITE_PIN)));
                        break;
                    case 0x05:
                        transmitMessage("Altimeter␣Write␣Pin␣State:␣" + String(digitalRead(ALTITUDE_READY_PIN)));
                        break;
                    case 0x06:
                        transmitMessage("Received:␣" + String(received));
                        break;
                    case 0x07:
                        digitalWrite(WRITE_PIN, LOW);
                        transmitMessage("Eggtimer␣Write␣Pin␣Low");
                        break;
                    case 0x08:
                        digitalWrite(ALTITUDE_READY_PIN, LOW);
                        transmitMessage("Altimeter␣Write␣Pin␣Low");
                        break;
                    case 0x09:
                        static int heartbeatCount = 0;
                        heartbeatCount++;
                        transmitMessage("Heartbeat:␣" + String(heartbeatCount));
                        break;
                    case 0x0A:
                        // Implementation for opcode 0x0A
                        break;
                    case 0x0B:
                        // Implementation for resetting RP2040
                        rp2040.reboot(); // This will reset the RP2040
                        break;

                }
            }
        }
    }
}
```

Listing 3: functions.cpp