MONTANA SPACE GRANT CONSORTIUM B.O.R.E.A.L.I.S.

# O.C.C.A.M.S.

## Optimized Command and Control Aerial Management System

**Last Edit: 12/15/2022**

Documentation for OCCAMS v9 or later and software versions 1-3 or later. Notes and troubleshooting will apply to previous versions but use your discretion. Differences in board design can be seen in each respective design section.

# *Table of Contents*

# *Introduction and Overview*

This is to serve as an operational and troubleshooting guide for the Optimized Command and Control Aerial Management System (O.C.C.A.M.S.) developed for use by Montana Space Grant Consortium BOREALIS high altitude ballooning program. OCCAMS is a programmable tracking and termination system aimed to provide reliable, versatile, and wireless vent control and termination options for both latex and zero pressure balloon flights. OCCAMS is designed to operate as both a master and a slave device in the sense that it can broadcast the commands from Iridium wirelessly or receive broadcasted commands. Figure 1 shows the basic system architecture.

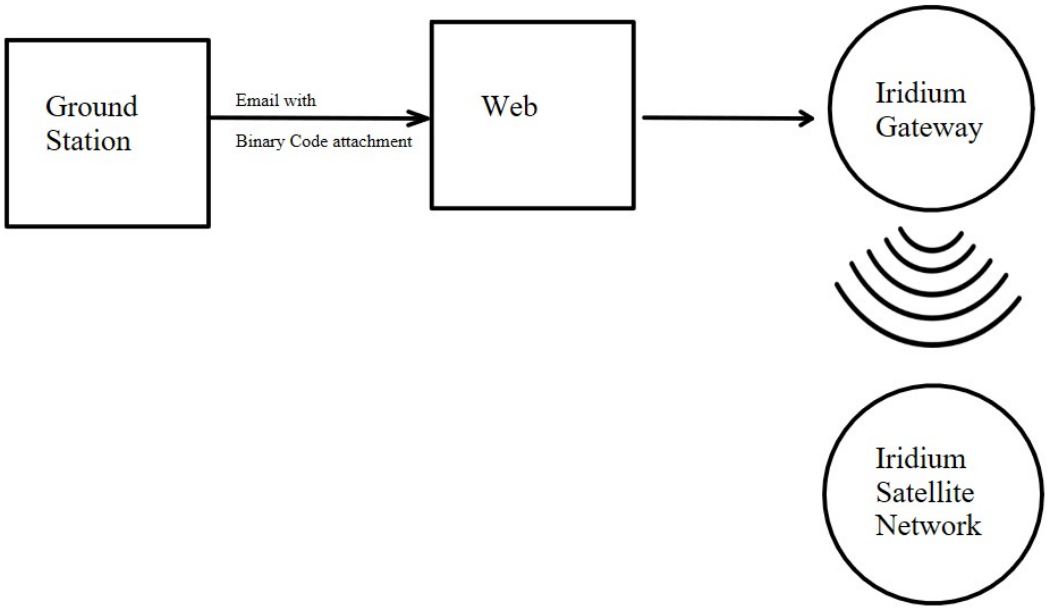



**Figure 0 – Ground station to Iridium Gateway model**

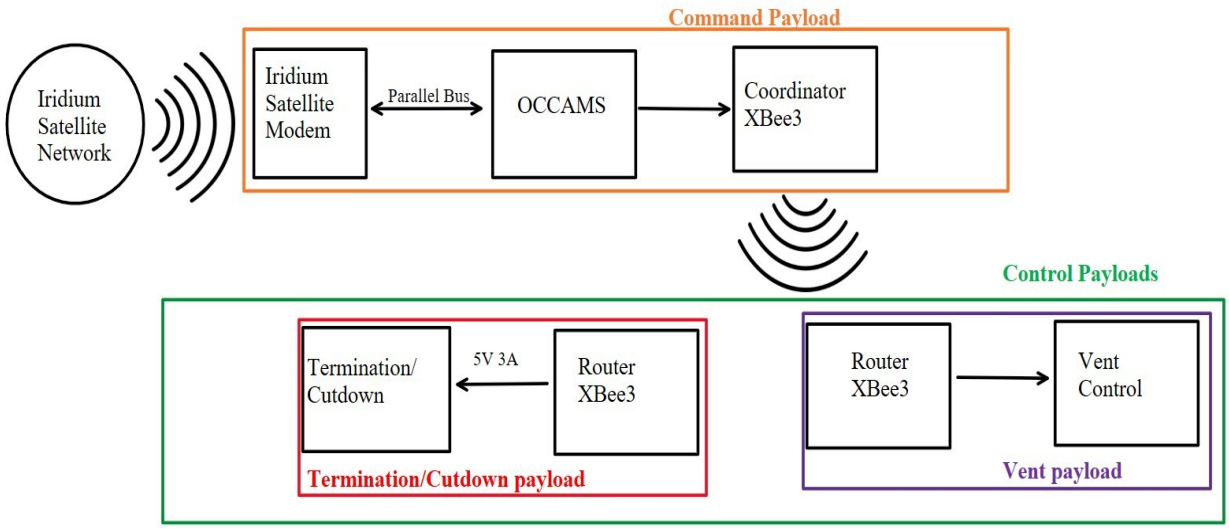


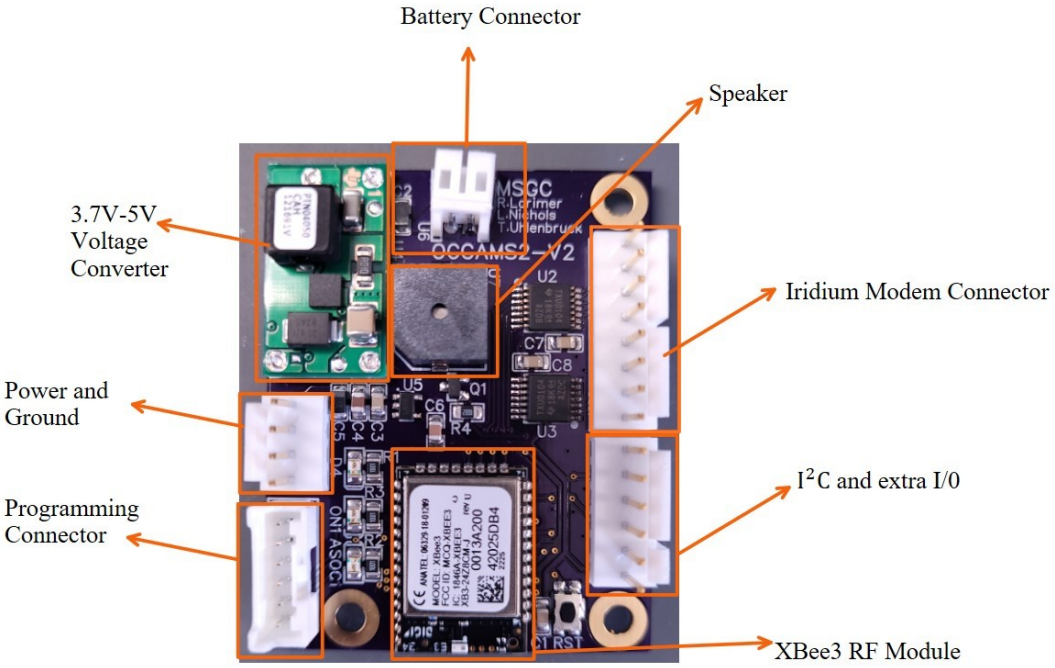

**Figure 1 – OCCAMS intended applications and system architecture**

Depicted in Figure 0 is the Iridium satellite network receiving an attachment that is emailed through ground station. From there, the Iridium satellite network sends that attachment's binary code, from zero to seven, to the Iridium modem on the payload. The Iridium modem is connected to the OCCAMS board, the modem then sends the binary code to the OCCAMS board. On the OCCAMS board is a radio frequency module named the XBee3, it is what receives the binary code through the OCCAMS board, the Coordinator XBee3 then sends that same binary code out to anything on the network with the same Pan ID that it is on, it sends this same code over and over until a new code is received. Router XBee3's that are on the same Pan ID as the Coordinator then receive whatever binary code is sent by the Coordinator Xbee3. The Router XBee3's then initialize whatever they are coded to do once that code is received. For example, if the Termination/Cutdown payload's XBee3 receives 001 as the binary code, it then runs the code that allows the current from the battery to flow through the nichrome wire and heat it up to cut the payload string. The Vent payload's XBee3 sees the same binary code, however it does not have any code that initializes on the code 001 so it stays idle. The binary code changes from 001 to 011, the Termination/Cutdown has no code that reacts to 011 so it doesn't do anything, but the Vent payload does, so it initializes the code it has.

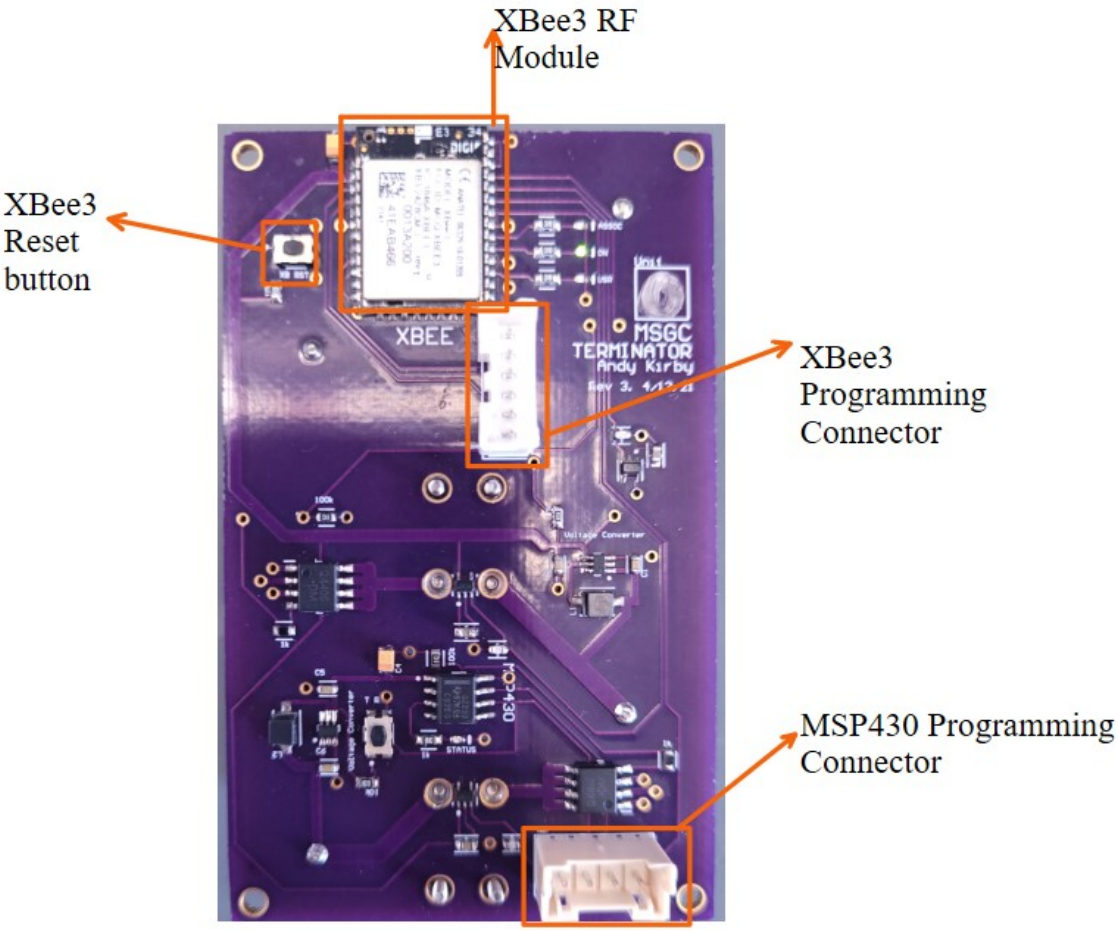
# *Hardware*

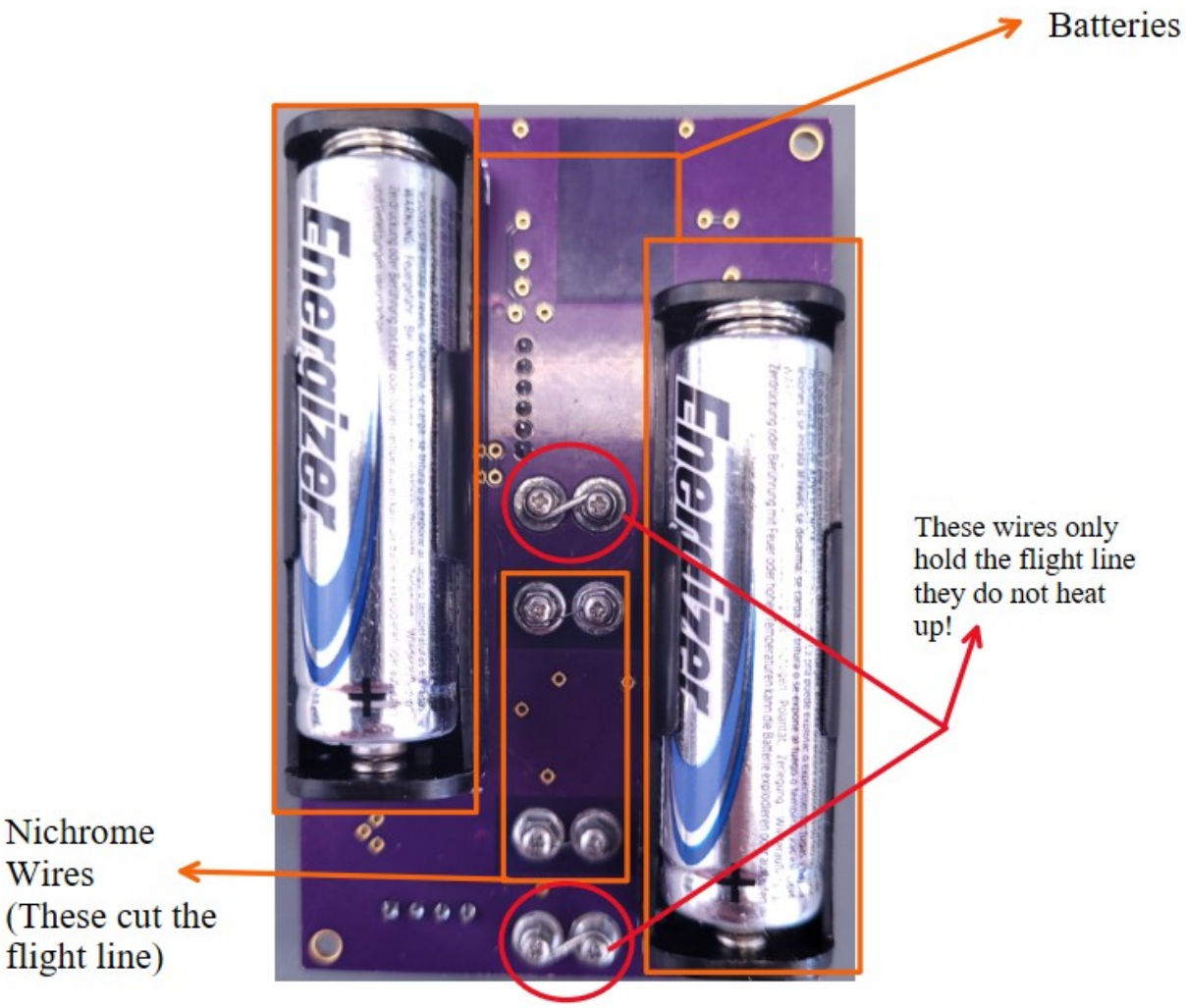
This section covers the full printed circuit board operation from a hardware perspective as well as the external wiring connections OCCAMS uses for testing and flights. Wiring for programming is covered in the software section.

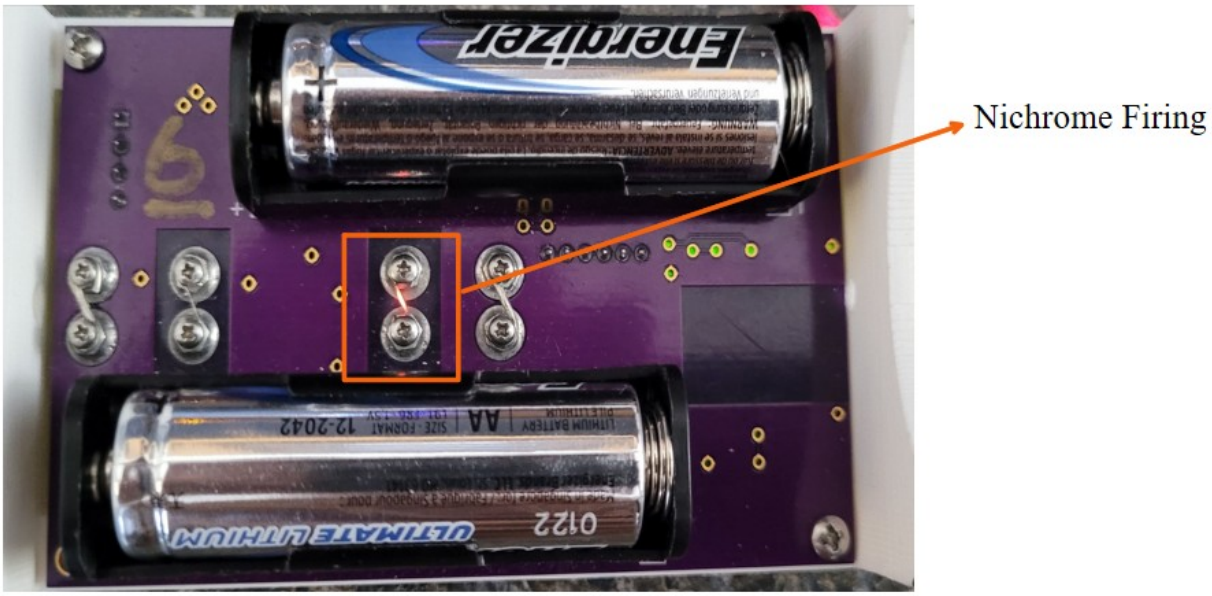
## *OCCAMS*



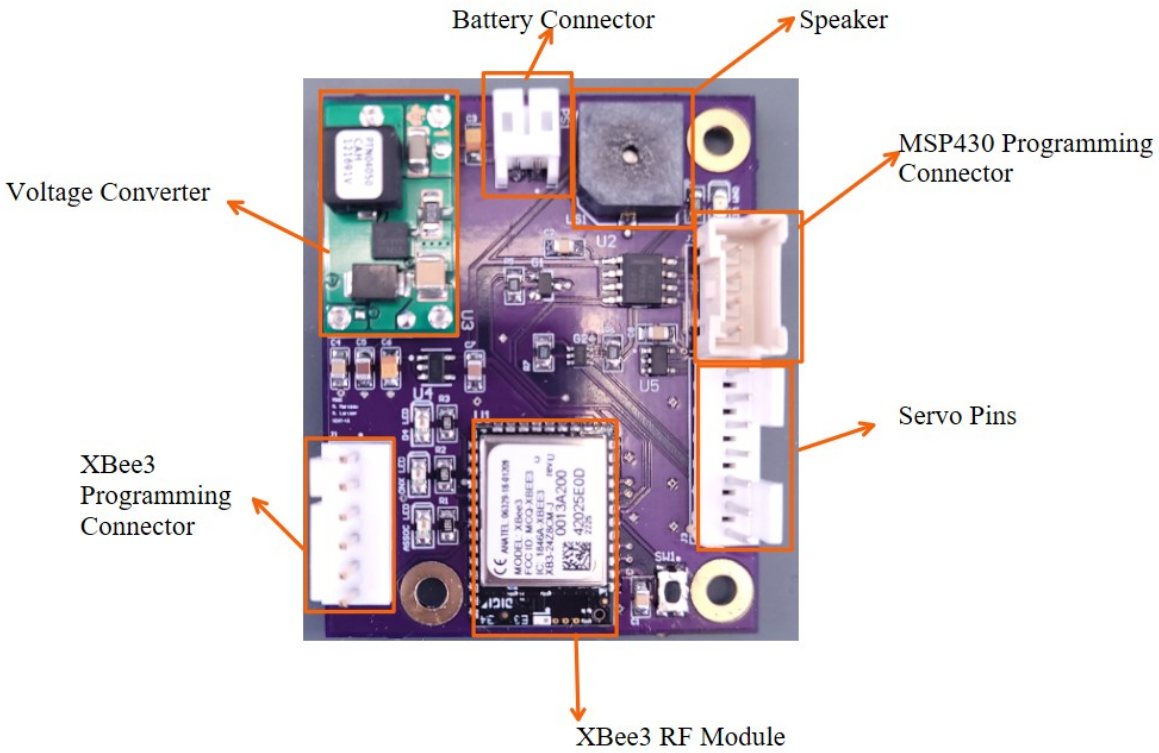

## *Cutdown*


XBee3 RF Module
XBee3 Reset button
XBee3 Programming Connector
MSP430 Programming Connector
MSGC
TERMINATOR
Andy Kirby


Batteries
These wires only hold the flight line they do not heat up!
Nichrome Wires (These cut the flight line)


Nichrome Firing

***Vent***


Battery Connector
Speaker
MSP430 Programming Connector
Voltage Converter
Servo Pins
XBee3 Programming Connector
XBee3 RF Module

## *Hardware - Normal Operation*

As far as normal operation goes for the OCCAMS board, it is fairly simple. Once you provide power to the board the ON LED should stay on. Notice the LED labeled D4 is blinking the same time as the heartbeat sounds, this is another redundant form of the heartbeat in case the speaker stops working.

For the Cutdown, it roughly the same idea. Check that the ON LED remains on while the batteries are in. The STATUS LED should be blinking about every second. The ASSOC LED will only turn on, then stay on, once it has received the binary code to terminate the flight.

During a normal flight, there is a setup procedure to reset the timer module running on the board. The standard launch day sequence is as follows

**OCCAMS/Control Payload:**

1. Connect 8 pin Molex to the iridium modem.
2. Connect the Iridium modem's power to OCCAMS through 4 pin Molex.
3. Connect the battery.
4. Listen for audible "heartbeat" from speaker. If you hear this it is operating properly.
5. Your Control Payload is ready for flight!

**Termination/Cutdown:**

**1.** Insert batteries.

2. Check the ON and STATUS lights, ON should be solid, Status should blink about every second. The ASSOC should **NOT** be on, if it is the nichrome has fired and you need to send the idle command through the Iridium network to OCCAMS.

**3.** Check if the Nichrome wire is firing (it will be glowing orange every second). IF it is firing immediately after it turns on it means that the cutdown code is being sent still and your Control payload needs to have the idle code before flight.

4. Thread your flight line **THROUGH** the thicker outer wires and **OVER** the nichrome wires, the outer wires hold the flight line down so the nichrome wires can heat and cut the flight line.
5. Your cutdown is ready for flight!

# *Hardware – Setup*

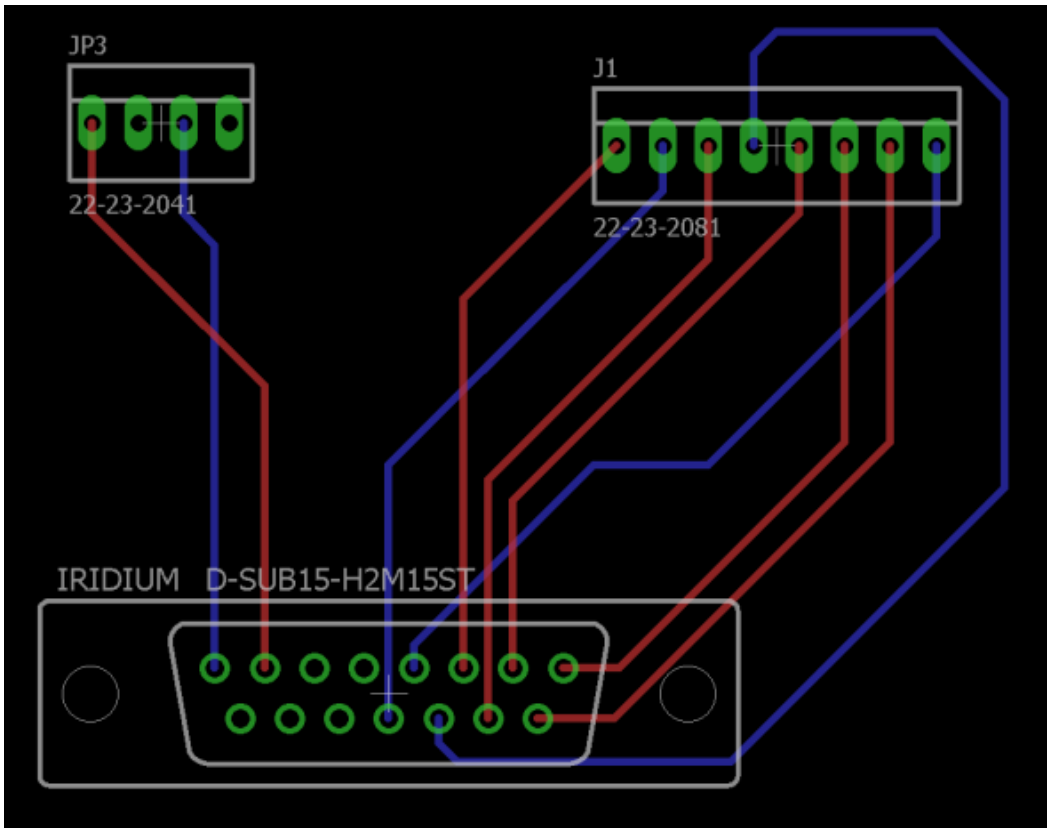
## *Iridium Connection*

There are some cable assemblies associated with both cutdown and the iridium connection. Below in Figure 2 shows the 15 pin D connectors wiring in respect to the OCCAMS board. Table 1 shows the absolute connection between iridium pins and OCCAMS pins.

**Table 1 – OCCAMS to Iridium pinout and pin descriptions**

| Pin | OCCAMS | Wire Color | IRIDIUM | PURPOSE |
|---|---|---|---|---|
| 1 | +5V | Red | External Power Input (3.6V to 5.3V) | Supply Power |
| 2 | GND (4-pin) | Black | External Power Input (GND) | Supply Power |
| 3 | NC* | - | RS232 Input | Programming Modem |
| 4 | NC* | - | RS232 Output | Programming Modem |
| 5 | GND (8-pin) | Black | Signal Ground, 0V signal reference | Ground reference for TTL |
| 6 | A0 | Green | External TTL/CMOS INPUT S0 | Status bit going to Iridium |
| 7 | D2 | Blue | TTL/CMOS Output 0 | Command bit from Iridium |
| 8 | D3 | Orange | TTL/CMOS Output 1 | Command bit from Iridium |
| 9 | NC* | - | External Power Input (6V to 32V) | Supply Power |
| 10 | NC* | - | Reserved | None |
| 11 | NC* | - | Reserved | None |
| 12 | A1 | Yellow | External TTL/CMOS INPUT S1 | Status bit going to Iridium |
| 13 | A3 | Grey | External TTL/CMOS INPUT S3 | Status bit going to Iridium |
| 14 | A2 | Blue | External TTL/CMOS INPUT S2 | Status bit going to Iridium |
| 15 | D4 | Green | TTL/CMOS Output 2 | Command bit from Iridium |

*NC = No Connect (not wired to OCCAMS)


JP3
22-23-2041
J1
22-23-2081
IRIDIUM D-SUB15-H2M15ST

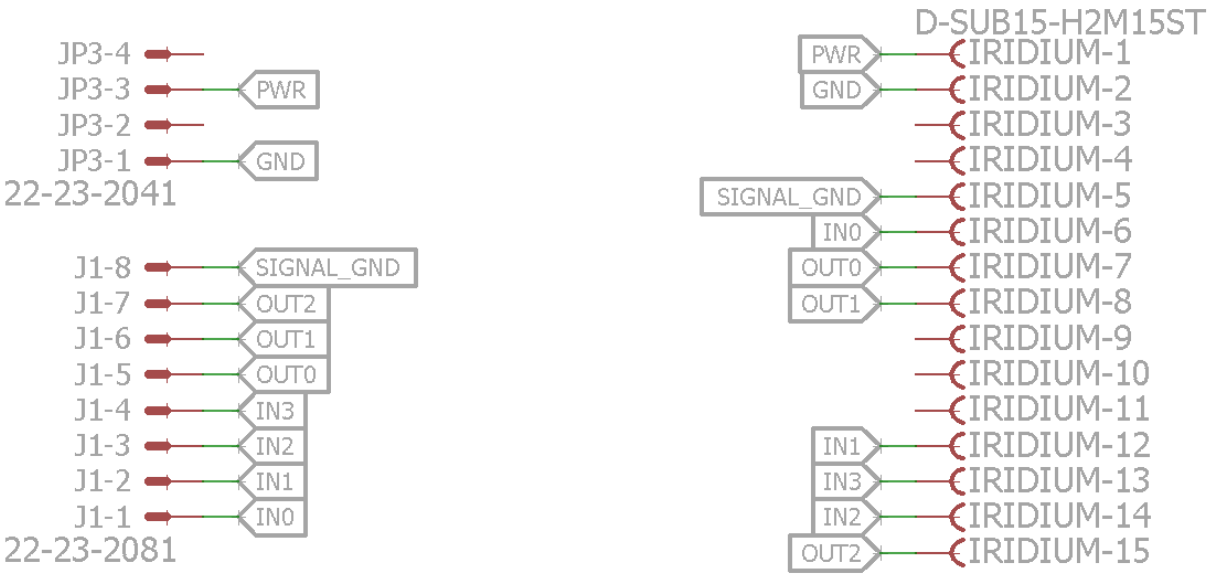



**Figure 2 – Iridium wiring harness layout to OCCAMS. The left connector drawing is the male end on the Iridum to PCB pads.**

The above wiring diagram in Figure 4 is to be used as reference but use caution. It would be very easy to mistakenly reverse a Molex connector or the D connector wiring. The preference is to use Table 1 for any wiring, referencing the OCCAMS hardware design to ensure correct wiring.

## *Termination*

The termination wiring for this system consists of a +5V 2.4A supply, a data line capable of PWM operation, a ground, and a MOSFET suspended ground. Below, in Figure 5, shows the wiring configuration for termination.

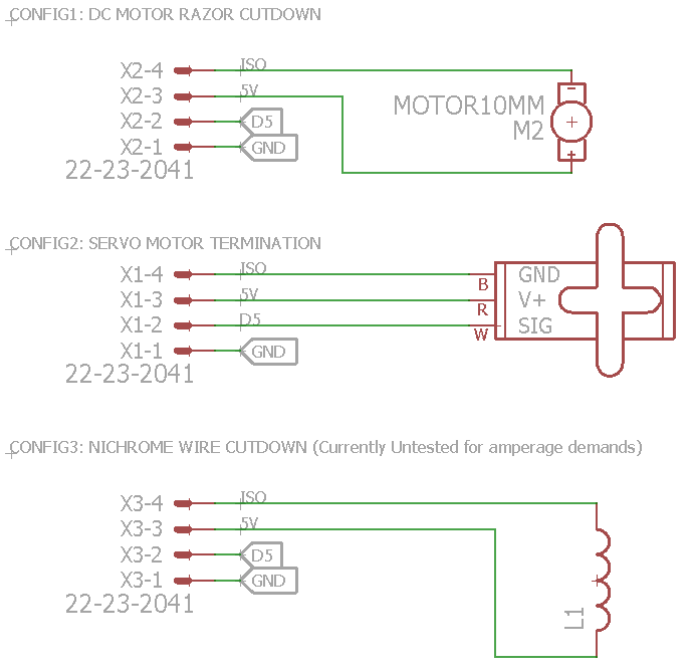



**Figure 3 – Termination Options and their respective wiring**

MSGC at Montana State University has tested and prefers using the Nichrome wire for termination. We have experimented with nichrome wire, 36 gauge round wire. Using three strands of 36 gauge nichrome twisted together at a length of 2 inches, the cold resistance measured at 1.96 ohms and the wire drew approximately 2 amps. We were able to use this wire multiple times in lab conditions to cut an 80 lbs. twisted nylon string. There are only 2.4 amps of available current, 10mA of which are used by the microprocessor and 150mA peak by the XBee wireless module. This leaves only 2.24 amps as the max draw current for the termination system. If the system overdraws in current, the batteries protection circuit kicks in and shuts the battery off. The battery is turned back on by disconnecting and

reconnecting the battery. MSGC at Montana State University has tested CONFIG 1 using a DC motor to spin a fabric blade against the suspension line consuming 20mA while on.

## *Hardware - Design*

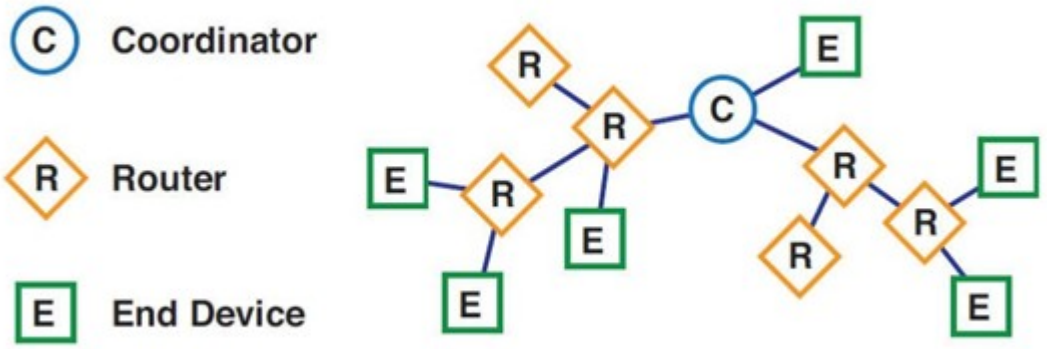
# *Software*

## *Software – uploading code onto XBee3*

1. Download and install PyCharm
2. Start PyCharm and click plugins
3. Search “Digi Xbee” and install, then search “MicroPython” and install.
4. Restart PyCharm IDE
5. Select “New Digi Project”, select next
6. Select “Create Digi Project”, select next
7. Select “XBee Module”, select next
8. Select “Digi XBee3 Zigbee3”, select next
9. Don’t import any libraries unless you specifically need to. If you don’t know, you probably don’t need to import any. Select next.
10. Keep the default settings for the project, but feel free to change directory and name of project.
11. Select create
12. Delete the main.py file that is automatically added, that just prints “Hello World”
13. PyCharm is weird and does not have an easy “add existing file to project” option, instead right click the project>New>Micropython File. Name the file whatever you want.
14. Simply dragging the file you need into PyCharm just allows you to view and edit the code in PyCharm, but does not add it to the project, which needs to happen.
15. Copy and paste the Occams dispatcher Code into the new Micropython file you have just made
16. Select the “Run Project” in the top right corner.
17. In the XBee Device Selector Screen: Select Local PC>Select the XBEE that shows up>select “OK”
18. The code will delete whatever was previously on the board, upload, flash, then run the code loaded onto it.
19. Congratulations you have uploaded code onto the Xbee3!

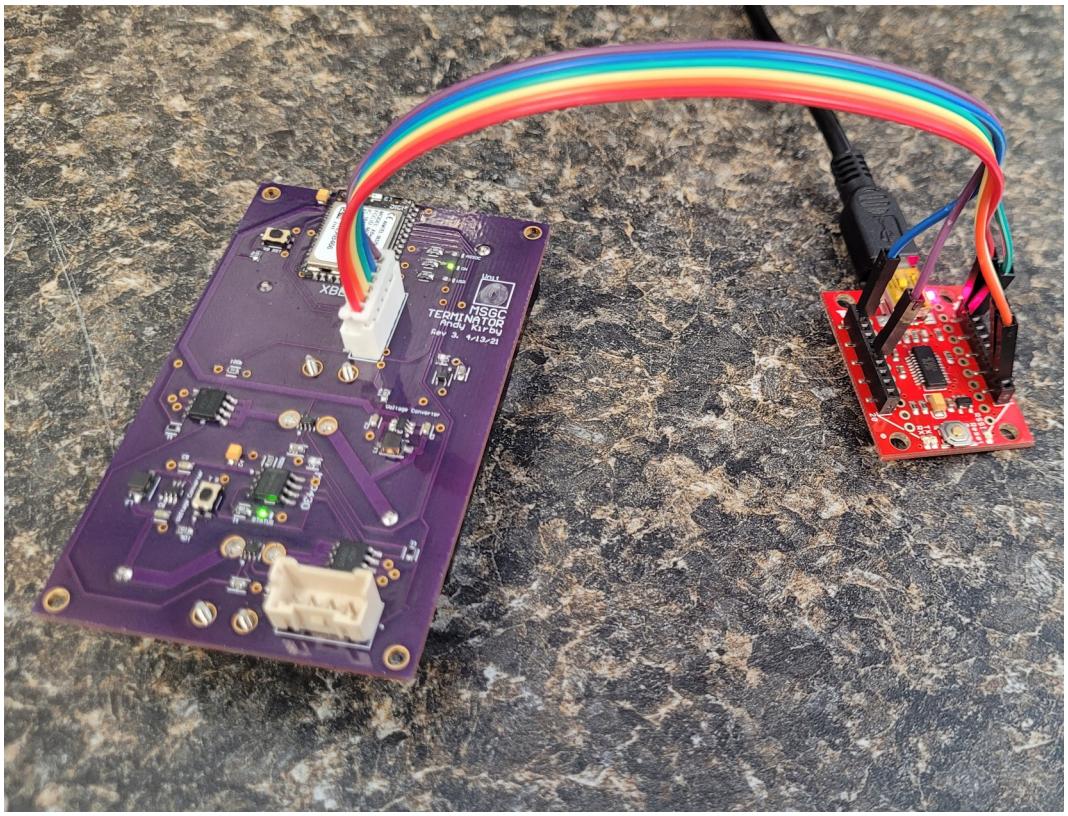
### *XBee Wireless Modules*

This tutorial will walk you through the process of setting up your XBee modules to communicate in pairs or as a mesh network. Xbees need one coordinator per mesh network and no more. The coordinator will be the OCCAMS command module and both the terminator/cutdown and the vent/valve will be router devices. “End devices” have been phased out of current XBee implementation and now it is just a Coordinator/Router relationship, however below is an image that helps model how the mesh network is set up and how it works.

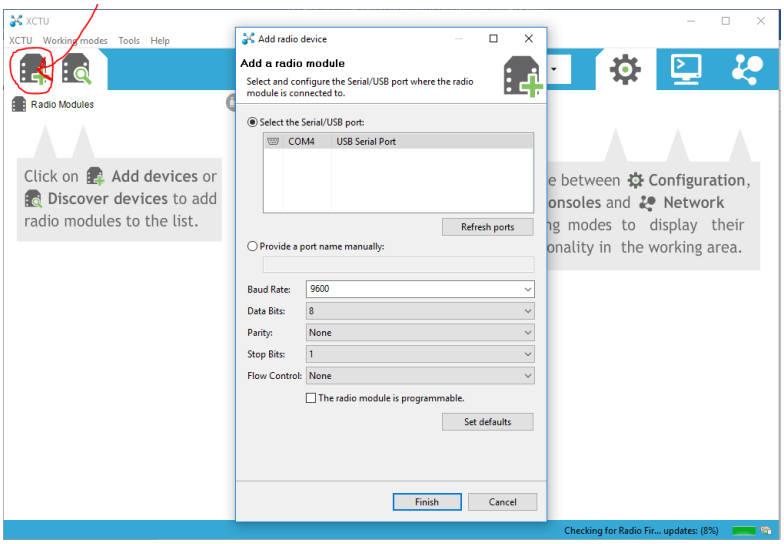



Alternative tutorial: https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu

1. To program the Xbee for our application, we need an FTDI cable, an XBee radio module, and a 5V to 3V adapter board pictured below.



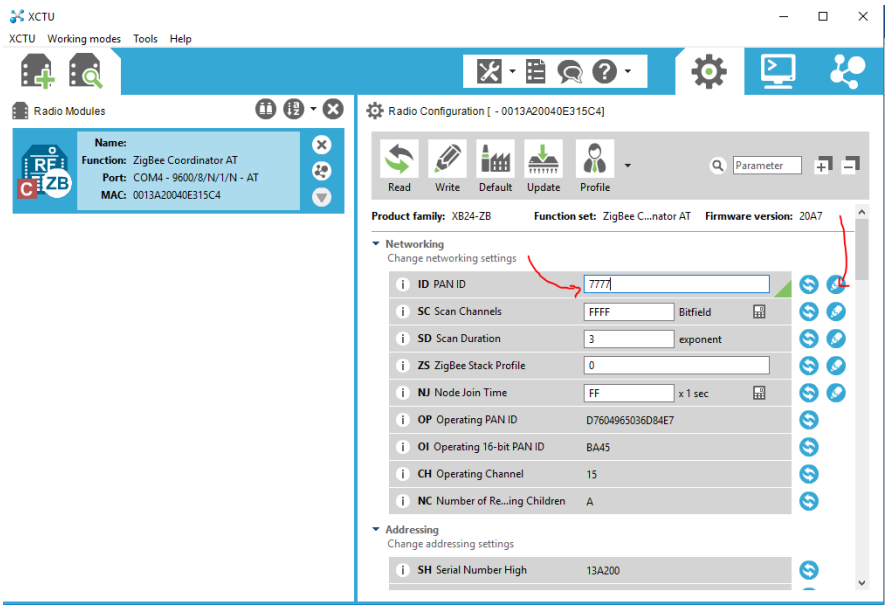
2. The FTDI cable will require a driver if you have never used one before. The driver can be installed via an executable: http://www.ftdichip.com/Drivers/CDM/CDM21216_Setup.exe
   Or you can do a full tutorial walk through here: https://learn.sparkfun.com/tutorials/how-to-install-ftdi-drivers?_ga=1.123561099.1200671417.1455319667
3. Next we need XCTU to edit the settings, so open XCTU. If it is not already installed, you can get the program here: http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu
4. Opening XCTU from Digi, select the "Add a Radio Module" button in the top left:




5. If you're FTDI connection is seen, you can see that in the Serial USB port list. COM4 is my FTDI cable connected XBee module. Select this port, and assuming factory settings of: 9600, 8, none, 1, none as shown above and select finish. This will discover the XBee module on this port and download its firmware settings. You may have to do a power cycle of the module during this process but this will be indicated through a popup in the XCTU software. To power cycle, unplug the adapter board from the FTDI cable and plug it back in. If this doesn't find your device, try the Discover radio modules option to the right of add radio modules and search you selected COM ports with factory settings.

6. Once the module is discovered, you will see the module on the left of the screen, double click it to pull up the full device settings.
7. If we are configuring the device to be a Coordinator, for the wireless module for the OCCAMS connected to the Iridium module, or a Router, in the case of a termination wireless module, we need to change the Device Role. If the XBee module is supposed to be a Coordinator (OCCAMS) the Device Role needs to be set to Form Network. If the XBee module is supposed to be a Router (Termination/Cutdown and/or Vent) the Device Role needs to be set to Join Network.
8. Now the module is correctly in the Coordinator mode we need it to be in for the payload control OCCAMS. However you will notice that the module's PAN ID is 0 there are some features the module provides.
    a. If a coordinator is configured with PAN ID 0, it performs a PAN scan to identify nearby ZigBee networks and uses a random unused PAN ID to start the new network.
    b. If a router or end device is configured with PAN ID 0, it performs a PAN scan and tries to join to the first ZigBee network it finds.

    Because of these features, it is required that you change the PAN ID to something that is unique and consistent for your network to avoid complications. To change the Pan ID, type in an ID between 0x0000 and 0xFFFF (hexadecimal) and write the PAN ID to the module. This same procedure can be done for any setting on the module. The write button is the pencil next to the setting. All modules on a network must have the same PAN ID.




It is recommended that you read the setting back after writing using the refresh button.

9. The module is now configured!

## *Software – Design*

The coding structure is as follows:

- Imports
- Pin Definitions
- Function definitions
- Main loop

## *OCCAMS Code*

```
# Imports
import xbee
import machine
from machine import Pin
import time

commands = ["ABC", "DEF", "GHI", "JKL", "MNO", "PQR", "STU", "VWX"]

# Pin Definitions
usr_led = Pin(machine.Pin.board.D4, Pin.OUT, value=0)
asc_led = Pin(machine.Pin.board.D5, Pin.OUT, value=0)
spk = Pin(machine.Pin.board.D10, Pin.OUT, value=0)

signal0 = Pin(machine.Pin.board.D0, Pin.IN)
signal1 = Pin(machine.Pin.board.D2, Pin.IN)
signal2 = Pin(machine.Pin.board.D3, Pin.IN)

out0 = Pin(machine.Pin.board.D15, Pin.OUT, value=0) # Reserved for signaling cutdown states
out1 = Pin(machine.Pin.board.D16, Pin.OUT, value=0) # Reserved for signaling cutdown states
out2 = Pin(machine.Pin.board.D17, Pin.OUT, value=0)
out3 = Pin(machine.Pin.board.D18, Pin.OUT, value=0)


cutDownStatus = 0
auxStatus = 0
mspSpike = False
```

Shown above are the imports and pin definitions. It is important that you import xbee, machine, and time. The xbee import allows you to interface with the XBee3, machine allows the user to define and use the pins on the board, and time allows the user to set specific time values for any functions.

```
def resolveSettingCutdownSignal():
    #Set the output pins based on the status
    out0.value(cutDownStatus % 2)
    out1.value(int(cutDownStatus/2) % 2)
    out2.value(auxStatus % 2)
    out3.value(int(auxStatus / 2) % 2)

def checkAck():
    global cutDownStatus
    global auxStatus
    global mspSpike
    try:
        packet = xbee.receive()                         #Try to recieve packet
        [xbee.receive() for i in range(100)]            #Clear the buffer
        reply = packet.get('payload').decode('utf-8')[:3]
        if reply is not None:
            print(reply)
            if reply == "CAK" and cutDownStatus == 1:  # If cutdown was sent
                cutDownStatus = 2  # Set to ack received status
            if reply == "TXB":  # If the XBee temp has spiked
                cutDownStatus = 3  # Set to ack received status
            if reply == "TMS":  # If the MSP temp has spiked
                cutDownStatus = 3  # Set to ack received status
                mspSpike = True
            if reply == "VOA":
                auxStatus = 1
            if reply == "VCR":
                auxStatus = 2
    except:
        pass
```

Above are the functions we define to use later in the Main loop. The resolveDispatch function concatenates the values from three pins connecting the Iridium modem to the OCCAMS board to make the binary code. It then checks the

```
# Main Loop
while True:
    resolveDispatch()
    checkAck()
    resolveSettingCutdownSignal()
    print(cutDownStatus)



    # Blink and wait
    beep()
    usr_led.value(1)
    time.sleep_ms(1000)
    usr_led.value(0)
    time.sleep_ms(1000)
```

Shown above is the Main loop using functions that were defined previously in the code to be used here.

## *Termination/ Cutdown Code*

```
# Imports
import xbee
import machine
from machine import Pin
import time

# Set up ADCs
pri_cdn_flag = True

# Set up pins
usr_led = Pin(machine.Pin.board.D4, Pin.OUT, value=0) #Enable pin on XBEE3 set to low initially
asc_led = Pin(machine.Pin.board.D5, Pin.OUT, value=0) #Enable pin on XBEE3 set to low initially
pri_cdn = Pin(machine.Pin.board.D0, Pin.OUT, value=0) #Enable pin on XBEE3 set to low initially
pwm_pin = Pin(machine.Pin.board.P0, Pin.OUT, value=0) #Enable pin on XBEE3 set to low initially

print("Simple XBEE3 Flight Termination Unit")
```

```
# Get command
def getCmd():
    packet = None  # Set packet to none
    while packet is None:  # While no packet has come in
        packet = xbee.receive()  # Try to receive packet
    [xbee.receive() for i in range(100)]  # Clear the buffer
    return packet.get('payload').decode('utf-8')[:3]  # Return the last three charters decoded

# Process commands
def processCmd(command):
    print(command)
    if command == 'DEF':  # If command is the cutdown command
        priCutdown()  # Cutdown
    elif command == 'ABC':  # If it is idle
        idle()  # Idle

def priCutdown():
    global pri_cdn_flag  # Get global variable

    if pri_cdn_flag:  # If cutdown is ready to happen
        pri_cdn_flag = False  # Make sure it can't happen again without reset
        for i in range(10):  # Cut down for 10 seconds
            pri_cdn.value(1)
            asc_led.value(1)
            time.sleep_ms(700)  # Time on
            pri_cdn.value(0)
            time.sleep_ms(300)  # Time off
```

```
# idle
def idle():
    global pri_cdn_flag  # Get global variable
    pri_cdn_flag = True  # set flag so cutdown can happen again
    asc_led.value(0)

# Blink the LED
def blink_led():
    usr_led.value(1)
    time.sleep_ms(100)
    usr_led.value(0)


# Main loop
while True:
    processCmd(getCmd())  # Get and process the command
    blink_led()  # Blink LED
```

# *Testing*

## *Procedure*

## *Results*

# *Troubleshooting and Errors*

This section will address indicators of atypical operation and how to best go about troubleshooting.