Andy Dequin
DequinAndy@JohnDeere.com

Wireless Drawbar Pin Telemetry System Documentation

Cost-effective alternative to TECAT and ATI telemetry

JOHN DEERE WATERLOO WORKS

2024 Intern - Product Engineering

July-August 2024

Author: Andy Dequin (DequinAndy@JohnDeere.com)

Andy Dequin
DequinAndy@JohnDeere.com

Table of Contents

Andy Dequin
DequinAndy@JohnDeere.com

Introduction

Project Overview

This is a general telemetry system that was developed using consumer electronics as a cheaper and more convenient alternative to existing equipment used for load surveys. Specifically, measuring tractor implement loads through the drawbar pin[1]. The overall system takes in the strain gauge bridges on the pin as input, and outputs a differential CAN (type of communication protocol) signal into the main controller in the tractor's cabin.

The system contains of two parts: a transmitter, and a receiver.

The transmitter consists of an amplifier that can read the strain gauge signals (mV scale) and a microcontroller, to decode the amplifier data and to transmit the signals.

The receiver that consists of a microcontroller with wireless capabilities and a CAN module for converting the microcontroller signals into a CAN signal that is compatible with the tractor's CAN network.

Objectives and Goals

The project addresses the issue of expensive load measurement equipment being used in the field and not being returned, as well as the complexity of installation and use of the equipment.

Specifically, the current telemetry systems are from TECAT (~6000 USD), and ATI (>10 000 USD).

When asking about what aspects of the current systems I could maybe improve upon, I was told something like "anything would be better than TECAT" (from anonymous technician).

The goal of this project would be to create a system that can achieve a similar result, but using <100 USD worth of parts, with a relatively straightforward assembly, installation, use, and troubleshooting.

[1] Check glossary for image

How to Make It

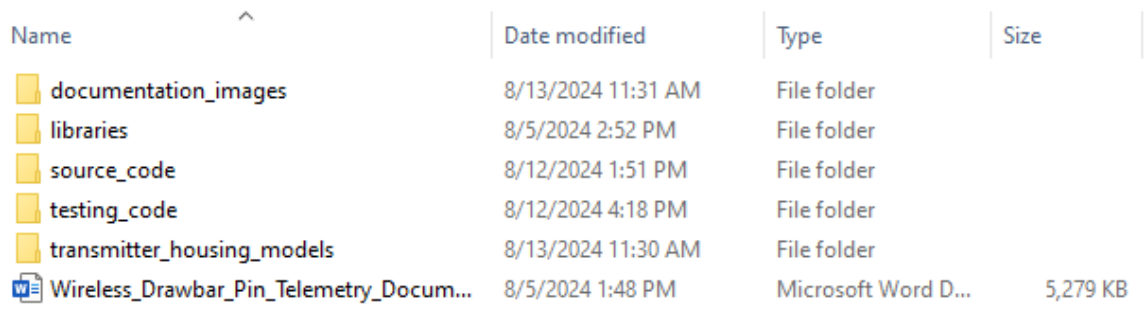
To make the system you need to do the following

1. Order parts
2. Solder components and assemble
3. Set up IDE and libraries
4. Upload code
5. Test operation and debug
6. Print housing

To do these, I recommend reading these sections in order

1) Parts Catalog
2) Hardware Setup
3) Firmware
4) Operation
5) Troubleshooting
6) Prototype summary

If using ReferenceDocs_EVANS, you can find a lot of the files you may want in the Wireless_Drawbar_Pin_Telemetry_Documentation folder



Name	Date modified	Type	Size
documentation_images	8/13/2024 11:31 AM	File folder	
libraries	8/5/2024 2:52 PM	File folder	
source_code	8/12/2024 1:51 PM	File folder	
testing_code	8/12/2024 4:18 PM	File folder	
transmitter_housing_models	8/13/2024 11:30 AM	File folder	
Wireless_Drawbar_Pin_Telemetry_Docum...	8/5/2024 1:48 PM	Microsoft Word D...	5,279 KB

Andy Dequin
DequinAndy@JohnDeere.com

Parts Catalog

Component List and Descriptions

PART	QUANTITY	PURCHASE LINK	COST	DESCRIPTION	DIMENSION (MM)	PITCH
XIAO ESP32C3	2	113991054 Seeed Studio	4.99	Thumb size microcontroller package with Wi-Fi and Bluetooth capability	21x17.5	0.1"
Hx711 Sparkfun	2	474-SEN-13879	9.95	Instrumentation amplifier (single supply, but offset reference)	30.48x22.85	0.1"
XIAO Grove-Shield	2	103020312 Seeed Studio	4.5	XIAO breakout shield, has 4 connectors for grove modules and battery management	25x39	0.1" 2.0mm
Grove-CAN Adapter Module	1	114991377 Seeed Studio	19.9	XIAO CAN module. Connector and small flathead included	20x40x10	0.1"
Medium Lipo 1000mAh	2	ASR00012 TinyCircuits	8.52	Medium Lipo, 1Ah, JST 1.0mm connector	40x30x8	1.0mm
Grove Cables (5 pcs)	1	110990036 Seeed Studio | Mouser	2.1	4 pin JST connectors to communicate between grove modules	50	2.0mm

TOTAL: 77.92 USD

Andy Dequin
DequinAndy@JohnDeere.com

Product Information and Guides

Note – reading the guides is not necessary for building the telemetry system

COMPONENT	PRODUCT INFO	GUIDES
XIAO ESP32C3	Cost Effective RISC-V MCU with Wi-Fi, BLE - XIAO ESP32C3 (seeedstudio.com)	Getting Started with Seeed Studio XIAO ESP32C3 | Seeed Studio Wiki
Grove Serial CAN BUS module	Grove - CAN BUS Module based on GD32E103 - Seeed Studio	SERIAL CAN BUS MODULE - Longan Docs (longan-labs.cc)
XIAO Grove-shield	Grove Shield for Seeed Studio XIAO with embedded battery management chip - Seeed Studio	Grove Shield for XIAO with battery management chip | Seeed Studio Wiki
Medium Lipo 1000mAh	Lithium Ion Polymer Battery 3.7V 1000mAh | TinyCircuits.com	Lithium-ion Polymer 1000mAh Battery Datasheet (mouser.com)
HX711 Sparkfun	SparkFun Load Cell Amplifier - HX711 - SEN-13879 - SparkFun Electronics	Load Cell Amplifier HX711 Breakout Hookup Guide - SparkFun Learn

Andy Dequin
DequinAndy@JohnDeere.com

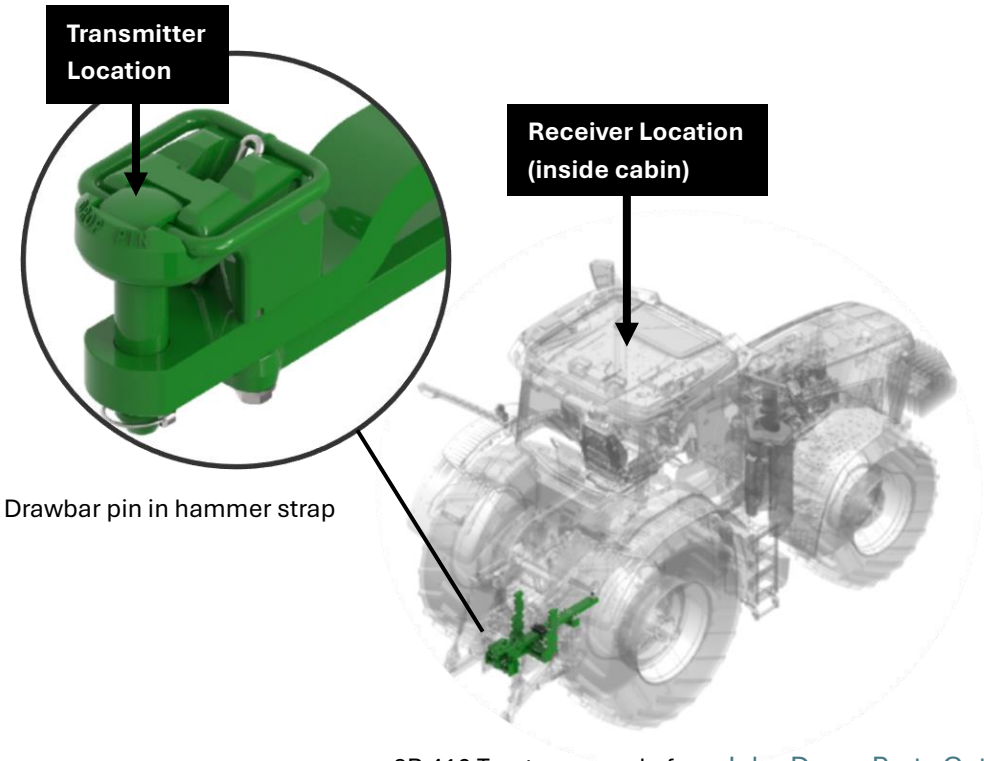
Other Tested Components

Part	Link	Cost	Description	Dimension (mm)	Pitch
Grove-Differential Amplifier Module	103020016 Seeed Studio | Mouser	13.38	Seeed Studio differential amplifier, Mouser has remaining stock. Discontinued product. Single Supply Voltage with no offset. Not reliable, but native to Seeed Studio and grove compatible	40x20	-
Xiao SAMD21 (presoldered)	102010388 Seeed Studio | Mouser	6.5	Microcontroller, without Bluetooth/Wi-Fi, but can come presoldered for working on the solderless solution.	21x17.5	0.1"
Breadboard (solderless)	FIT0096 DFRobot | Mouser	2.9	Breadboard for breaking out modules. Helpful for prototyping and debugging.	(81.3mm x 50.8mm)	0.1"
Small Lipo 115mAh	ICP501421PS Renata	11.32	Small lipo to fit with Grove shield, only 115mAh, non-JST	22.5x14.1x5 .2mm	Flying leads
XIAO ESP32C6	113991254 Seeed Studio | Mouser	7.1	Capable of more wireless data transfer methods that could be helpful. Solid SMD onboard antenna block	21x17.5mm	0.1”
Grove Expansion	103030356 Seeed Studio | Mouser	16.4	XIAO breakout, larger footprint but has JST battery plug. Could reduce soldering requirement	58x42.5mm	-
JST 2.0mm Connectors	321050009 Seeed Studio | Mouser	0.1	JST 2.0mm to connect into development boards	-	2.0mm

Andy Dequin
DequinAndy@JohnDeere.com

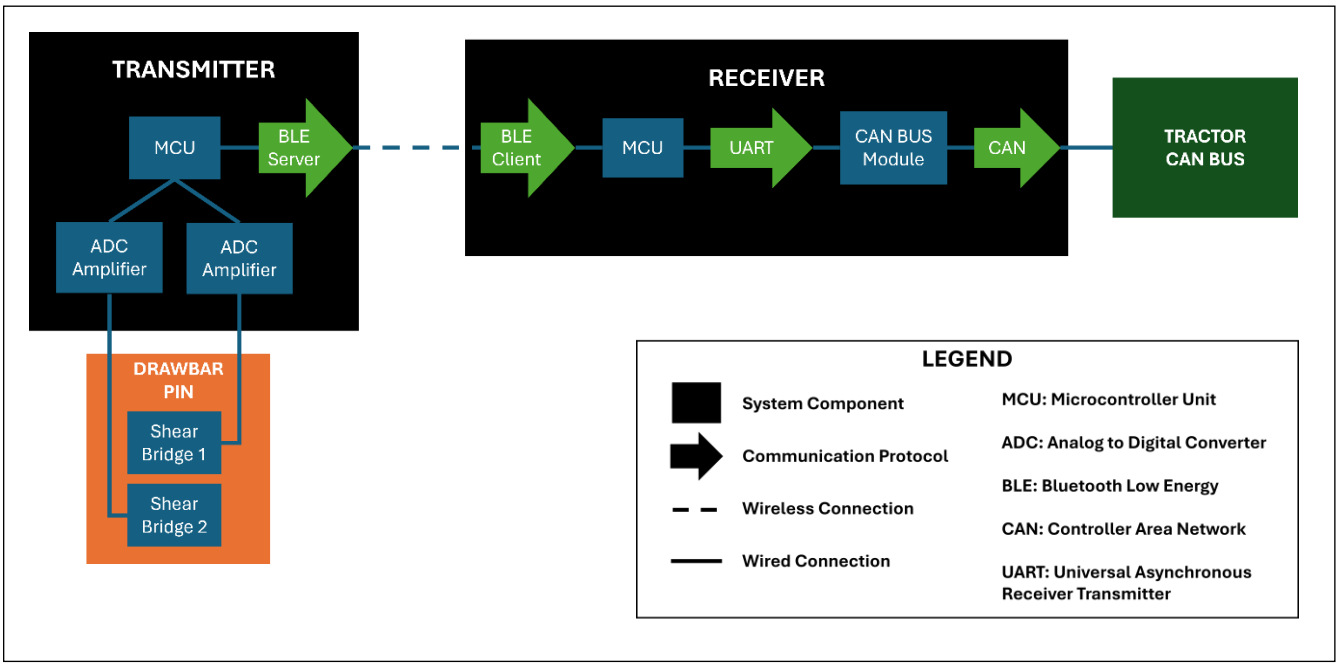
System Overview

Locations on Tractor



8R 410 Tractor example from John Deere Parts Catalog

Data Flow and Communication



Andy Dequin
DequinAndy@JohnDeere.com

Component and Role Summary

ROLE	COMPONENT
Transceiver and Receiver	XIAO ESP32C3
Microcontroller to CAN adapter	Grove Serial CAN BUS module
Breakout board for microcontroller and battery management	XIAO Grove-shield
Battery	Medium Lipo 1000mAh
Load Cell Amplifier	HX711 Sparkfun

Specifications

The input and output specifications of the overall telemetry system are determined by the requirements of the amplifier modules for the input, and the serial CAN module for the output.

	ADC AMPLIFIER	SERIAL CAN MODULE
Input Range	±20mV	-
Output	24-bit digital	CAN differential (~1.8-2.5V)
VCC	3.3V	3.3V
Excitation	3.3V	-
Tested Cable interface	M12	Terminal block (high, low)

Additional Notes:

Output must be ±20mV to be compatible with the default x128 gain on the HX711 boards.

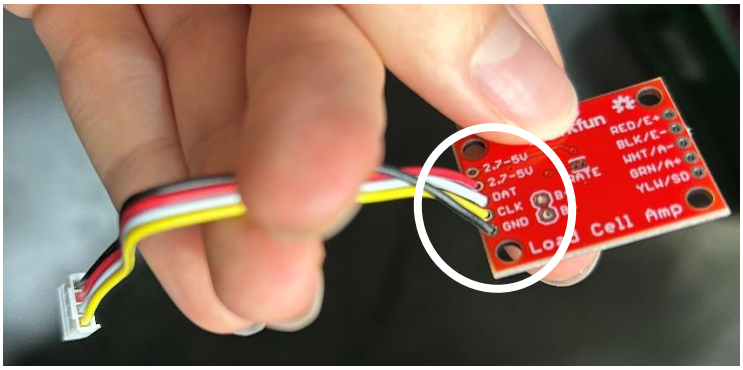
To use ±40mV use the .set_gain(64) function from the HX711 library to change the gain to x64.

The housing is designed with a 5mm opening for M12 cables.

Andy Dequin
DequinAndy@JohnDeere.com

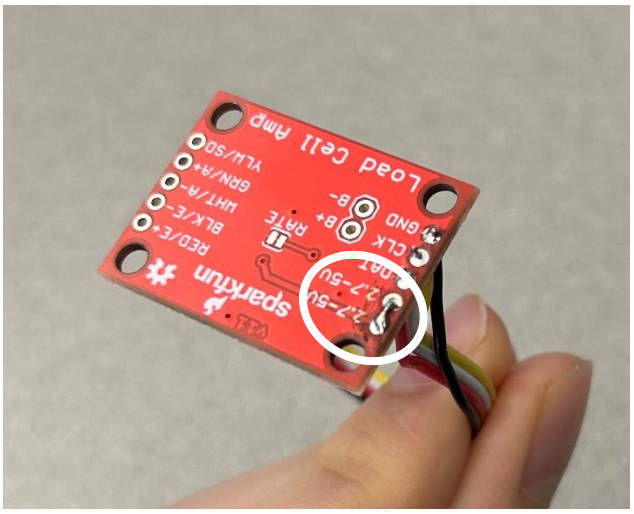
Hardware Setup

Soldering Reference

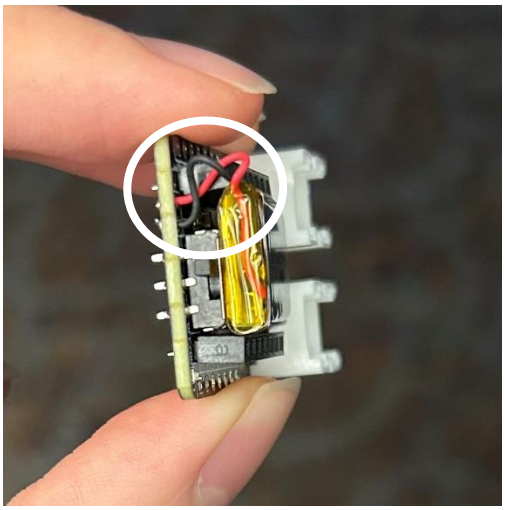


Cut a Grove 4 pin JST connector to 50mm. Solder the DATA and CLOCK pins to the white or yellow lines. Connect ground to GND and 3V3 to VCC or VDD.

Solder the 7 pin headers (from the grove shield package) to the XIAO



Bridge the VCC and VDD pins on the HX711 breakout board

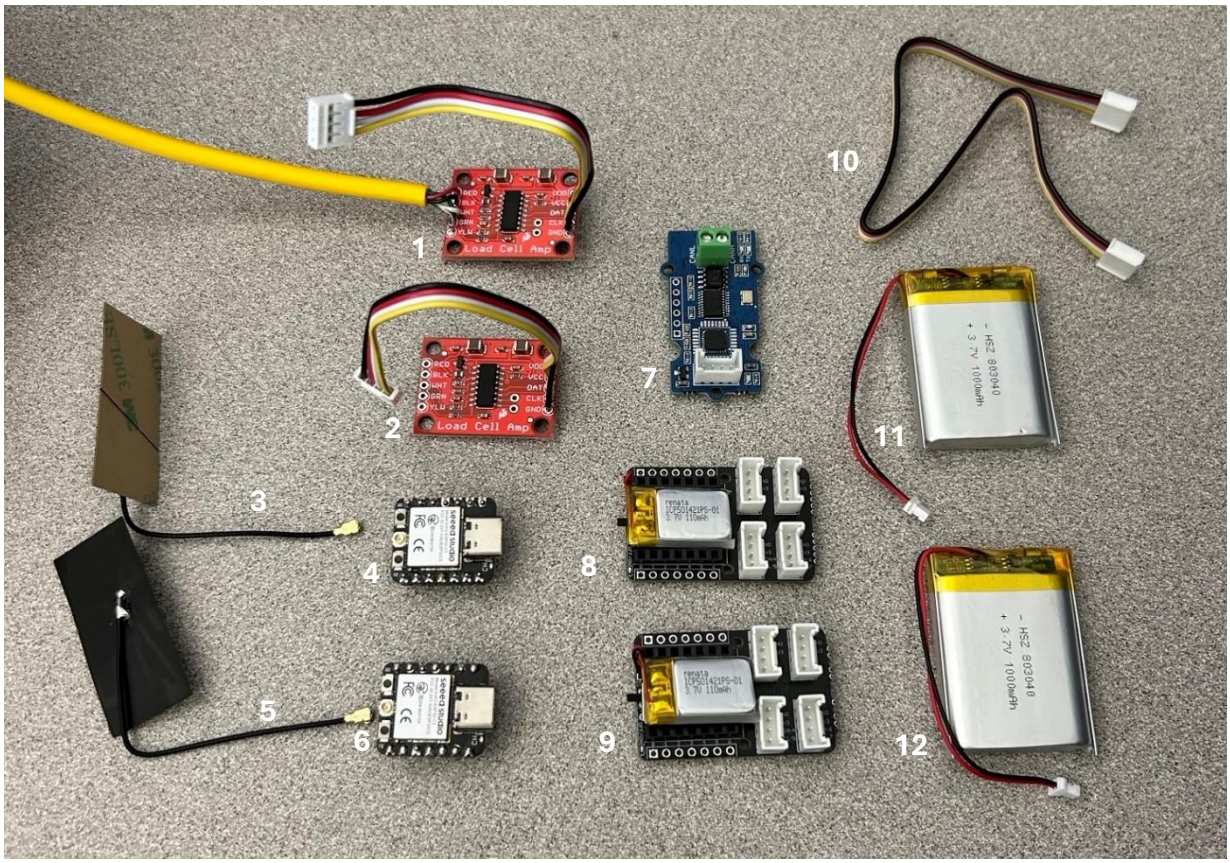


Remove the connector shell and solder the battery leads into the labeled + and – terminals

(This image shows the wrong battery; the medium one should be soldered underneath instead)

Andy Dequin
DequinAndy@JohnDeere.com

Assembly Instructions



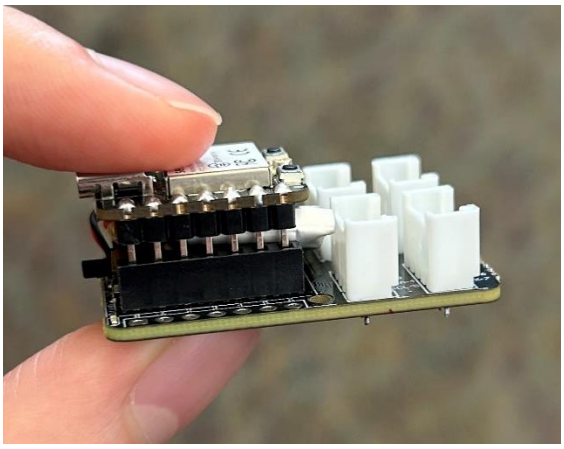
1. Load Cell Amplifier with soldered grove connector (optional M12 connector on input)
2. Load Cell Amplifier with soldered grove connector
3. Antenna for XIAO ESP32C3 microcontroller
4. XIAO ESP32C3 microcontroller with soldered pins
5. Antenna for XIAO ESP32C3 microcontroller
6. XIAO ESP32C3 microcontroller with soldered pins
7. Serial CAN module
8. Grove Shield with soldered battery and break-off (use battery 11 instead of the small one)
9. Grove Shield with soldered battery and break-off (use battery 12 instead of the small one)
10. Grove 4 pin 2.0mm JST connector (included with Serial CAN module)
11. Medium 1000mAh Lipo battery
12. Medium 1000mAh Lipo battery

Note: do not solder the M12 connectors if you want to easily calibrate the amplifiers with a power supply first

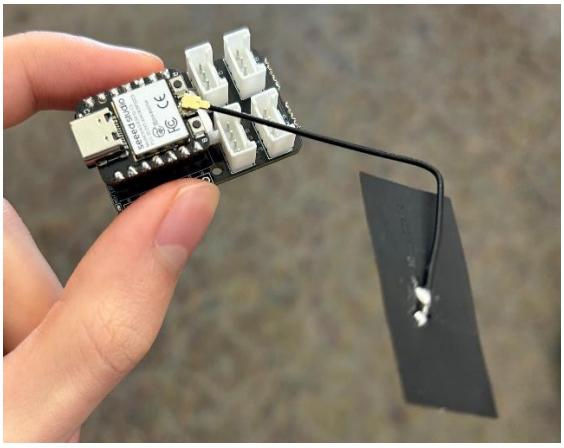
Andy Dequin
DequinAndy@JohnDeere.com

Transmitter Assembly

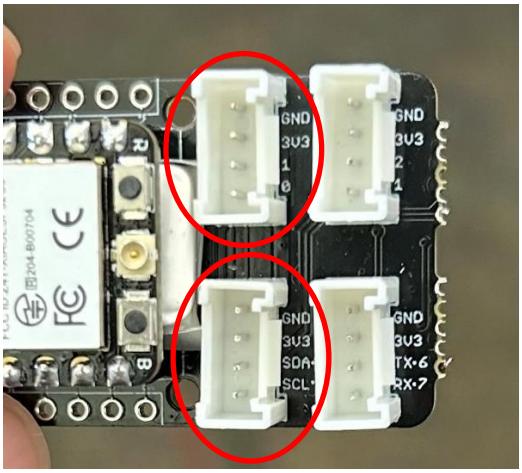
1) Connect the XIAO ESP32C3 to the Grove Shield

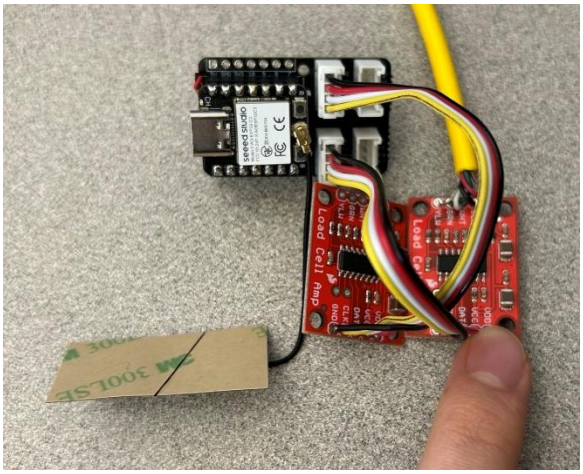


2) Attach the antenna to the XIAO ESP32C3



3) Connect the load cell amplifier modules to the ports labeled “GND, 3V3, SDA 4, SCL 5” and “GND, 3V3, 0, 1”

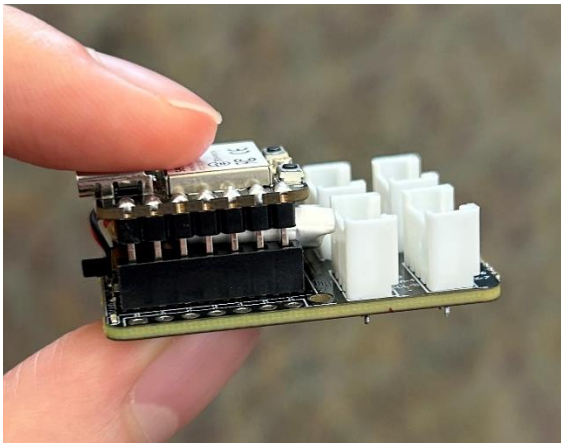




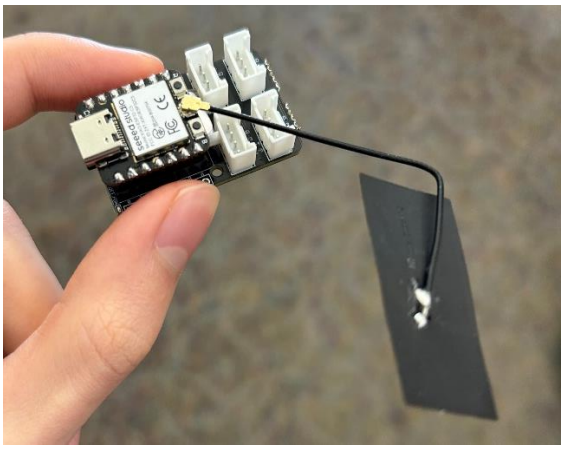
Andy Dequin
DequinAndy@JohnDeere.com

Receiver Assembly

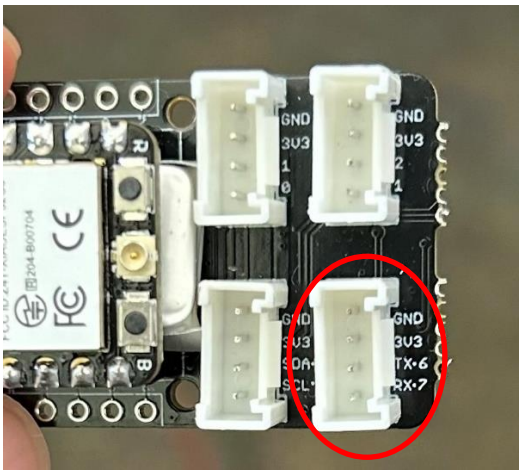
1) Connect the XIAO ESP32C3 to the Grove Shield

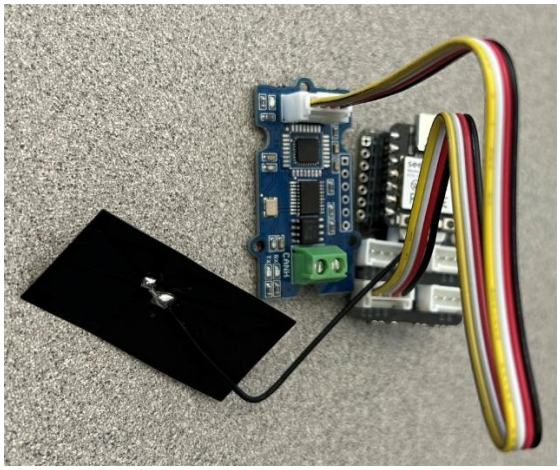


2) Attach the antenna to the XIAO ESP32C3



4) 3) Connect the CAN module to the ports labeled “GND, 3V3, TX 6, RX 7” (for UART)





Andy Dequin
DequinAndy@JohnDeere.com

Transmitter Housing

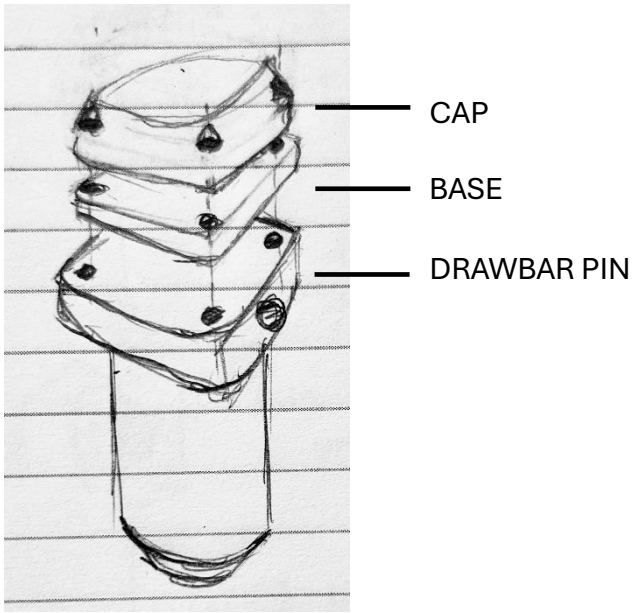
Design concepts and requirements

Target characteristics:

- Signal permeable
- Fits on drawbar pin head (~2x2in)
- Short (1-2in tall)
- Stiff and strong
- Waterproof
- Internal chock absorption

The following housing designs and documentation focus on the geometric compatibility and feasibility. Further testing remains to meet and characterize all the target specifications.

The basic concept is to have the transmitter sit on top of the head of the drawbar pin. The drawbar pin would be modified with threaded holes for fastening.

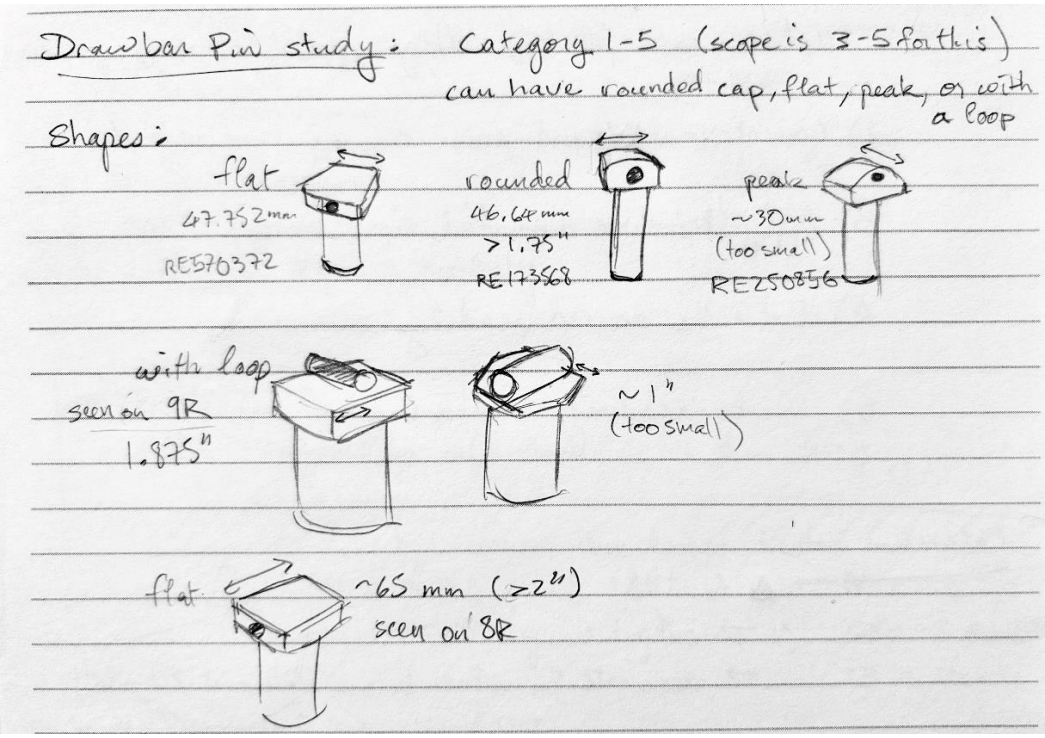


Drawbar pin in hammer strap from John Deere Parts Catalog

The housing and electronics should fit on Category 3, 4, and 5 drawbar pins. For the scope of this project, the housing prototype to reasonably fit onto a pin head surface of 46.64x46.64mm[2].

[2] dimensions of part RE173568

Andy Dequin
DequinAndy@JohnDeere.com

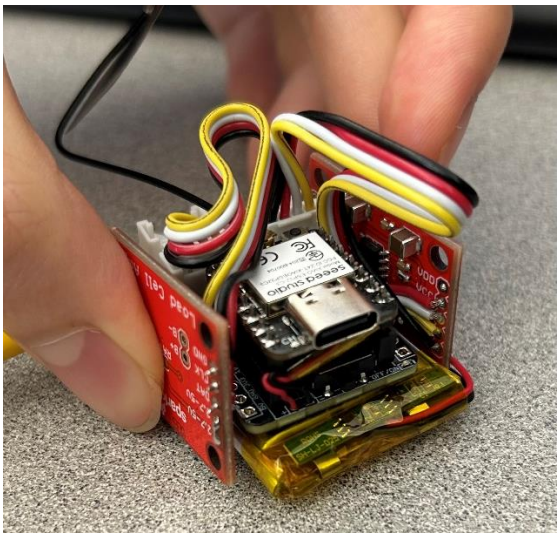


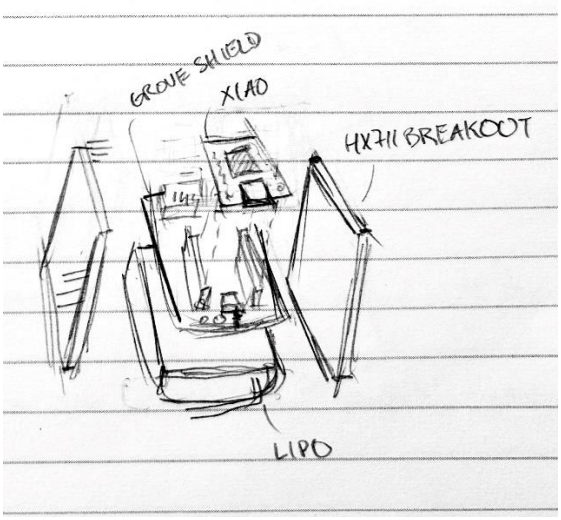
These drawbar pins were found on tractors and by looking through 7R and 8R tractors on the John Deere Parts Catalog.

To account for various shapes, different bases can theoretically be printed to match different pin surfaces while keeping the same cap.

Pin heads with dimensions too close to the payload dimensions were considered too small and would require redesigning the system or considering different attachement methods.

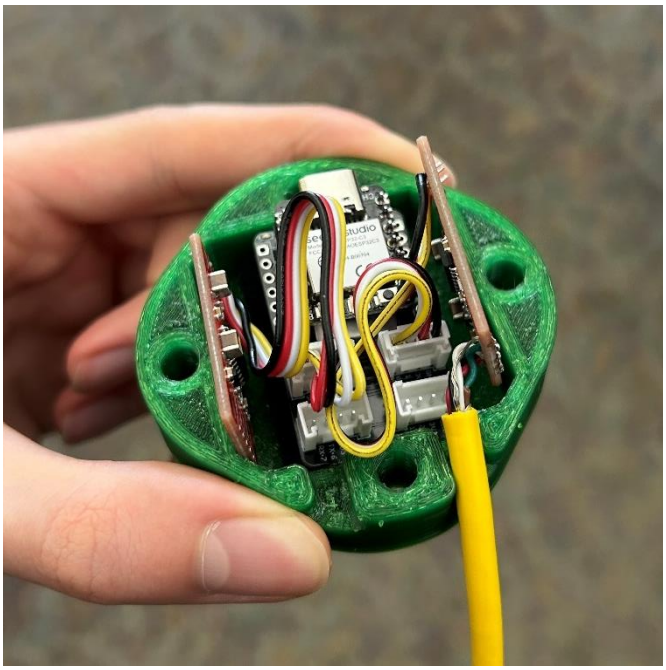
Electronics Layout





This layout was determined to use the space efficiently while leaving enough space for wires and considering HX711 asymmetry.

Cutting the wires to 50mm or using 50mm grove cables, as mentioned in the soldering reference, is helpful for closing the case (images show 100mm cables).



The height of the base part was designed to hold all the components while leaving access to the USB port and power switch. The M12 cable openings are facing opposite of the USB port to avoid clutter.

Prototypes

Two iterations of prototypes were designed in Creo and printed on a Creality cr-10 V3 3D printer out of PLA and PETG with heat-set inserts to fasten the cap and base (inserts not used when fastening to the drawbar pin head).

Summary

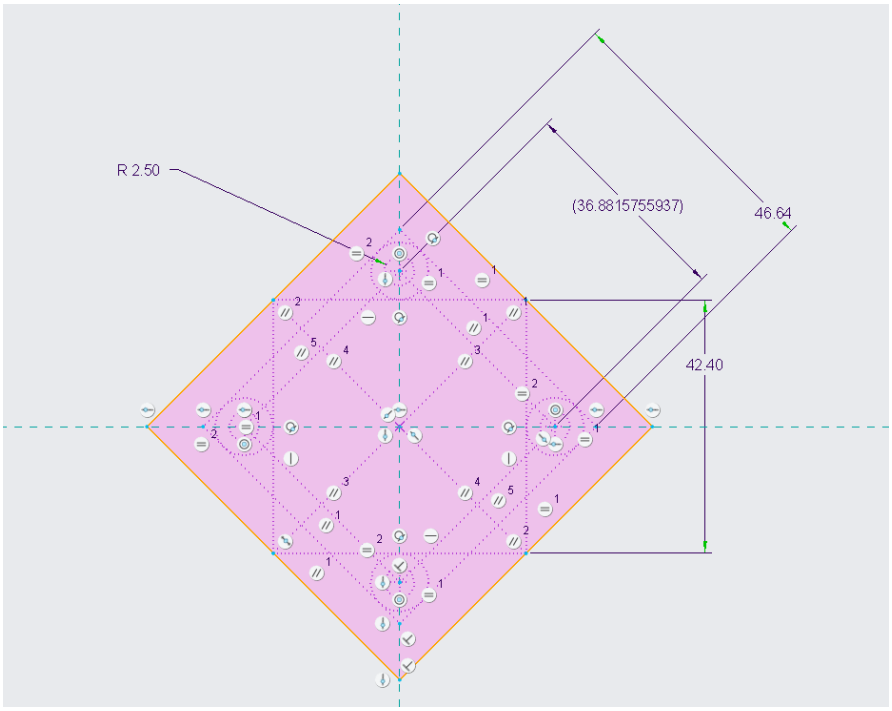
Model Parameters	Test 1	Test 2
Payload Dimensions	40x40x32	42.4x42.4x28
Payload Height in Base	3mm	15mm
Overall Dimensions	~56.5x56.5x40	60.0x60.0x37
Hole Spacing	~35.5mm	36.9mm
Hole dimensions	5m	4.2mm (M4 head) 2.5mm (M4 thread) 6mm (for inserts)
Inserts	M4	M4
Wall thickness	2mm	2mm
Creo Export chord length	0.1	0.01
Material	Black PLA	Green PETG
Density	60-80%	60-80%
Nozzle	0.4mm	0.4mm

Andy Dequin
DequinAndy@JohnDeere.com

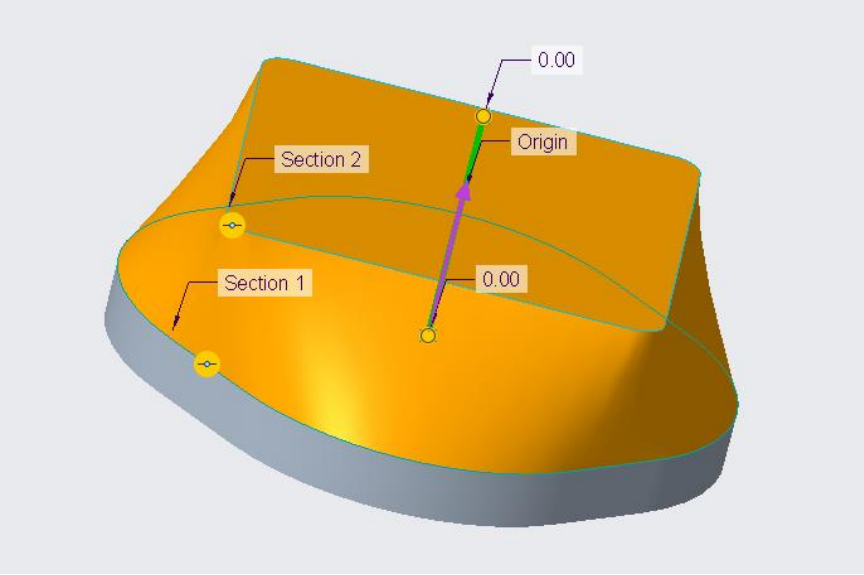
Modeling

The .3mf files that were used for printing are included in the documents. The creo files are called **housing_prototype.prt** (base), and **housing_top.prt** (cap).

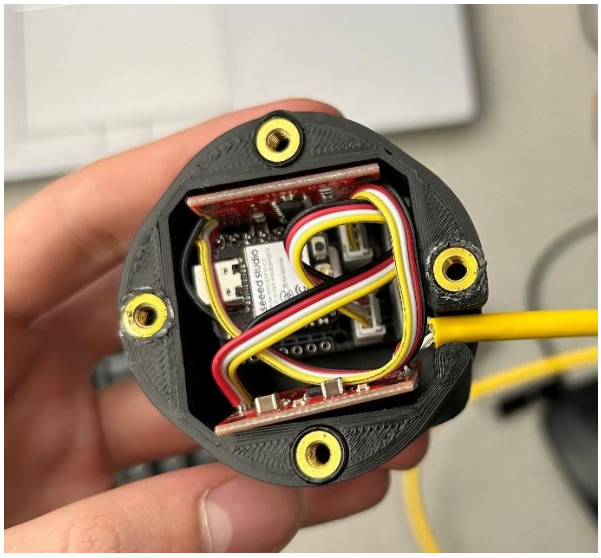
The base sketch imposes a maximum screw hole distance, specific payload dimension, and screw hole dimension. This ensure that we meet the geometric design requirements.

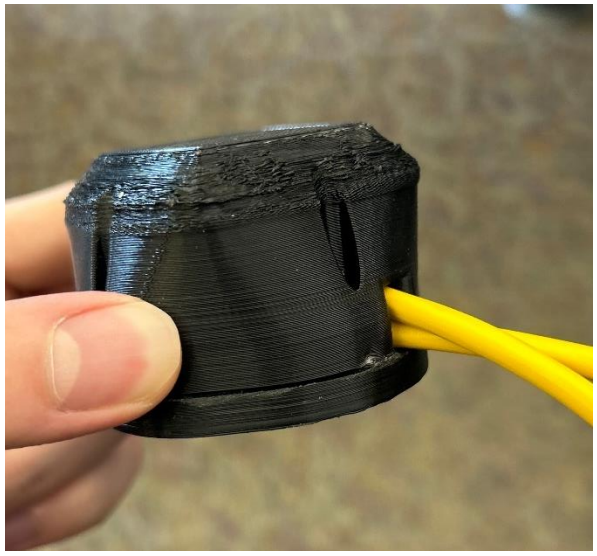


For the outer shell, since the top does not need space for the screws, Swept Blend is used to connect the different sketch profiles.



Iteration 1



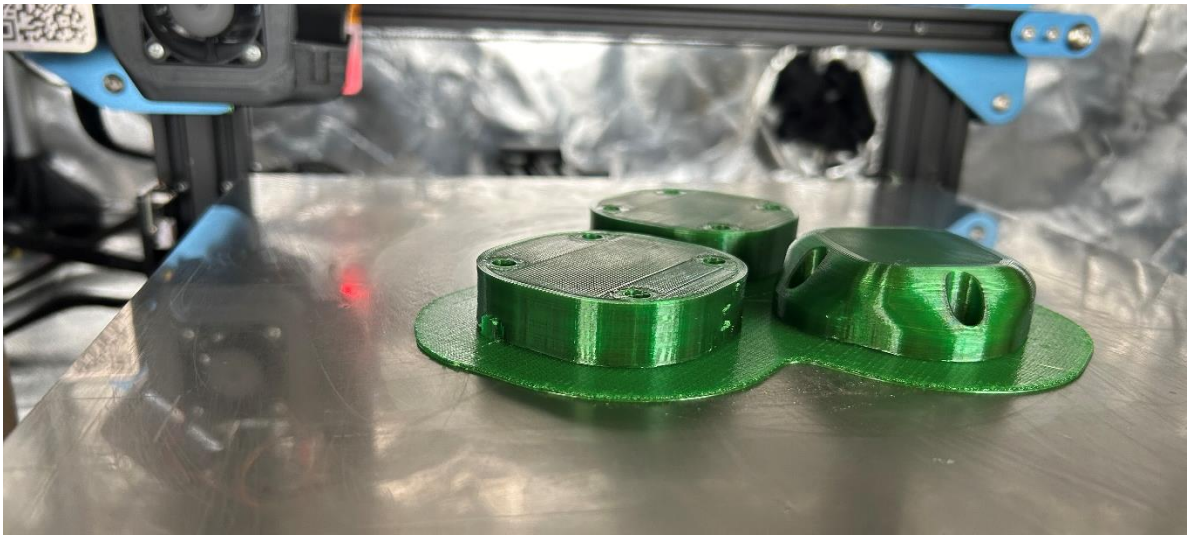


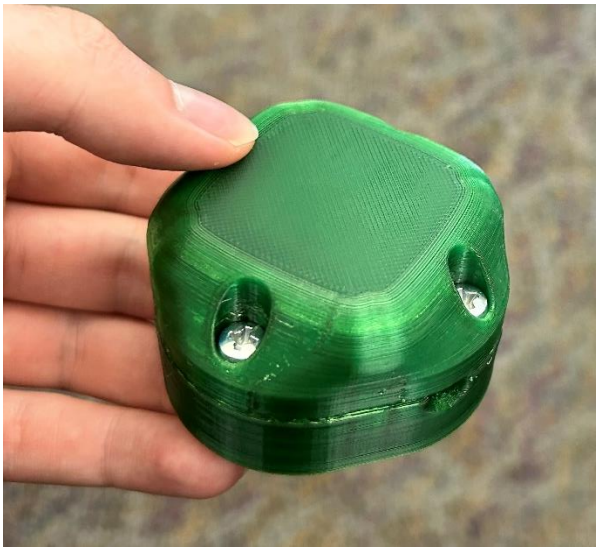
Notes for iteration 2:

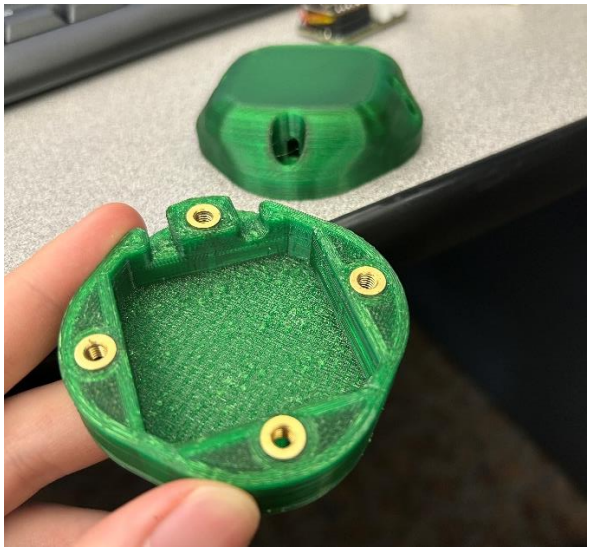
1. 5mm is too small for M4 inserts, use diameter larger than tapered side of insert and add chamfer
2. No space for M4 screw heads
3. Don’t need inserts on both parts, print second base without inserts as well
4. Single opening for M12 cables is inconvenient, place closer to HX711 output
5. 40mm is insufficient for grove shield or battery to fit without rotating vertically
6. Printing cap upside down gives inconsistent finish, but no internal supports
7. Difficult to access USB or power switch
8. Try PETG for comparison

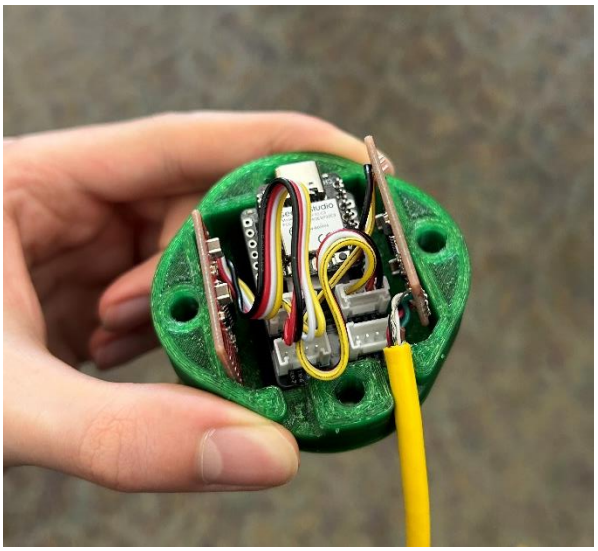
Andy Dequin
DequinAndy@JohnDeere.com

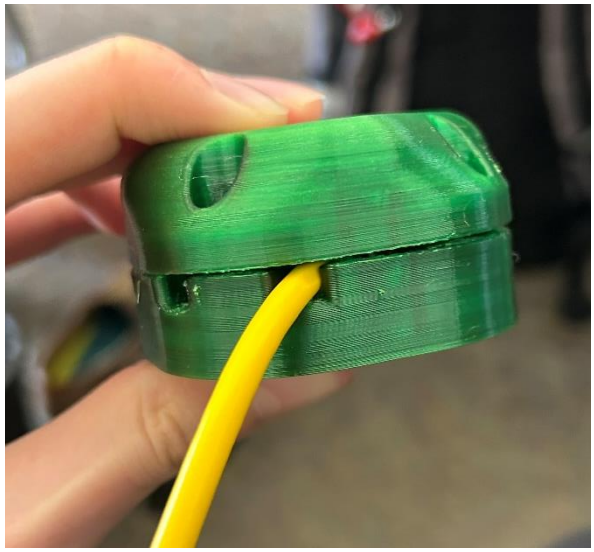
Iteration 2











Firmware

Code Overview

In order to use the XIAO ESP32C3 Microcontrollers to do what we want; we need to program them. To do that, we are using the Arduino IDE to upload custom code that will setup the transmitter to read amplifier data and transmit over Bluetooth and code to setup the receiver to receive Bluetooth and convert it into a CAN output.

The code files should be stored in the same folder that this word document is stored in. They are divided into the ready-to-use "**source**" files, the "**testing**" files used for better understanding the code or debugging, and the Arduino "**libraries**" that the code files rely on.

To get the Firmware working properly, we need to:

1) Download Arduino
2) Configure Arduino to connect to XIAO ESP32C3
3) Calibrate transmitter to read load cell amplifier
4) Upload transmitter and receiver code

What does the code do?

Guided_Calibration_Sparkfun_HX711.ino guides the user through the process of finding the offset and scaling factor for both load cell amplifiers. These four values are then copied over into the Transmitter_code.ino code file.

Transmitter_code.ino integrates reading load cell amplifier data and Bluetooth transmission. It manages retrieving data and sets up a Bluetooth server with a specific identifier that waits for a client to connect. It then encodes the data and transmits it as a characteristic.

Receiver_code.ino looks for the same identifier that the transmitter broadcasts and then decodes the characteristic containing the amplifier data. It also sets up the 29-bit identifier and baud rates used to communicate with the tractor controller (following J1939)

Development Environment Setup

Installing Arduino and XIAO ESP32 Cores

Follow instructions from the Seeed Studio website on setting up Arduino IDE to be compatible with XIAO ESP32C3: Getting Started with Seeed Studio XIAO ESP32C3 | Seeed Studio Wiki

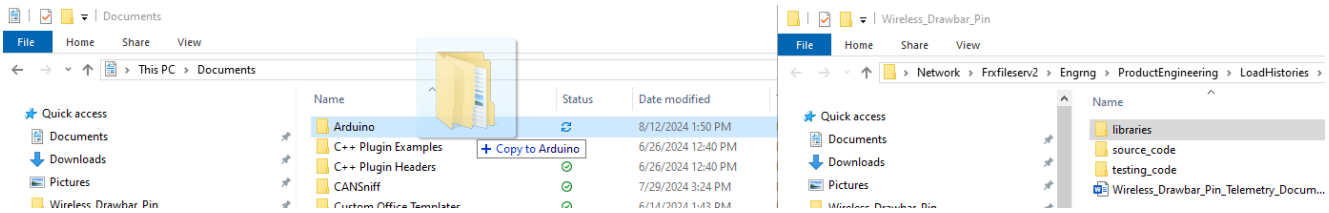
In case the link breaks, here is the outline from the website

1) Install Arduino: Software | Arduino (free)
2) Navigate to **File > Preferences**, and fill **"Additional Boards Manager URLs"** with the url below: https://raw.githubusercontent.com/espressif/arduino-ESP32/gh-pages/package_ESP32_index.json
3) Navigate to **Tools > Board > Boards Manager...**, type the keyword "**ESP32**" in the search box, select the latest version of **ESP32**, and install it.
4) Navigate to **Tools > Board > ESP32 Arduino** and select "**XIAO_ESP32C3**". The list of board is a little long and you need to roll to the bottom to reach it.
5) Navigate to **Tools > Port** and select the serial port name of the connected XIAO ESP32C3. This is likely to be COM3 or higher (**COM1** and **COM2** are usually reserved for hardware serial ports).

Note: In Arduino IDE 2.0 the UI is slightly different and the board may automatically select “ESP FAMILY DEVICE”. It is still important to select the exact board “XIAO_ESP32C3”, since it might fail the upload and require a reboot otherwise. If that does happen, head to the Troubleshooting part of this document.

Arduino Libraries

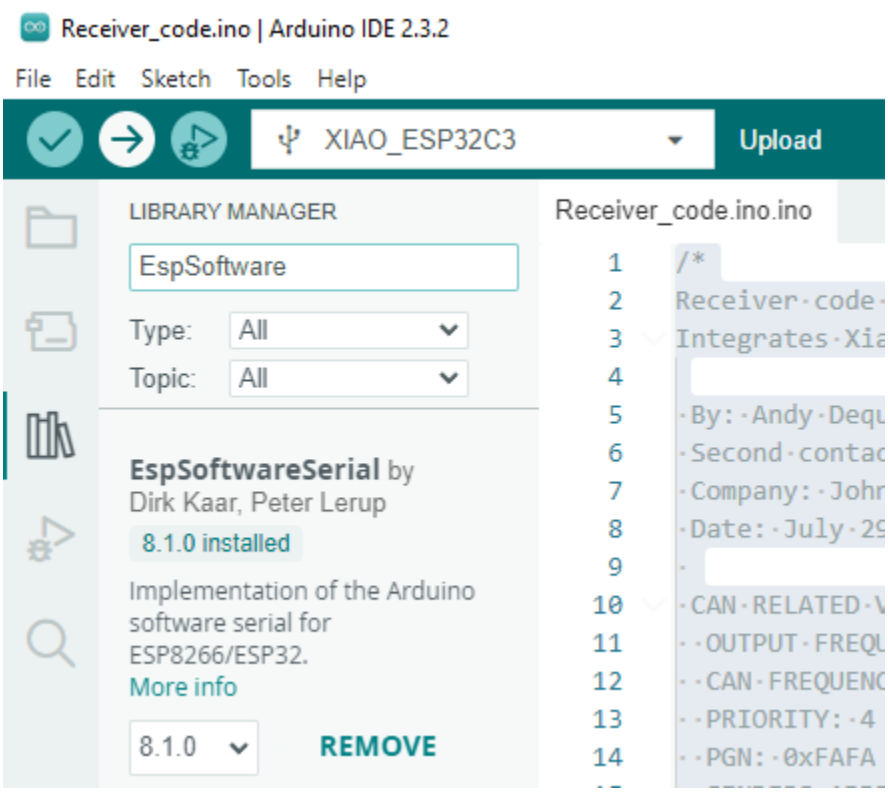
Next, we need to include the relevant Arduino libraries that the source code depends on. If you are using ReferenceDocs_EVANS, you can directly add the libraries folder from to your arduino sketchbook folder. **AND NOW YOU ARE DONE!!!! SKIP TO NEXT SECTION**



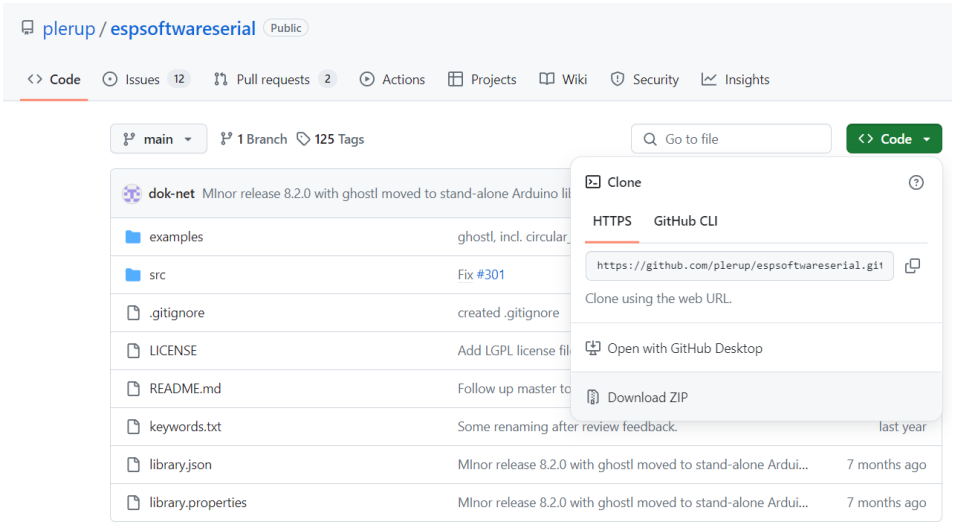
Otherwise, you can add the files manually (which is many more steps). The Serial_CAN_module library also needs to be modified before it can work.

You can learn more about including libraries here: Installing Additional Arduino Libraries | Arduino

1) The EspSoftwareSerial can be installed directly from Arduino UI in Library Manager



2) For the other libraries, you need to first download the zip file with the code from GitHub. Click on the green “Code” button on the top right.

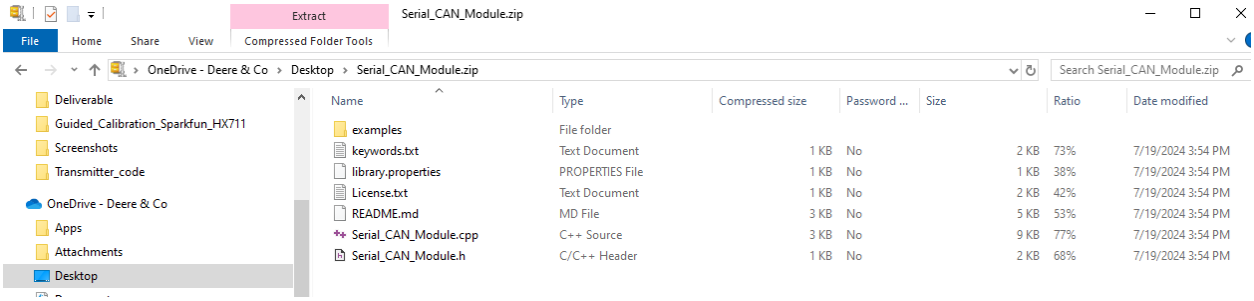


Here are the github libraries:

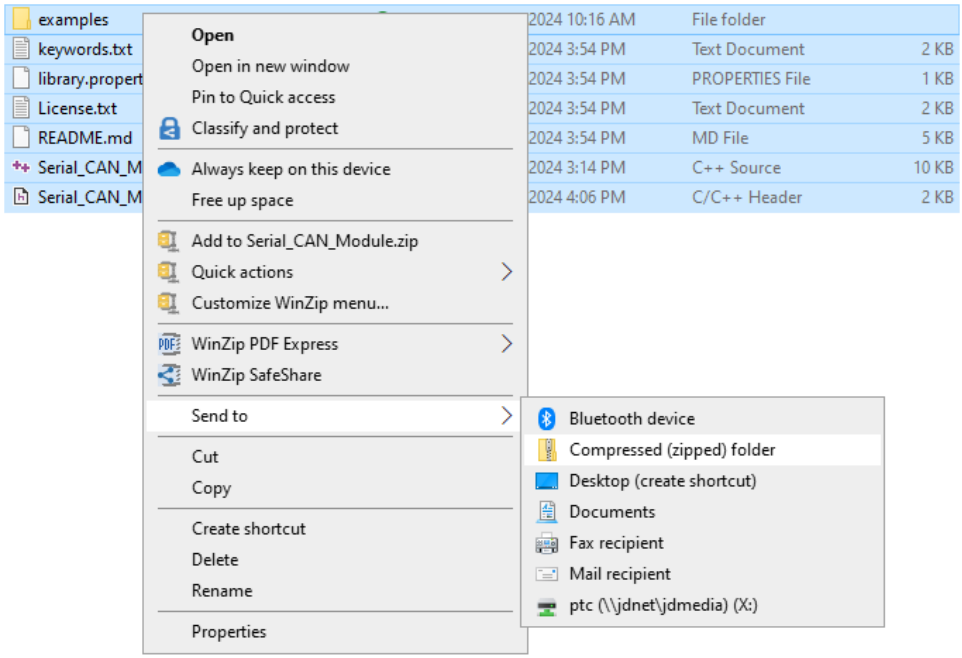
HX711 library: [GitHub - bogde/HX711: An Arduino library to interface the Avia Semiconductor HX711 24-Bit Analog-to-Digital Converter (ADC) for Weight Scales.]

Serial_CAN_Module: [GitHub - Longan-Labs/Serial_CAN_Arduino]

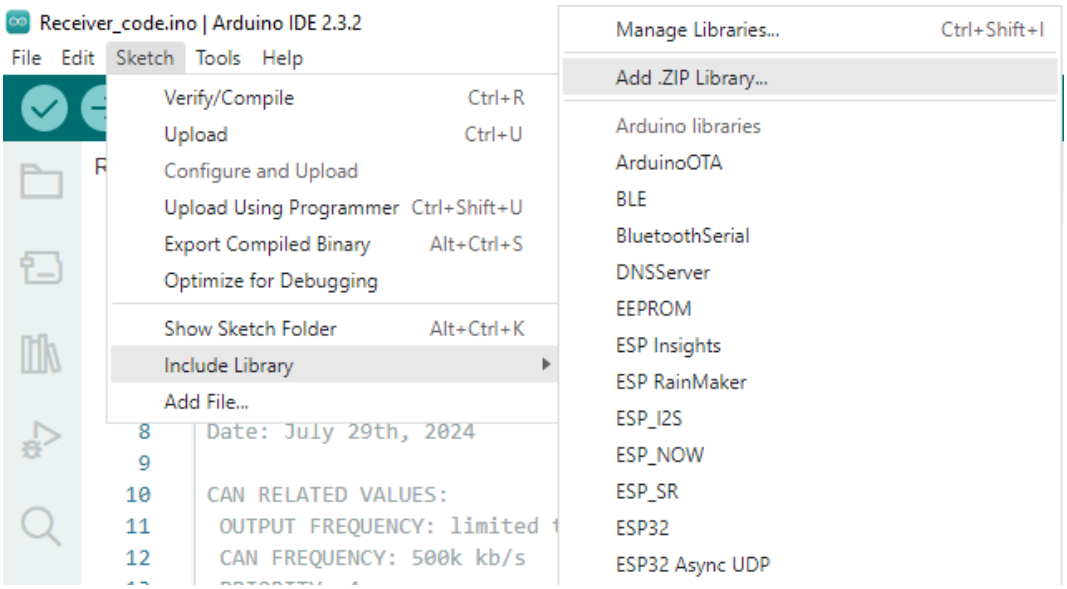
3) Once you have downloaded the zip folders, you can use the Arduino interface to add them to your libraries. You need to do some rearranging so that the .cpp and .h files inside of the “src” folder are top level. Like this



This involves unzipping the file, moving the .cpp and .h files to the top level, and then compressing again.



4) Once both of them are top level, in Arduino, go to Sketch>Include Library> Add .ZIP Library and select the libraries.



5) Once the libraries are added, you need to fix some lines in Serial_CAN_Module.h so that it is compatible with the ESP32 version of SoftwareSerial. These files cannot be edited directly in Arduino for some reason, so we need to use a different software IDE like Visual Studio.

In Serial_CAN_Module.h:

```
1  #ifndef __SERIAL_CAN_MODULE_H__
2  #define __SERIAL_CAN_MODULE_H__
3  // ID3 ID2 ID1 ID0 EXT RTR DTA0 DTA1 DTA2 DTA3 DTA4 DTA5 DTA6 DTA7
4  #include <Arduino.h>
5  #include <SoftwareSerial.h>
6  #define uchar unsigned char
7
```

add “#include <SoftwareSerial.h>” after line 4

```
33  class Stream;
34  using SoftwareSerial = EspSoftwareSerial::UART;
35  class HardwareSerial;
```

Change “using SoftwareSerial;” to “using SoftwareSerial = EspSoftwareSerial::UART;” on line 34

Once you have made the changes, make sure to save the file and it will update automatically in Arduino.

6) There is a mistake in Serial_CAN_Module.cpp that we need to fix as well. In the send method, add a return statement right before the closing bracket. “return 0;”

```
28  unsigned char Serial_CAN::send(unsigned long id, uchar ext, uchar rtrBit, uchar len, const uchar *buf)
29  {
30      unsigned char dta[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};
31
32      dta[0] = id>>24;         // id3
33      dta[1] = id>>16&0xff;    // id2
34      dta[2] = id>>8&0xff;     // id1
35      dta[3] = id&0xff;        // id0
36
37      dta[4] = ext;
38      dta[5] = rtrBit;
39
40      for(int i=0; i<len; i++)
41      {
42          dta[6+i] = buf[i];
43      }
44      for(int j=0; j<14; j++)
45      {
46          canSerial->write(dta[j]);
47      }
48      return 0;
49  }
50
```

Calibration Using Arduino Interface

Once you have properly installed Arduino and the board packages, we can move on to setting up the firmware.

Since the transmitter is also reading from the load cell amplifiers (HX711 boards), it needs the **OFFSET VALUE** and **SCALING FACTOR** that maps the raw readings to the value you are interested in. We will upload the Guided_Calibration_Sparkfun_HX711.ino code file into the transmitter and following the calibration sequence to find these two values.

Components Required for Calibration

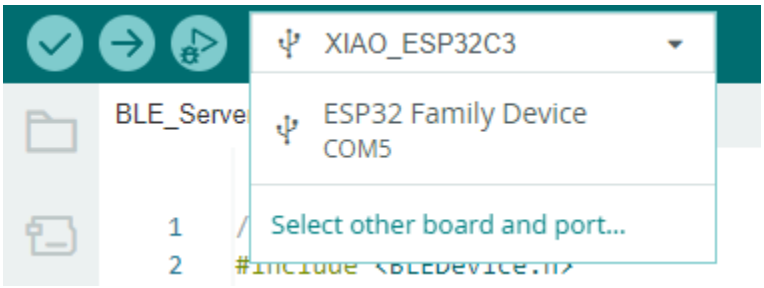
For the following instructions you will need:

1. The transmitter module assembled in Transmitter Assembly with at least one amplifier
2. A computer with Arduino installed and a USB type-C cable
3. Something to be measured. Either a or b:
    a. Practical testing
        i. Load Cell
        ii. M12 connectors
        iii. A way to apply a known force
    b. Actual calibration
        i. DC power supply (capable of mV output)
        ii. Banana cable with hook ends
        iii. Multimeter (in case your power supply readout isn't precise)

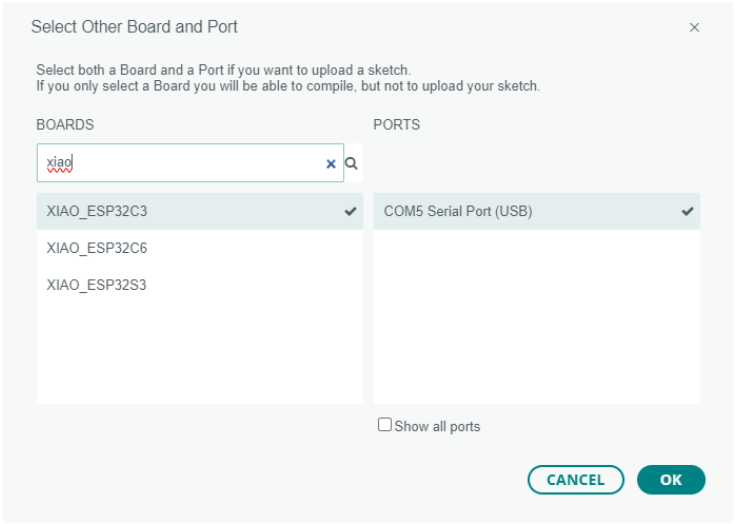
Load Cell Amplifier Setup

1) Plug in the microcontroller to your computer using a USB type C cable
2) Make sure the red LED indicator labeled "CH" lights up (not when battery powered)
3) If you are using an Arduino version less than 2.0, complete step 4) and 5) in Install Arduino and XIAO ESP32 Cores, to connect to the correct board and port. Those steps also work for Arduino 2.0+.

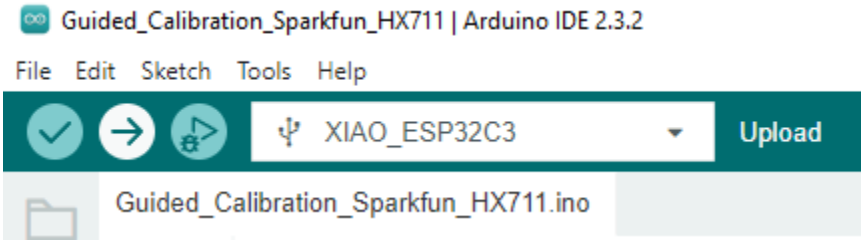
Alternatively, in Arduino 2.0, the port should autofill with the right type of device and



We won't rely on the autofill completely however, but it's a good indication that the microcontroller is working, and that things are properly installed on your computer. Click on "Select other board and port" to specify the board, then search for "xiao" and select XIAO_ESP32C3. The right COM port should already be autofilled correctly.



4) Go to File>Open and find Guided_Calibration_Sparkfun_HX711.ino
   It might be at this path if you work at John Deere
   \\Frxfileserv2\Engrng\ProductEngineering\LoadHistories\ReferenceDocs_EVANS\Wireless_Drawbar_Pin\source_code
5) Upload the file by clicking on the right arrow button

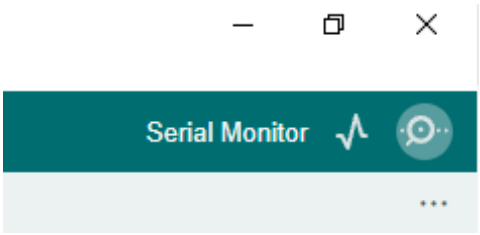


Once the upload is complete, you should see this in the Output window:

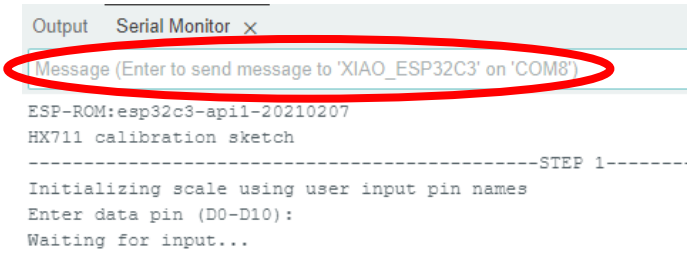
```
Writing at 0x000436bc... (80 %)
Writing at 0x0004a807... (90 %)
Writing at 0x00050a56... (100 %)
Wrote 278896 bytes (155905 compressed) at 0x00010000 in 2.1 seconds (effective 1042.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

If not, refer to the Troubleshooting section

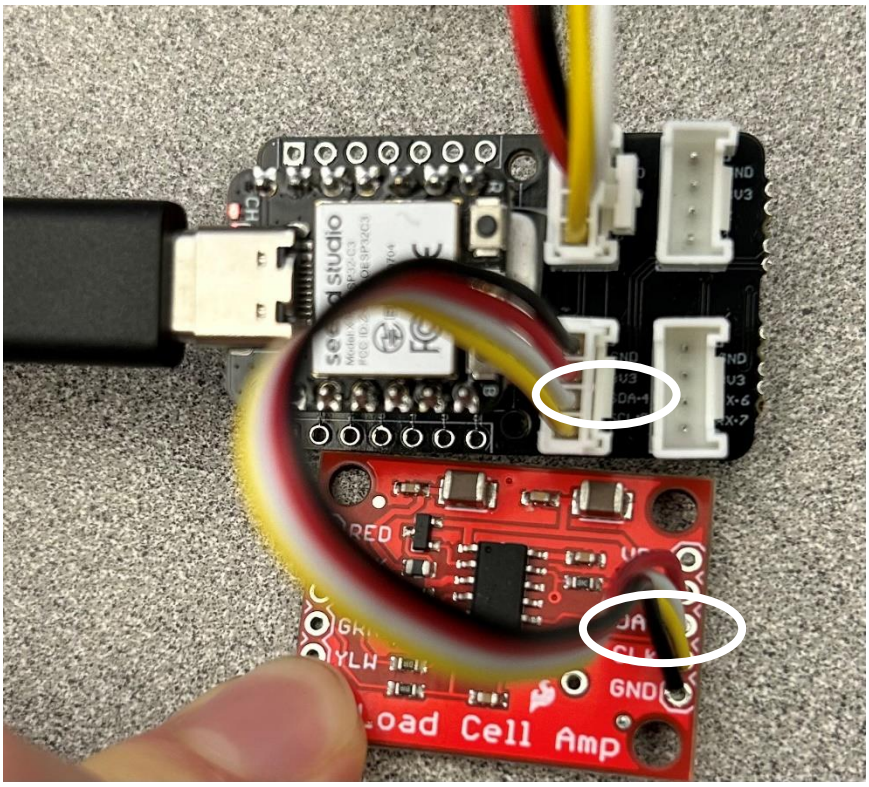
6) To start the calibration program, make sure the Serial Monitor is opened by clicking on the magnifying glass in the top right corner. If the Serial Monitor was automatically opened upon upload, but no text is present, close and open the Serial Monitor again to trigger the start of the program. If still nothing shows up, press enter into the Serial Monitor message box.



7) To input values into the program, use the Message text box



The first prompt asks for the pin of the XIAO ESP32C3 that the data pin of the HX711 board connected to. To figure this out, follow the cable connecting the pin labelled “DAT” to the Grove Shield and find the number printed next to it.



In this case, the DAT pin is the white cable and goes into the pin labeled “SDA-4” which is the same as pin “D4”[3].

8) After entering D4 (or whichever pin you are using for data), find the clock pin in the same way and press enter. This will create a load cell object that we can read from.

Calibration Sequence

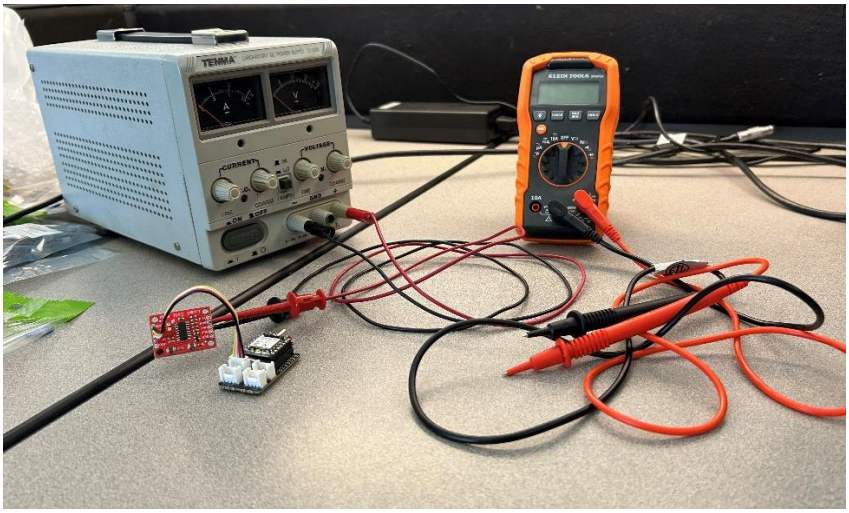
For the remaining steps, you will need to connect the amplifier to something.

For example, here the amplifier is soldered onto an M12 connector and connected to a torsion transducer:

[3] See glossary for pin naming convention

Note that if you do use a load cell, you will have to find your own conversion factor and units. The reference calibration value that you use is also limited to 327 (2^16/2/100)

If you just want to calibrate it to read millivolts, which is probably more realistic for actual use, use a DC power supply and multimeter (if the power supply readout doesn't have mV precision). Hook – to GRN and + to WHT.



Once we have everything ready, we can continue from the Load Cell Amplifier Setup steps to start the calibration:

9) Apply a reference load to the load cell or set the power supply to 0.0mV (or as close as you can). Then press enter in the Serial Monitor interface to continue.

10) You can verify that the reading makes sense with the printed information. If the amplifier is reading 0.00, it is very likely that the microcontroller is not reading values properly or that the input pins to the amplifier are floating. Here is a more typical reading:

```
value parsed as:
Setting scale and offset...
Read average of 10: -565232.88
Scale with offset: -1187.63
Reading with offset and scaling: -110.87
```

While you may expect the "reading with offset and scaling" to be the same as "Scale with offset" here, they are different because they are separate readings subject to noise and the amplifier's resolution.

11) If calibrating to read millivolts, set known the voltage to 20mV then press enter in the Serial Monitor interface. If using a load cell, apply a known force and type the input value then press enter.
Note: For x128 gain on the amplifier (default setting), the input range is ±20mV.

12) We now have the SCALING FACTOR and OFFSET VALUE! We just need to make sure they do what we want them to. Press enter again to start reading data continuously.

13) Once you've confirmed the range of accuracy, press enter again to display the SCALING FACTOR and OFFSET VALUE again. These are the values you need to copy over to the Transmitter_code.ino file that you are using

14) Press enter again to start the process again to start over or to get calibration values for the second amplifier.

15) Once you confirmed that the values are what you expect, you can copy them over to your local copy of the Transmitter_code.ino file. Make sure to set the correct data

pin and clock pin for each load cell.

```
//////////////////////////////EDIT THESE VALUES//////////////////////////////
#define scaling_factor1 18269.10 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define offset_value1 681309 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define DOUT1 D4 //ENTER YOUR DATA PIN HERE
#define CLK1  D5//ENTER YOU CLOCK PIN HERE

#define scaling_factor2 18269.10 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define offset_value2 681309 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define DOUT2 D1 //ENTER YOUR DATA PIN HERE
#define CLK2  D0//ENTER YOU CLOCK PIN HERE
////////////////////////////////////////////////////////////////////////////////
```

Change all these values. Now you are done! The next step is to upload the transmitter and receiver code to the respective microcontrollers and to make sure they can connect via Bluetooth.

Source Code

The source code is stored in the same directory where this document is. It is also reproduced below, but it may have issues with line breaks in pdf form due to wrapping.

If you want to learn about the CAN and BLE communication implementations, read the comments.

The code files and folders need to be downloaded and stored locally first in order to work.

Guided_Calibration_Sparkfun_HX711.ino

```
/*
 Example using the SparkFun HX711 breakout board as transducer amplifier

 By: Andy Dequin (DequinAndy@JohnDeere.com)
 Second contact: Scott Evans (evansbenjamins@johndeere.com)
 Company: John Deere
 Date: Aug 12th, 2024

TESTED SETUP:
  Upload to a XIAO ESP32C3 connected to a Grove Shield
  A grove connector (4 pin JST 2.00mm pH) is used to connect pins {1, 2, 3V3, GND} to the Sparkfun HX711 breakout board
  The grove connector is stripped on one side to connect to the pins {GND, CLK, DATA, VCC} on the Sparkfun HX711 breakout
  On the Sparkfun HX711 breakout, VCC and VDD are soldered together by bending the lead from the bare wire soldered to VCC
  GND and + outputs of a DC Power supply are connected to WHT and GRN inputs of the Sparkfun HX711 breakout

  Note: Calibration was attempted using a breadboard wheatstone bridge, but unsuccessful.
    Most any pin on the XIAO ESP32C3 will be compatible with DOUT/CLK. check pin information on https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/
    The HX711 board can be powered from 2.7V to 5V so the XIAO Grove Shield Connector 3V3 power should be fine.

CALIBRATION SEQUENCE:
 1) Upload code with no voltage applied to i/o pins (recommended), if upload gets stuck: hold the boot (B) button before connecting to computer and then release
 2) The code is waiting for the Serial Monitor to open. If the Serial Monitor is open, but nothing is printed, close and open it.
 3) Connect V- and V+ or apply 0V (differential), then enter any input to tare
 4) Apply known voltage difference to V- and V+, then enter the desired value that it should read as
 5) Verify the readings in continuous readout mode, enter any input to restart the calibration sequence
```

```
 This example code uses bogde's excellent library: https://github.com/bogde/HX711
 bogde's library is released under a GNU GENERAL PUBLIC LICENSE
*/

#include "HX711.h" //This library can be obtained here http://librarymanager/All#Avia_HX711
#include <Arduino.h> //for serial string parsing
int datapin; // values assigned through user input in loop()
int clockpin;
float cal_factor;

//helpers
float read_all(HX711 &scale_obj){
  //reads and prints out the averaged reading, the avg reading-OFFSET, and (avg reading-OFFSET)/SCALE
  //used for debugging during calibration
  //PARAMETER: HX711 object reference
  //RETURNS: float value from .get_units() which reads value from the hx711, subtracts the offset, then
divides by the scale factor

  Serial.print("Read average of 10: ");
  Serial.println(scale_obj.read_average(10));
  Serial.print("Scale with offset: ");
  Serial.println(scale_obj.get_value(10));
  Serial.print("Reading with offset and scaling: ");
  Serial.println(scale_obj.get_units(10));
  Serial.println("");
  return scale_obj.get_units(10);
}

void init_scale(HX711 &scale_obj, int datapin, int clockpin){
  //initialize scale with data pin and clock pin,
  //sets scale factor to 1.0,
  //offsets the current reading to be 0 (tare)
  //PARAMETERS: reference to HX711 object (should be declared at the top),
  //            data pin number (D1-D10 are prespecified internal mappings to the XIAO ESP32C3)
  //            clock pin number
  //RETURNS: nothing
  scale_obj.begin(datapin, clockpin);
  Serial.println("Initializing scale and offset...");
  scale_obj.set_scale(1.0);// sets scaling factor
  scale_obj.tare(10); //Reset the scale to 0 by setting offset to current reading
  read_all(scale_obj);
}

void clear_Serial(){
  //clears out any characters in the Serial buffer (inputs into the Serial Monitor)
```

```
  //PARAMETERS: none
  //RETURNS: nothing
  while (Serial.available()) {
    Serial.read();
  }
}

int getPinNumber(const String& pinName) {
  if (pinName == "D0") return D0;
  if (pinName == "D1") return D1;
  if (pinName == "D2") return D2;
  if (pinName == "D3") return D3;
  if (pinName == "D4") return D4;
  if (pinName == "D5") return D5;
  if (pinName == "D6") return D6;
  if (pinName == "D7") return D7;
  if (pinName == "D8") return D8;
  if (pinName == "D9") return D9;
  if (pinName == "D10") return D10;
  return -1; // Return -1 if the pin name is invalid
}

float wait(const char* announcement = ""){
  // wait for Serial Monitor input, can also request user value for calibration
  // when parsing serial input, first checks for D0-D10 pins, then converts to float
  // PARAMETERS: announcement string to specify user input
  // RETURN: float value from user input in Serial Monitor
  clear_Serial();
  Serial.println(announcement);
  Serial.println("Waiting for input..." );
  // Wait until a character is entered in the serial monitor
  while (Serial.available() == 0) {// Do nothing, just wait for input
  }
  String input_pin = Serial.readStringUntil('\n');
  input_pin.trim(); // remove trailing/leading whitespace

  Serial.print("value parsed as: ");
  Serial.println(input_pin);
  Serial.println("");
  int pinNumber = getPinNumber(input_pin);

  if (pinNumber == -1){// if not a pin number, assume a float value
    clear_Serial();
    return input_pin.toFloat();
```

```
  }else {// otherwise return the converted pinNumber
  return pinNumber;
  }
}

void print_scale_and_offset(HX711 &scale_obj){
  //returns the offset value and scaling factor to be used in main HX711_BLE_server code file
  //PARAMETERS: HX711 object
  //RETURNS: none
  Serial.print("OFFSET VALUE: ");
  Serial.print(scale_obj.get_offset());
  Serial.print(", SCALING FACTOR: ");
  Serial.println(scale_obj.get_scale());
}

void setup() {
  Serial.begin(115200);
  Serial.println("HX711 calibration sketch");
  while (!Serial) { // Wait for serial port to connect
  }
}

void loop() {
  Serial.println("-----------------------------------------------STEP 1-----------------------------------
----------");
  Serial.println("Initializing scale using user input pin names");
  datapin = wait("Enter data pin (D0-D10): "); //implicit casting float output to int
  clockpin = wait("Enter clock pin (D0-D10): ");//no error handling here
  wait("Make sure the input is 0mV, then press Enter to initialize. This will set the offset value
(tare)");
  HX711 scale;
  init_scale(scale, datapin, clockpin); // INTIALIZE HX711 BOARD WITH GIVEN DATA AND CLOCK PINS
  // the offset value and the offset+scaled value will be different even if the scaling is 1.0 due to
multiple readings

  Serial.println("-----------------------------------------------STEP 2-----------------------------------
----------");
  Serial.println("");
  Serial.println("Set your input to 20mV for a range of +/-20mV then press Enter");
  float expected = wait("note: if using load cell, apply a known force and type in the value (must be
<328), then press Enter");
  float calibration_value = scale.get_units(10);  // read current value as the calibration reference value

  if (expected){
    cal_factor = calibration_value/expected;
```

```
  } else{
    cal_factor = calibration_value / 20;
  }

  scale.set_scale(cal_factor);
  print_scale_and_offset(scale);
  Serial.println("");


  //continuous readout until input entered
  Serial.println("-----------------------------------------------STEP 3-----------------------------------
----------");
  wait("Confirm values by reading values continuously\nPress Enter to start (then press Enter again to
exit)");

  while(Serial.available() == 0){
    Serial.println(scale.get_units());
    delay(100);
  }
  Serial.println("-----------------------------------------------STEP 4-----------------------------------
----------");
  Serial.println("Make sure to copy the scale and offset values into the Transmitter_code.ino file");
  print_scale_and_offset(scale);
  Serial.println("");
  wait("Press Enter to restart");
}
```

Transmitter_code.ino

```
/*
Transmitter code for Wireless Drawbar Pin Telemetry System
Integrates reading from Sparkfun HX711 load cells and XIAO ESP32C3 Bluetooth Low Energy Communication

 By: Andy Dequin (DequinAndy@JohnDeere.com)
 Second contact: Scott Evans (evansbenjamins@johndeere.com)
 Company: John Deere
 Last Modified: Aug 8th, 2024

MAKE SURE TO EDIT THE LOAD CELL PIN AND CALIBRATION VALUES
MAKE SURE TO EDIT THE LOAD CELL PIN AND CALIBRATION VALUES
MAKE SURE TO EDIT THE LOAD CELL PIN AND CALIBRATION VALUES

DATA LENGTH AND RATE
```

```
  Data is read from 2 load cells as float values (64 bits), then copied into an 8 byte data array (64 bits) for transmission
  Packets are sent roughly every 90ms, 900ms if you average 10 times.

HOW TO USE:
  Turn on or press reset button (R). Requires a connection to start transmitting
  I use the LightBlue app on an iphone by Punchthrough to connect to the server and read transmissions (look for "XIAOESP32C3_BLE")

COMPONENTS:
  Upload to a XIAO ESP32C3 with attached antenna, connected to a Grove Shield with soldered battery
  A Sparkfun HX711 Load Cell Amplifier is connected to port D4 and D5, and another one is connected to D0 and D1
  The amplifiers should be connected to a load cell or a power supply for testing. Connect power supply using the GRN and WHT pins.

REFERENCES:
  Calibration values are determined using the Guided_Calibration_Sparkfun_HX711.ino code
  The load cell amplifier firmware uses bogde's library which is released under a GNU GENERAL PUBLIC LICENSE: https://github.com/bogde/HX711
  Bluetooth setup is modified from the template server code found here: https://wiki.seeedstudio.com/xiao_ESP32c6_bluetooth/
*/
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <BLEServer.h>
#include "HX711.h"

//define load cell objects to use HX711 library methods
HX711 load_cell_1;
HX711 load_cell_2;

////////////////////////////////EDIT THESE VALUES//////////////////////////////
//values for load cell 1
#define scaling_factor1 115994.63 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define offset_value1 4339 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define DOUT1 D4 //ENTER YOUR DATA PIN HERE
#define CLK1  D5//ENTER YOU CLOCK PIN HERE

//values for load cell 2
#define scaling_factor2 115994.63 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define offset_value2 4339 //obtained from Guided_Calibration_Sparkfun_HX711.ino
#define DOUT2 D1 //ENTER YOUR DATA PIN HERE
#define CLK2  D0//ENTER YOU CLOCK PIN HERE
```

```
////////////////////////////////////////////////////////////////////////////

//BLE Server name (the other ESP32 name running the server sketch)
#define bleServerName "XIAOESP32C3_BLE"

//bluetooth things
BLECharacteristic *pCharacteristic; // declare characteristic pointer
bool deviceConnected = false;// initialize connection flag (modified by callback class)

class MyServerCallbacks: public BLEServerCallbacks {
  //handles connection and disconnection events
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };

  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};

void setup() {

  BLEDevice::init(bleServerName); //initialize BLE device and assign name
  BLEServer *pServer = BLEDevice::createServer();// create a BLE server instance and assign to pointer
  pServer->setCallbacks(new MyServerCallbacks());// set callback functions for the BLE server to handle
connection events

  //create a service on the server
  BLEService *pService = pServer->createService(BLEUUID((uint16_t)0x181A)); // Environmental Sensing ID,
not particularly relevant but necessary

  //create characteristic of service
  pCharacteristic = pService->createCharacteristic(
    BLEUUID((uint16_t)0x2A59), // Analog Output ID, defines type of characteristic
    BLECharacteristic::PROPERTY_NOTIFY // indicates characterisitc will support notifications
  );
  pCharacteristic->addDescriptor(new BLE2902());//enables server to notify clients of value changes, and
enables clients to subscribe

  pService->start(); //start service, clients can now connect if they have the ID

//set up advertising so device actually be found, broadcasting id continuously
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();//get advertising pointer from BLE device
  pAdvertising->addServiceUUID(pService->getUUID()); // uses the service UUID defined earlier for
advertising
```

```
  pAdvertising->setScanResponse(true);// allow other devices to scan for more information
  pAdvertising->setMinPreferred(0x0); // sets minimum delay between advertizing packets, 0 means no
specific requirement
  pAdvertising->setMinPreferred(0x1F);// sets max delay as 31 * 0.625ms (this is the time unit for BLE)
  BLEDevice::startAdvertising();// start advertising name and other information

  //load cell initialization code using user defined values
  load_cell_1.begin(DOUT1, CLK1); // intializes using data and clock pins
  load_cell_1.set_scale(scaling_factor1); //This value is obtained by using the
Guided_Calibration_Sparkfun_HX711 code
  load_cell_1.set_offset(offset_value1);  //Assuming there is no weight on load cell at start up, reset
the load cell to 0

  load_cell_2.begin(DOUT2, CLK2);// intializes using data and clock pins
  load_cell_2.set_scale(scaling_factor2); //This value is obtained by using the
Guided_Calibration_Sparkfun_HX711 code
  load_cell_2.set_offset(offset_value2);  //Assuming there is no weight on load cell at start up, reset
the load cell to 0
}

void loop() {
  if (deviceConnected) {

    uint8_t byte_array[8];//container for data to be sent

    float force_val_1 = load_cell_1.get_units();//32 bit value
    // float force_val_1 = analogRead(A1);
    memcpy(byte_array, &force_val_1, sizeof(float));//copy raw bits from float

    float force_val_2 = load_cell_2.get_units();//32 bit value
    // float force_val_2 = analogRead(A0);//32 bit value
    memcpy(byte_array + sizeof(float), &force_val_2, sizeof(float));
    //copy raw bits from float at data array location shifted so that its behind the first float

    pCharacteristic->setValue(byte_array,8);
    pCharacteristic->notify(); // notifies client so that they don't have to read continuously
    // delay(70); // bluetooth stack will go into congestion if too many packets are sent
  }
}
```

Receiver_code.ino

```
/*
Receiver code for Wirless Drawbar Pin Telemetry System
Integrates Xiao ESP32C3 Bluetooth Low Energy Communication and a Serial CAN communcation
```

```
 By: Andy Dequin (DequinAndy@JohnDeere.com)
 Second contact: Scott Evans (evansbenjamins@johndeere.com)
 Company: John Deere
 Date: July 29th, 2024

 CAN RELATED VALUES:
  OUTPUT FREQUENCY: limited to 19200 UART baudrate (can set to 9600 19200 38400 115200)
  CAN FREQUENCY: 500k kb/s
  PRIORITY: 4
  PGN: 0xFAFA
  SENDERS ADDRESS: 0xDB
  29-bit CAN ID: 100 00 1111 1010 1111 1010 1101 1011
  which is parsed as 0001 0000 1111 1010 1111 1010 1101 1011


RESOLUTION:
  Data is read from 4 byte data arrays, one for each load cell, (64 bits) via Bluetooth, then inserted
into the CAN message data array directly

HOW TO USE:
  I use the LightBlue app on an iphone by Punchthrough make sure the server is active
  Turn off and on the receiver to start
  Check the green indicator on the Serial CAN module is solid and not flashing
  Check that the red indicators labeled TX and RX are either on or flashing to indicate is being output.
  If there are integration issues, try uncommenting out all the Serial. related lines to debug

COMPONENTS:
  Upload to a XIAO ESP32C3 with attached antenna, connected to a Grove Shield
  A Serial CAN Module plugged into the port with the RX and TX pins (for UART communcation)
  A different XIAO ESP32C3 is sending signals using the Transmitter_code or BLE_Server example code

  Note: the Serial CAN module must be version 1.3 or higher (1.4 is used here) to support VCC powered at
3V3. This is what the grove connector is supplying so this is necessary.
    There are many different types of CAN modules, I picked this one because it was available on mouser
and compatible with the grove ecosystem. Make sure to pick the right one you want to use.

REFERENCES:
  Bluetooth setup is modified from the template client code found here:
https://wiki.seeedstudio.com/xiao_ESP32c6_bluetooth/
  Communication with the Serial CAN module uses longan labs' documentation https://docs.longan-
labs.cc/1030001/#software
*/
```

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEClient.h>
#include "SoftwareSerial.h"
#include "Serial_CAN_Module.h"

// Define input and output channel values
#define can_tx RX  // tx of serial can module, the yellow cable (TX is mapped to pin 21 in pins_arduino.h)
#define can_rx TX  // rx of serial can module, the white cable (RX is mapped to pin 20 in pins_arduino.h)
//why are these flipped? Im not sure, but it works.

Serial_CAN can;

// Define CAN bus frame identifier
#define PRIORITY_VALUE 0b100
#define PGN_VALUE 0xFAFA
#define SENDER_ADDRESS_VALUE 0xDB
#define CAN_FRAME_ID (((unsigned int)PRIORITY_VALUE << (PGN_BITS + SENDER_ADDRESS_BITS)) | \
                      ((unsigned int)PGN_VALUE << SENDER_ADDRESS_BITS) | \
                      SENDER_ADDRESS_VALUE)

#define EXTENDED 1
#define NOT_EXTENDED 0
#define REQUESTING 1
#define NOT_REQUESTING 0

#define PRIORITY_BITS 3
#define PGN_BITS 18
#define SENDER_ADDRESS_BITS 8

// Setup Bluetooth variables
BLEClient* pClient;
bool doconnect = false;
#define bleServerName "XIAOESP32C3_BLE"
static BLEAddress *pServerAddress;
BLEUUID serviceUUID("181A");
BLEUUID charUUID("2A59");

// Checks for target server and gets address
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    if (advertisedDevice.getName() == bleServerName) {
      advertisedDevice.getScan()->stop();
      pServerAddress = new BLEAddress(advertisedDevice.getAddress());
      //Serial.println("Device found. Connecting!");
```

```
      doconnect = true;
    }
  }
};

void encodeAndSend(float __load_x, float __load_y=0.0f) {
  /*
  Converts 32-bit float into unsigned 16-bit integers scaled by 10
  Packs into CAN frame output. Check documentation for decoding parameters.

  PARAMETERS: 32-bit float values from Bluetooth
  RETURN: nothing
  */

  auto clampToInt16 = [](float value)-> uint16_t {
    //encoding and error handling function
        float test_val = value*100+32767; //encoding
        if (test_val > 65535) return 0xFFFF;
        if (test_val < 0) return 0xFFFF;
        return static_cast<uint16_t>(test_val);
    };

  uint16_t x_to_send = clampToInt16(__load_x);
  uint16_t y_to_send = clampToInt16(__load_y);
  //Serial.print("encoded value x: ");
  //Serial.println(x_to_send);
  //Serial.print("encoded value y: ");
  //Serial.println(y_to_send);
//construct data 8-byte array
 unsigned char __data[8] = {
        0xFF, 0xFF,
        static_cast<unsigned char>(x_to_send >> 8),
        static_cast<unsigned char>(x_to_send & 0xFF),
        static_cast<unsigned char>(y_to_send >> 8),
        static_cast<unsigned char>(y_to_send & 0xFF),
        0xFF, 0xFF
    };
  //construct and send CAN frame
  can.send(CAN_FRAME_ID, EXTENDED, NOT_REQUESTING, 8, __data);
}

bool initializeCAN() {
  /*
  Initialize Serial CAN Module's UART transmission at 9600
  Attempts to switch to 38400 baud
```

```
  CAN Communication rate is 500k by default, attempts to set 500 for validation
  If anything fails, the Serial CAN module may not be working properly (power cycle)
  */
  //Serial.println("Initializing CAN...");
  can.begin(can_tx, can_rx, 9600);  // Initialize CAN with TX/RX pins and baud rate
  if (can.baudRate(SERIAL_RATE_38400)) {
    //Serial.println("CAN UART rate set successfully");
  } else {
    //Serial.println("Failed to set CAN UART rate");
    return false;
  }

  if (can.canRate(CAN_RATE_500)) {
    //Serial.println("CAN baud rate set successfully");
  } else {
    //Serial.println("Failed to set CAN baud rate");
    return false;
  }
  return true;
}

void initializeBLE() {
  //Serial.println("Starting BLE client...");
  BLEDevice::init("XIAOESP32C3_Client");
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30); // scans for 30 seconds to reduce power consumption
}

void setup() {
  //Serial.begin(9600);

  initializeCAN();
  initializeBLE();
}

void loop() {
  if (doconnect) {
    pClient = BLEDevice::createClient();
    pClient->connect(*pServerAddress);
    //Serial.println(" - Connected to server");

    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    BLERemoteCharacteristic* pCharacteristic = pRemoteService->getCharacteristic(charUUID);
```

```
    pCharacteristic->registerForNotify([](BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t*
pData, size_t length, bool isNotify) {
      float load_x, load_y;
      memcpy(&load_x, pData, sizeof(float));
      memcpy(&load_y, pData + sizeof(float), sizeof(float)); //shift pointer to next float

      //Serial.print("load x: ");
      //Serial.print(load_x);
      //Serial.print(" load y:");
      //Serial.println(load_y);
      encodeAndSend(load_x, load_y);
    });
    doconnect = false; // Reset doconnect to avoid reconnection attempts
  }

  // delay(1000); // Adjust this delay as necessary
}
```

Andy Dequin
DequinAndy@JohnDeere.com

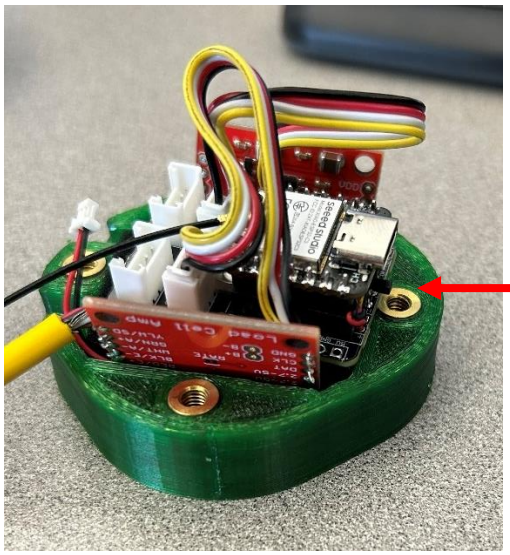
Operation

Usage Instructions

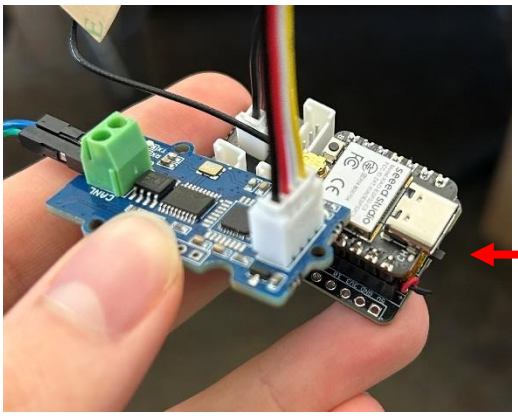
Once the system is fully assembled, you just need to make sure that the transmitter is turned on before the receiver and the load cell data should be transmitting into the CAN network of the tractor controller.

Currently once the connection is broken, the transmitter needs to be restarted. Resetting the receiver does not work, it needs to be turned off to restart.

1) Turn on Transmitter



2) Turn on Receiver. The receiver has a 30s timer to connect before it stops scanning.



3) If not transmitting (TX and RX red indicators flashing), power cycle the receiver.

For future improvements:

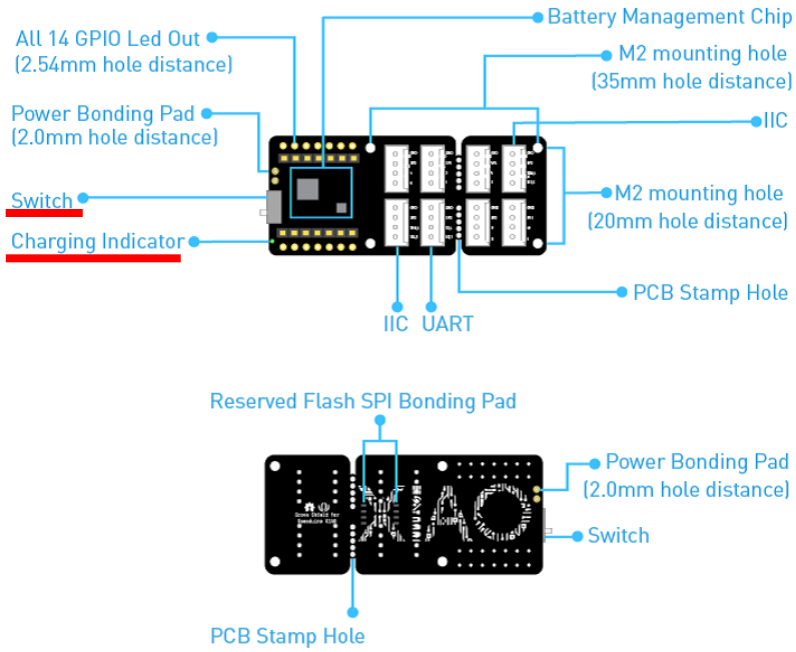
- Instead of manually turning off the transmitter (which would be in the back of the tractor), it would be nice to be able to send Bluetooth commands like “SLEEP” and “WAKE” for it to enter low power mode.
- The Transmitter would not stop broadcasting after the client disconnects.

Charging Instructions

To charge the Lipo battery that is soldered onto the Grove Shield, plug in the ESP32C3 while the battery switch is in the on state. The battery will be fully charged once the indicator turns off.

Note that the indicator does not turn on if the board is not plugged in. The battery works by connecting to the ground and 3V3 pin of the XIAO microcontroller (5V is not used).

Read the product information for more details.



(image from the seeed studio product information page)

Grove Shield Charging Indicator

Indicator State	Significance
FLASHING	Not charging
ON	Charging
OFF	Fully charged

Grove Shield Specifications

Parameters	Values
Power Supply	5V / 3.7V Lithium Battery
Load Capacity	800mA
Charging current	400mA (Max)
Operating Temperature	- 40°C to 85°C
Storage Temperature	-55°C to 150°C

CAN Data Interpretation

If you are reading the raw CAN data, you will likely see something like this:

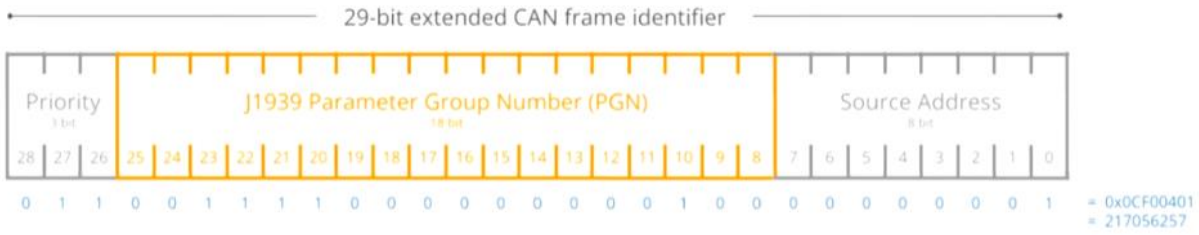
```
r xtd 01 10FAFADB 08 FF FF 7F FC 82 44 FF FF 0   0    319.672754      0.074965 .....D..
r xtd 01 10FAFADB 08 FF FF 7F FC 82 41 FF FF 0   0    319.772770      0.100016 .....A..
r xtd 01 10FAFADB 08 FF FF 7F FC 82 2A FF FF 0   0    319.847743      0.074973 .....*..
r xtd 01 10FAFADB 08 FF FF 7F FC 81 EC FF FF 0   0    319.947759      0.100016 ........
r xtd 01 10FAFADB 08 FF FF 7F FC 81 A8 FF FF 0   0    320.047775      0.100016 ........
r xtd 01 10FAFADB 08 FF FF 7F FC 81 6A FF FF 0   0    320.122773      0.074998 .....j..
```

CanSniff log

The CAN data is generally structured with a 29-bit identifier followed by 1-bit declaring the extended identifier, 1-bit for remote frames, and 8-bytes of data. The data for a specific variable is stored at a specified start bit and requires accounting for little endian and applying a scaling factor.

Priority, PGN, and Source Address

The 29-bit identifier containing information about the priority number (0-7), PGN, and source address.



Taken from css electronics, “J1931 Explained- A Simple Intro [v2.0|2021]”[4]

[4] https://www.youtube.com/watch?v=vlqxu9ojbHg

Priority, PGN and Source Address values for the drawbar pin telemetry system

CAN IDENTIFIER	Priority	PGN	Source Address (SA)
Number of bits	3	18	8
Decimal	4	64250	219
Hexadecimal	0x 4	0x FAFA	0x DB
Binary	0b 100	0b 001111101011111010	0b 11011011

Note: The PGN value has 2 leading zeros for Data Page (DP) and Extended Data Page (EDP) which is typically 00 in J1939. These bits will change the first hex value when combined. (e.g. 0x 4 FAFA DB → 0x 10FAFADB)

Identifier Significance

Priority was chosen between 3 and 4 as a typical value. 4 was chosen to avoid issues with automatic bit reductions. 0b011 is sometimes reduced to a 2-bit value into 0b11, which sometimes affected print statements or conversions. 0b100 (4) is used instead for convenience.

The PGN was selected while referencing PGN spreadsheets. Both "SuperSystem1750Cab_CANSpreadsheetMY23.xlsx" and "SuperSystem_Tractor_DArc_MY25_CAN.xlsx" were used to avoid conflicts on the VCB bus from tractors MY23-MY25 and MY25+. 0xFAFA was chosen to stay within the 65000-64000 range of values (not sure why this matters), but it has no particular significance. I was told that any CAN standard value could also work on the bus as they were not used, but I chose the extended to stay consistent with J1939.

The Sender's Address was selected while referencing the VNSM database to avoid conflicts. 0xDB was chosen as an acronym for "Drawbar" but has not particular other significance.

Decoding

For decoding, the data for a specific variable is stored at a specified start bit. Please note that if the amplifier is calibrated directly on a load cell instead of a power supply, then the Scale, Offset, Unit, and Range parameters do not apply, and are entirely dependent on the reference value used.

If calibrated with power supply	Load Cell 1	Load Cell 2
Bit Start	16	32
Length (bytes)	2	2
Scale	0.01	0.01
Offset	327.67	327.67
Unit	mV	mV
Range	±20	±20
Accuracy[5]	±5%	±5%
Resolution	0.01	0.01

e.g. to find the value of the second load cell in “FF FF 7F FC 82 44 FF FF”:

1) Find the fifth byte and count 2 bytes: “82 44”.
2) Convert to decimal: 33348.
3) Apply scaling factor: 333.48
4) Subtract the offset: 333.48-327.67 = 5.81mV

Note: if reading a value from a CAN BUS, you might have to flip the bytes due to little endian encoding. So “44 82” becomes “82 44”.

Error Value

An output of 65535 or 0xFFFF denotes an overflow error. This is implemented in the Receiver_code.ino’s “encodeAndSend” function.

[5]Variance in amplifier readings are said to be ±5% https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide#resources-and-going-further

Andy Dequin
DequinAndy@JohnDeere.com

Troubleshooting and Testing

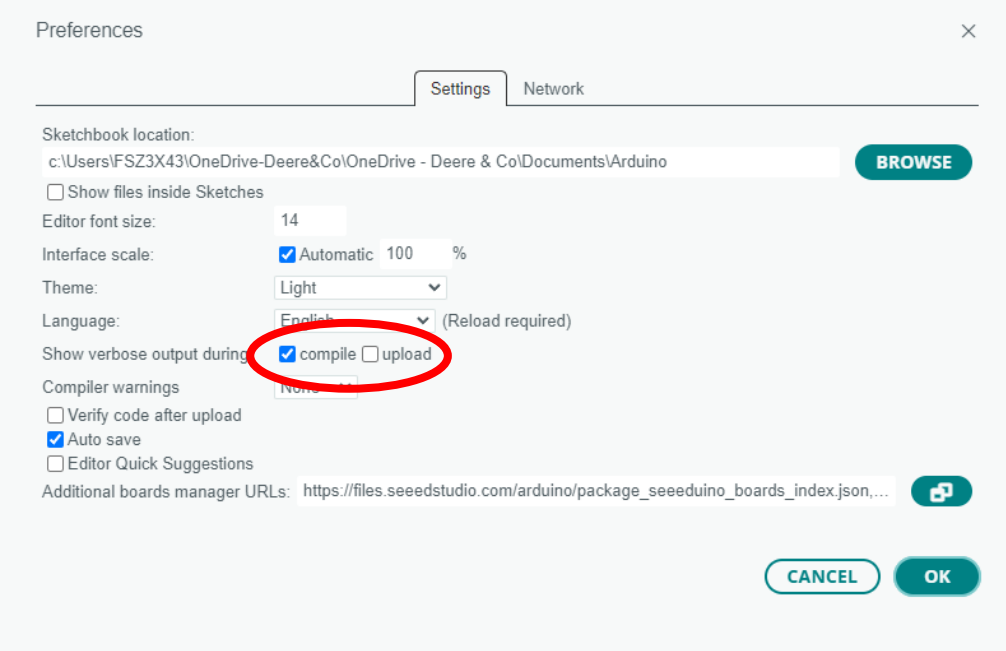
Arduino Issues

Arduino Compiler

If you get a compilation error or exit status 1 message:

1. Check your syntax. The error message should be indicative of the problem.

2. If the error message is not helpful, go to File>Preferences and check “show verbose output during compilation.

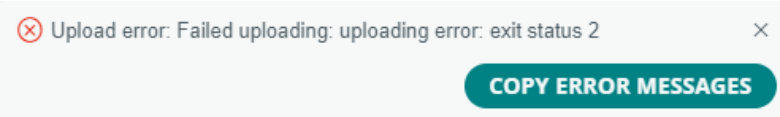


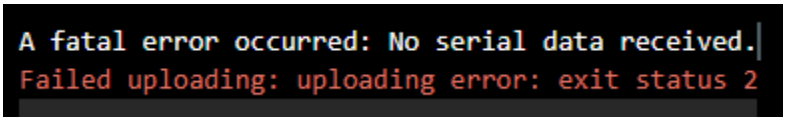
3. Some obscure weird path related issues like “partition.csv not found” may indicate that you have “ESP Family Device” as the board instead of “XIAO_ESP32C3”. If that does not fix it, try restarting Arduino or your computer.

4. If you are using the Serial_CAN_Module library, make sure you fixed the mistakes mentioned in the Arduino Libraries section of the firmware setup.

5. Otherwise, consider asking ChatGPT or DeereAI to read your code and error.

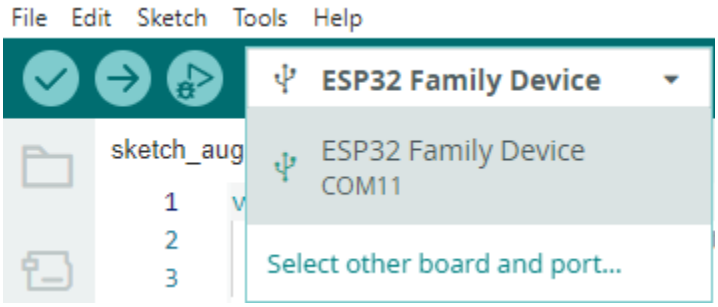
Arduino Upload



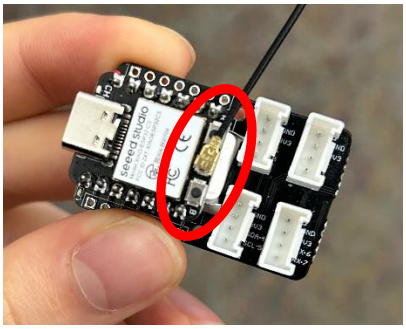


If you see these messages, there are a couple of things to check:

1. Make sure your board is plugged into the laptop.
2. Check Port and Board type. Board should say “XIAO_ESP32C3” and not the autogenerated “ESP32 Family Device”, and port should be the only available port, for example “COM11 Serial Port (USB)”. This error often happens when switching between different boards.



3. The antenna cable should not be holding down the reboot (B) or reset (R) buttons on the microcontroller.

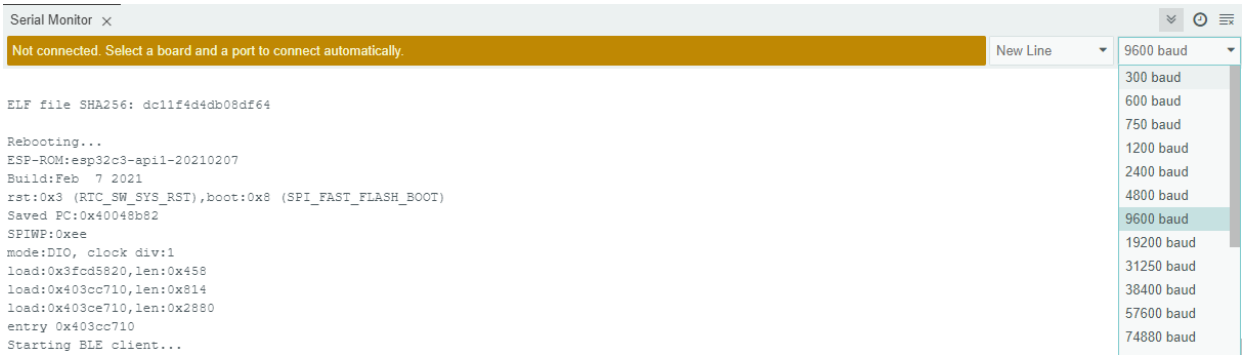


4. Otherwise, try **rebooting**. Hold down the R and B buttons (reset and reboot). Release R, then B. (source: it worked)
   You can also reboot by holding down B before plugging into your laptop, then release. Rebooting with both buttons seems to work in cases where rebooting with just B does not, however.

Arduino Serial Monitor

If the Serial Monitor is not printing out anything when you expect it to or printing something you don't expect, there are a few things to check:

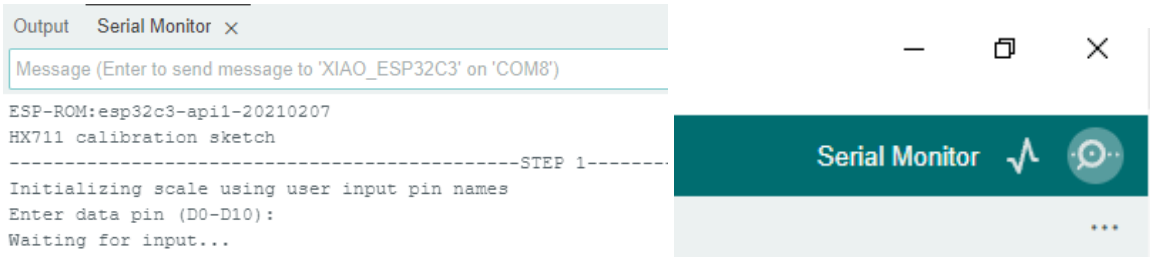
1. Check the baud rate in Serial.begin("rate"); is the same as the one you are using in the Serial Monitor.



2. Close Serial Monitors open across different Arduino windows.

3. If your loop is simple (e.g. Serial.println(analogRead(A0));), it may be running too fast for the Serial Monitor to print out. You can add a delay using delay() to fix this.

4. For programs based on Serial availability (e.g. while(!Serial);), close and open the Serial Monitor again to trigger the start of the program. If still nothing shows up, press enter into the Serial Monitor message box.



5. If you are doing divisions or conversions with integers and floats, make sure your operation is compatible or your values may turn into 0 (use floats for divisions).

6. If you are trying to interface with the Serial CAN Module or the HX711, go to their respective sections for more specific potential issues.

7. Otherwise, consider resetting the board, unplugging, power cycling, closing Arduino, switching boards, asking ChatGPT to read your code, etc.

XIAO Microcontroller

There are a lot of potential issues when using these microcontrollers that I had to figure out the first time. Here is a summary:

1. If you have not soldered the header pins to the microcontroller, you may experience inconsistent connections. You may have to apply pressure for a consistent connection.

2. Make sure to understand pin naming (explained in Glossary here)

3. Strapping pins are a thing. Refer to this. What this means is that if you apply voltage (0V or 3V3) to specific pins of the XIAO on startup or during upload, this might trigger unexpected behavior. You may need to reboot if this happens.

4. Power pins. If you want to power the board by connecting to the 5V and GND, you may fry the board if you accidentally flip the polarity or connect via USB at the same time. Refer to this for power pins, and this for battery usage. The grove shield manages all of this for us.

5. Don't use A3 (GPIO5) for analog reads. This is also from the pinout diagram from seeed studio.

6. On the ESP32, analogRead() is out of 4095 due to using a 12-bit ADC instead of a usual 10-bit ADC (1023)

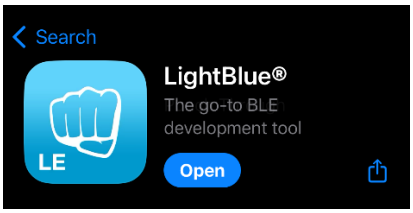
7. LED indicator does not light up when battery powered on the XIAO ESP32C3

8. Make sure to select the right board, port, and have the right cores installed (i.e. for ESP32c3 adding this link to the Additional Board Manager URLs). More details here.

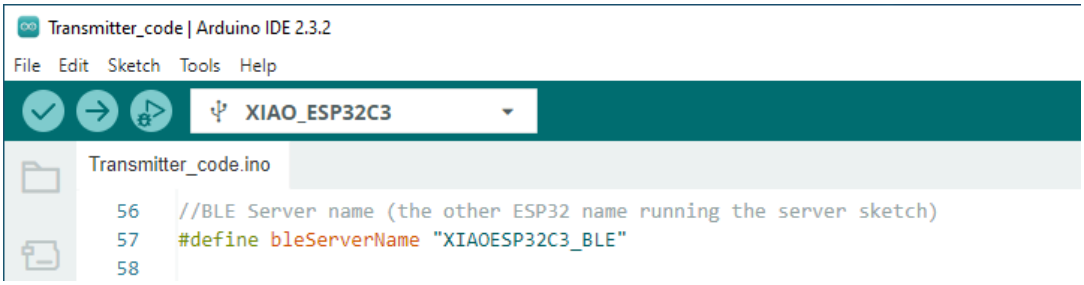
Bluetooth Debugging

For testing Bluetooth capabilities of the XIAO ESP32C3, you can use the BLE_Server.ino code in the Test Code section which will broadcast the mV reading from the A0 pin. Seeed studio also has a lot of sample code for connecting XIAO microcontrollers together (Bluetooth Usage | Seeed Studio Wiki).

I use the LightBlue app on my phone to verify that the server is broadcasting. You can find it in the App Store.

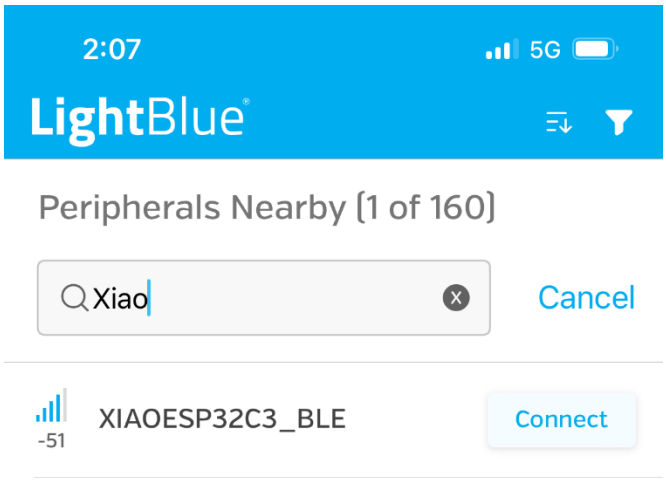


Once the transmitter is turned on and the code is running, you should be able to find the board using the "XIAOESP32C3_BLE" name. This is defined in Transmitter_code.ino.



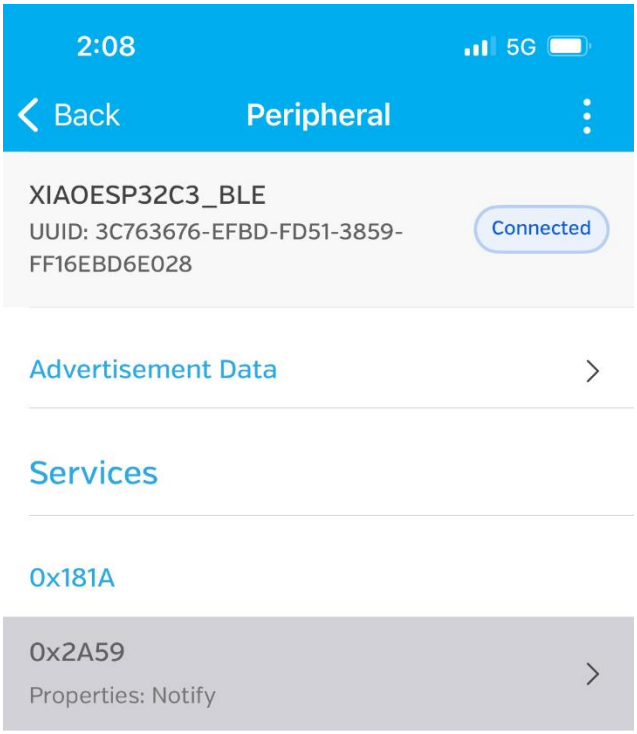
Open the LightBlue app and look for "xiao". If the server is broadcasting, you should be able to 1) find it, 2) click on the characteristic "0x2A59", and 3) subscribe to receive value notifications.

1) Find the transmitter and connect

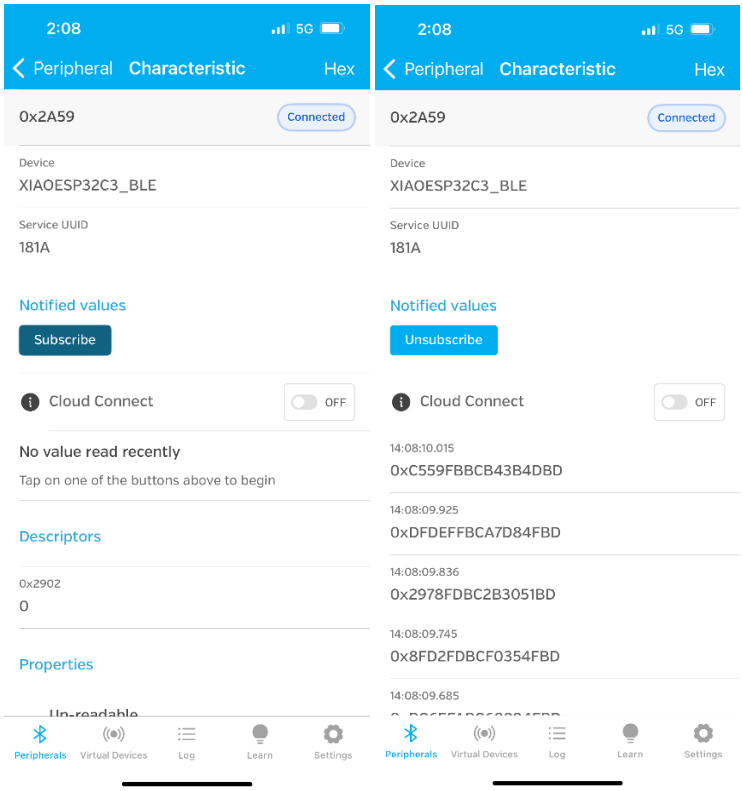


Andy Dequin
DequinAndy@JohnDeere.com

2) Open the 0x2A59 characteristic



3) Subscribe to the receive the values as notifications



Andy Dequin
DequinAndy@JohnDeere.com

Sparkfun HX711

For testing the Sparkfun HX711, I used a multimeter, power supply and banana cables with hooks to hook into the HX711 board. Instead of a power supply you could also use a breadboard, jumper wires, and a Wheatstone bridge from 3 resistors and a potentiometer.

As a very basic test, you can leave the inputs volatile and make them react to your finger touch. Touching both A- and A+ inputs with your fingers should change the values so you can tell values are being read.

Try using uploading amp_test.ino and read some values.

Here are some Sparkfun HX711 issues that I ran into. These are especially relevant if you know that the HX711 is likely to be the issue.

1. Check the code and make sure you have the right clock pin and data pin set up. This is explained in the Load Cell Amplifier Setup section

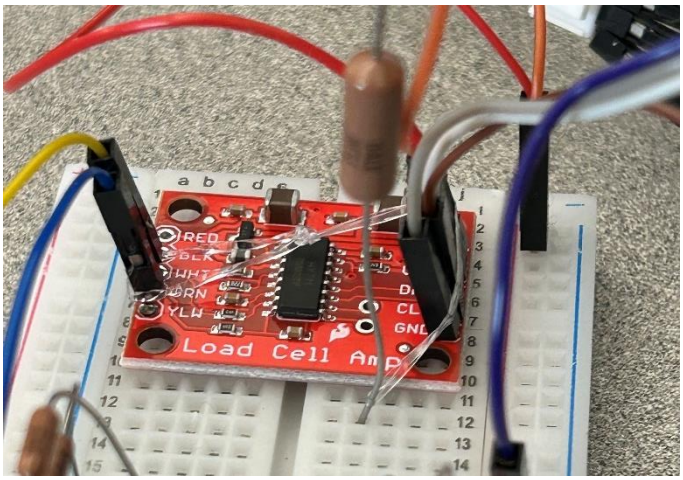
2. If you are using an HX711 library, make sure you are using the one that Sparkfun uses by bogde and not the default library in Arduino. This is also what the Sparkfun example code uses.

3. The HX711 amplifies, but also converts the reading into a digital value, so you can't just use analogRead() to get the value. Using analogRead in the same loop with .get_units() seems to break something as well.

   Use the .get_units() method to read values with scaling and offset. Raw readings can be huge numbers. Its normal to have a large offset and scaling factor.

4. Check hardware connections. Make sure the grove connectors are properly connected.
   If testing on a breadboard with jumpers, you need to create some tension for the jumpers to contact the board properly. You can do this with your fingers or something elastic.

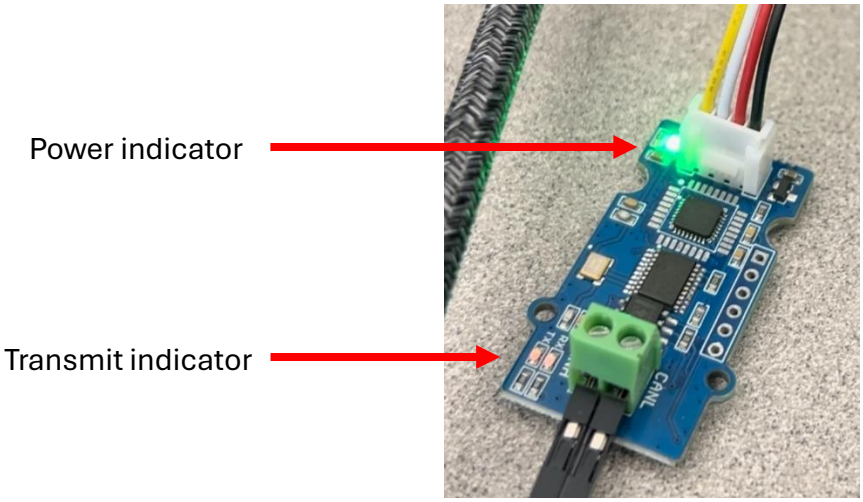
Andy Dequin
DequinAndy@JohnDeere.com



Andy Dequin
DequinAndy@JohnDeere.com

Serial CAN Module

For testing the Serial CAN Module, before getting CAN reading equipment and software, you can look at the indicators to see if the module is powered properly and transmitting/receiving signals.



If you can get the transmit indicator to either flash red or stay solid red, then you can move on to trying to read the CAN output.

I don't know too much about the equipment for reading CAN output so I won't go into detail, but here is what I used:

- CanSniff software (John Deere internal I think)
- DC Power supply (12V)
- CAN Cables to banana
- Banana to hook cables
- Jumpers to connect to the Serial CAN Module
- A CAN adapter (chucky cable)
- A VN1630 CAN Case
- A USB Type-B to USB Type-C cable

There should be other software able to read CAN output like CANalyzer. Using the VN1630 may require downloading drivers or hardware configuration software etc.

Here are some Serial CAN Module issues that I ran into. These are especially relevant if you know that the CAN module is likely to be the issue.

1. Make sure you have the right CAN module! There are a few very similar ones

2. Make sure the Serial CAN module is plugged into the Grove shield or the code will fail.

3. **For the Serial CAN Module, the R button for reset does not work. Pressing it on the XIAO makes the setup code fail (e.g. can.baudRate(SERIAL_RATE_38400))). Instead, you must either unplug and plug it back in, or turn the power off and on EVERY TIME for it to set up properly.**

4. Serial_CAN_Module library errors, e.g. something like "SoftwareSerial already declared", or the send() function never transmitting anything and being stuck in an infinite loop. These errors are due to a missing return statement in the library function, and a compatibility issue with using the ESP32 implementation of SoftwareSerial. Make sure to follow the steps in Arduino Libraries to fix these mistakes in the library.

5. Transmission speed is very important. If your CAN readings start and stop suddenly, you may be transmitting too fast. Test out different transmission rates like SERIAL_RATE_38400 and SERIAL_RATE_19200 (these are set up as global variables in the Serial CAN library). If testing using analogRead() values, make sure to add delay() in the code, or it will be too fast to transmit the CAN signal.

6. If testing a more complex pipeline like Bluetooth->CAN, your transmission speed is affected by multiple things. Doing analogRead -> BLE Server -> BLE Client -> CAN38400 or 19200 and adding a delay in the Ble loop or the CAN sending loop doesn't seem to work for some reason.

Unsolved

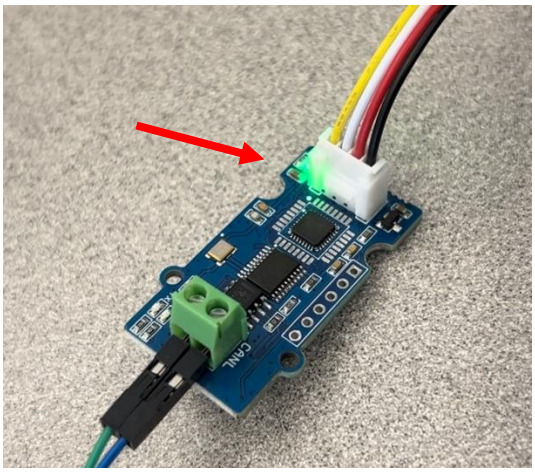
There were 2 issues that came up that I was unable to solve.

1. The XIAO ESP32 death reset cycle

```
Output   Serial Monitor ×
Message (Enter to send message to 'XIAO_ESP32C3' on 'COM5')
entry 0x403cc710
ESP-ROM:esp32c3-api1-20210207
Build:Feb  7 2021
rst:0x3 (RTC_SW_SYS_RST),boot:0x8 (SPI_FAST_FLASH_BOOT)
Saved PC:0x40048b82
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd5820,len:0x458
load:0x403cc710,len:0x814
load:0x403ce710,len:0x2880
entry 0x403cc710
ESP-ROM:esp32c3-api1-20210207
Build:Feb  7 2021
rst:0x3 (RTC_SW_SYS_RST),boot:0x8 (SPI_FAST_FLASH_BOOT)
Saved PC:0x40048b82
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd5820,len:0x458
load:0x403cc710,len:0x814
load:0x403ce710,len:0x2880
entry 0x403cc710
ESP-ROM:esp32c3-api1-20210207
Build:Feb  7 2021
```

   If you see this continuously printing out, something went wrong with the ESP32. Rebooting and reflashing did not seem to do anything

2. Serial CAN module power indicator is stuck blinking (green LED)



   Mine fixed itself after waiting a few hours. No amount of power cycling, changing code, microcontrollers seemed to do anything, however.

Test Code

Additional code files are included in the “test_module” folder to test the basic functionality of each module (Amplifier, Bluetooth, and CAN). These can be helpful to make sure the module actually works and that the issue is probably not hardware related.

The code files and folders need to be downloaded and stored locally first in order to work.

amp_test.ino

```
/*
Simple sketch to test scaling factor and offset value on the Sparkfun HX711 breakout board
The default calibration and zero factor should be close (+/-1mV) for reading mV input in the +/-20mV range
*/
#include "HX711.h"
#define calibration_factor 115994.63 //This value is obtained using the calibration sketch
#define zero_factor 4339 //This value is obtained using the calibration sketch

#define DOUT D4// set the data pin
#define CLK  D5// set the clock pin

HX711 scale;

void setup() {
  Serial.begin(115200);
  scale.begin(DOUT, CLK);
  scale.set_scale(calibration_factor); //This value is obtained by using the SparkFun_HX711_Calibration sketch
  scale.set_offset(zero_factor);  //Assuming there is no weight on the scale at start up, reset the scale to 0
  Serial.println("Readings:");
}

void loop() {
  delay(10);
  Serial.print("   ");
  Serial.println(scale.get_units(10), 3); //scale.get_units() returns a float
}
```

CAN_Communication.ino

```
#include "SoftwareSerial.h"
#include "Serial_CAN_Module.h"

/*
CAN message structure 14 byte frames:
```

```
byte 0-3: ID0-ID3 (CAN ID, ex: 0x3DC -> 0000 0000, 0000 0000, 0000 0011, 1101 1100) 32 bits identifier.
byte 4: EXT (extended? No:0, Yes:1)
byte 5: RTR (request? No:0, Yes:1)
byte 6-13: DTA0-DTA7 (one byte per data, generally only uses the "first" 2 bytes)

Message 4 byte ID organization according to J1939 Standard specifically the /71 application layer
specifications (extended 29 bit ID)
bit 0-2: priority (0 is highest, 7 is lowest)
bit 3-20: PGN (Parameter Group Number: 0xFAFA or 64250, and 2 leading bits set to 0 for EDP and RES)
bit 21-28: SA (Sender's Address: 0xDB or 219)

*/

//define input and output channel values
#define can_tx RX      // tx of serial can module, the yellow cable (TX is mapped to pin 21 in
pins_arduino.h)
#define can_rx  TX    // rx of serial can module, the white cable (RX is mapped to pin 20 in
pins_arduino.h)
Serial_CAN can;
uint8_t load_data;

#define EXTENDED 1
#define NOT_EXTENDED 0
#define REQUESTING 1
#define NOT_REQUESTING 0

#define PRIORITY_BITS 3
#define PGN_BITS 18
#define SENDER_ADDRESS_BITS 8

//define CAN bus frame identifier
#define PRIORITY_VALUE 0b100        //Priority number (0-7), use 3 or 4, no particular reason why
#define PGN_VALUE 0xFAFA  //PGN number for drawbar pin strain data
#define SENDER_ADDRESS_VALUE 0xDB //Sender's Address (0xDB)

//concatenates PRIORITY_VALUE, PGN_VALUE, and SENDER_ADDRESS_VALUE for the 29bit frame identifier
#define CAN_FRAME_ID (((unsigned int)PRIORITY_VALUE << (PGN_BITS + SENDER_ADDRESS_BITS)) | \
                      ((unsigned int)PGN_VALUE << SENDER_ADDRESS_BITS) | \
                      SENDER_ADDRESS_VALUE)

void sendLoadData(unsigned char __load_x, unsigned char __load_y=0xFF) {
    unsigned char __data[8] = {0xFF, 0xFF, __load_x, __load_y, 0xFF, 0xFF, 0xFF, 0xFF}; // data generally
limited to 2bytes per value
    can.send(CAN_FRAME_ID, EXTENDED, NOT_REQUESTING, 8, __data);   // SEND TO ID:0X55
}
```

```
void setup()
{
    Serial.begin(9600);
    while(!Serial);
    can.begin(can_tx, can_rx, 9600);      // set this baudrate to the current baudrate

    Serial.println(can.baudRate(SERIAL_RATE_19200) ? "set UART rate ok" : "set UART baud rate
fail");   //Setup UART baud rate
    Serial.println(can.canRate(CAN_RATE_500)        ? "set CAN rate ok" : "set CAN rate
fail");          // set to 500k which is what the tractor uses, max is 1000k using "18"
}


void loop()
{
  load_data = analogReadMilliVolts(A0);
  delay(1000);
  Serial.println(load_data);
  sendLoadData(load_data);
}
```

BLE_Server.ino

```
//example server code for bluetooth communication between two xiao ESP32c3
//server code modified from https://wiki.seeedstudio.com/xiao_ESP32c6_bluetooth/#ble-sensor-data-exchange
//connect to XIAOESP32C3_BLE on LightBlue app or other bluetooth app to read the data remotely
//or use the BLE_client code on a different xiao ESP32 to print out the readings

#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <BLEServer.h>

//BLE Server name (the other ESP32 name running the server sketch)
#define bleServerName "XIAOESP32C3_BLE"

BLECharacteristic *pCharacteristic;
bool deviceConnected = false;

int mV_input = 0;

class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };
```

```
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};

void setup() {
  Serial.begin(115200);

  BLEDevice::init(bleServerName);
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *pService = pServer->createService(BLEUUID((uint16_t)0x181A)); // Environmental Sensing
  pCharacteristic = pService->createCharacteristic(
    BLEUUID((uint16_t)0x2A59), // Analog Output
    BLECharacteristic::PROPERTY_NOTIFY
  );
  pCharacteristic->addDescriptor(new BLE2902());

  pService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(pService->getUUID());
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x0);
  pAdvertising->setMinPreferred(0x1F);
  BLEDevice::startAdvertising();
}

void loop() {
  if (deviceConnected) {
    mV_input = analogReadMilliVolts(A0);
    pCharacteristic->setValue(mV_input);
    pCharacteristic->notify();
    delay(10); // bluetooth stack will go into congestion, if too many packets are sent
  }
}
```

BLE_Client.ino

```
//client xiao receiving data from other xiao ESP32
//code modified from https://wiki.seeedstudio.com/xiao_ESP32c6_bluetooth/#ble-sensor-data-exchange
//make sure the server xiao is also powered and broadcasting
#include <BLEDevice.h>
#include <BLEUtils.h>
```

```
#include <BLEClient.h>

BLEClient*  pClient;
bool doconnect = false;

//BLE Server name (the other ESP32 name running the server sketch)
#define bleServerName "XIAOESP32C3_BLE"

//Address of the peripheral device. Address will be found during scanning...
static BLEAddress *pServerAddress;

BLEUUID serviceUUID("181A"); // Environmental Sensing
BLEUUID charUUID("2A59");    // Analog Output

char force_val[4095];

//Callback function that gets called, when another device's advertisement has been received
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    if (advertisedDevice.getName() == bleServerName) { //Check if the name of the advertiser matches
      advertisedDevice.getScan()->stop(); //Scan can be stopped, we found what we are looking for
      pServerAddress = new BLEAddress(advertisedDevice.getAddress()); //Address of advertiser is the one
we need
      Serial.println("Device found. Connecting!");
    }
  }
};

void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE client...");

  BLEDevice::init("XIAOESP32C6_Client");

  // Retrieve a Scanner and set the callback we want to use to be informed when we
  // have detected a new device.  Specify that we want active scanning and start the
  // scan to run for 30 seconds.
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30);

  pClient = BLEDevice::createClient();
  // Connect to the BLE Server.
  pClient->connect(*pServerAddress);
```

```
  Serial.println(" - Connected to server");

  // Obtain a reference to the service we are after in the remote BLE server.
  BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
  if (pRemoteService == nullptr) {
    Serial.print("Failed to find our service UUID: ");
    Serial.println(serviceUUID.toString().c_str());
    return;
  }

  // Obtain a reference to the characteristics in the service of the remote BLE server.
  BLERemoteCharacteristic* pCharacteristic = pRemoteService->getCharacteristic(charUUID);
  if (pCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID");
    return;
  }
  Serial.println(" - Found force value characteristics");

  pCharacteristic->registerForNotify([](BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pData,
size_t length, bool isNotify) {
    Serial.println("Notify received");
    Serial.print("Value: ");
    Serial.println(*pData);
    snprintf(force_val, sizeof(force_val), "%d", *pData);
  });

  doconnect = true;
}

void loop() {
  if (doconnect) {
    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    BLERemoteCharacteristic* pCharacteristic = pRemoteService->getCharacteristic(charUUID);
    pCharacteristic->registerForNotify([](BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t*
pData, size_t length, bool isNotify) {
      Serial.println(*pData);
      snprintf(force_val, sizeof(force_val), "%d", *pData);
    });
  }
  delay(1000);
}
```

Future Expansions and Recommendations

Scalability Suggestions

Accepting more inputs

This section outlines a path for modifying the current system to accept more inputs

The current system is currently set up for taking in 2 amplifier inputs. However, the grove shield can technically take up to 4 amplifiers in its small form. In the larger form it can even take up to 8 inputs. In order to implement this, you need to consider the following:

1. Each grove port uses 2 digital pins so there is some overlap. This may or may not affect your ability to use both ports for different amplifiers.

   To use both ports {GND 3V3 1 0} and {GND 3V3 2 1} for example, you may need to have 1 be the clock pin for both amplifiers (which entails flipping the data and clock cables when soldering for one of them). It is also possible that since the microcontroller only takes in data from one at a time, this doesn't matter.

2. In Transmitter_code.ino, you will need to add setup code for the additional amplifiers. This means if you want to add a third one for example, you would need to refactor the code to generalize and encapsulate things in a setup function, or just directly add:
   a. #define scaling_factor3 115994.63
   b. #define offset_value3 4339
   c. #define DOUT3 D4
   d. #define CLK3 D5
   e. HX711 scale3;
   f. load_cell_3.begin(DOUT3, CLK3);
   g. load_cell_3.set_scale(scaling_factor3);
   h. load_cell_3.set_offset(offset_value3);
   i. float force_val_3 = load_cell_3.get_units();
   j. memcpy(byte_array + sizeof(float)*2, &force_val_3, sizeof(float));

   You would also have to change the bit array size to be 4*nb of amplifiers, and the line with setValue() also needs to have 4*nb instead of 8 (careful of the max bluetooth packet size).

3. In Receiver_code.ino you would need to

   a. change the encodeAndSend() function to accept more values, and encode them (note: max 8 bytes of data can be sent CAN frame).
   b. Modify the decoding portion expand the Bluetooth packet into more variables and then input them all into the modified encodeAndSend().

4. Make a new housing model that can fit more than 2 amplifiers.

5. When decoding the CAN data from a Dewesoft Data acquisition device for example, would need to remember where you set the bit starts and ends etc.

Custom Load Cell output

If you want the CAN output to directly correspond to your load cell there are a couple things to consider.

1. Range. You would need to make sure that the range of values being read fits in the space allocated in the BLE transmission (4 bytes each for now).

2. Resolution. The HX711 Sparkfun library automatically reads in the 24-bit reading from the HX711 into a 32-bit float variable. We then convert this float into an unsigned 16-bit integer, which leads to some marginal data loss. If your application is sensitive to this, it would be worth changing the HX711 library or some other workaround.

3. Decoding. You would need to determine your own table of values with all the decoding parameters.

   Currently, the float that is read from the HX711 is multiplied by 100 before converted into an integer in order to preserve 4 significant figures, which, for a range of +/- 20 means that about 4000 values are used out of the 65535 in 2 bytes. This is because the accuracy is about 5%, so having super high resolution is not the most useful.

UI Improvements

As mentioned in the Usage Instructions section, and Future Improvements of the Transmitter Housing section, there are a lot of improvements that remain to be implemented or designed.

Sleep Mode

Instead of manually turning off the transmitter (which would be in the back of the tractor), it would be nice to be able to have the transmitter constantly operating on low power mode and wake up on command.

This would be implemented by creating a “Write” characteristic in the Bluetooth service that the server provides, where you could use the LightBlue app or some other bluetooth application to send specific command words like “SLEEP” and “WAKE” for it to enter or exit low power mode.

Continues After Disconnection

Currently, the transmitter stops after a client disconnects. This means that to reconnect afterwards, the transmitter needs to be restarted first, before broadcasting again.

To implement this, honestly, I would just give the code to ChatGPT or DeereAI and ask it why that isn’t happening. This would likely involve modifying connection flags or adding an outer looping structure.

Andy Dequin
DequinAndy@JohnDeere.com

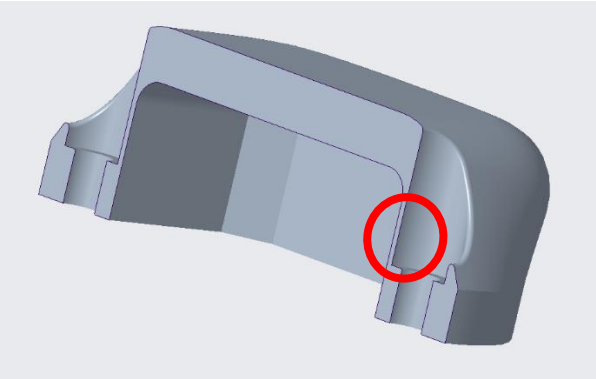
Housing Considerations

Geometric

The housing protrudes a bit on the sides of the pin head for dimensions smaller than 60x60mm which may not be compatible with the hammer strap if the pin head is significantly inset. Some modifications can be made to improve compatibility.

Make the payload a bit taller. 28mm is barely enough to fit everything. 29 or 30mm should do it.

The walls between the screw head hole and the payload are relatively thin. This may or may not be an issue.



Testing

These prototypes remain to be tested and mounted on an actual tractor. All the design requirements should be tested to see what is necessary to improve.

- Signal permeability (antenna external vs internal)
- Waterproof ability (sealing openings and waterproofing treatment)
- Geometric compatibility
- Strength (metal vs PETG)
- Shock absorption for safety of electronics (foam padding)

Andy Dequin
DequinAndy@JohnDeere.com

CAN Communication

Address claiming

One other thing that may or may not matter for integrated testing and operation is address claiming. If this is required for integration with the tractor's controller, then this remains to be implemented in the receiver code.

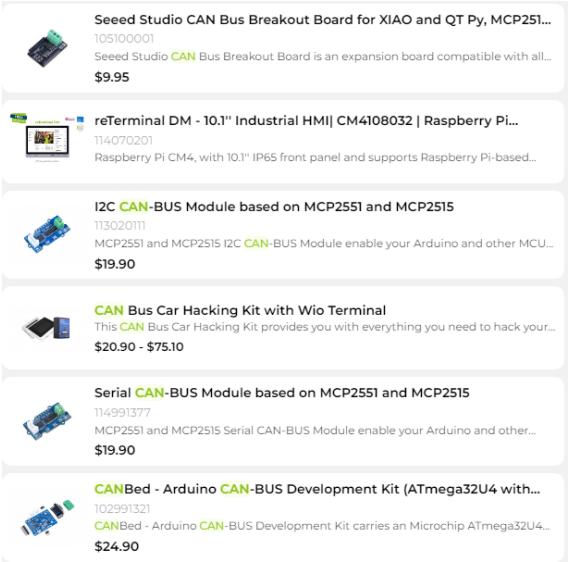
The only indication that I have for what this looks like, is from CanSniff logs where it seemed to happen. It looks like the PDU portion of the PGN becomes EE when this happens. Otherwise, I hope that this is handled internally when new devices broadcast to the CAN bus.

```
r xtd 01 10FAFADB 08 FF FF 03 30 03 49 FF FF 0   0     81.617281      0.012821 ...0.I..
r xtd 01 10FAFADB 08 FF FF 03 2A 03 48 FF FF 0   0     81.620983      0.003702 ...*.H..
r xtd 01 10EEFADB 08 FF FF 03 27 03 4B FF FF 0   0     81.668964      0.047981 ...'.K.. Address Claimed
r xtd 01 10FAFADB 08 FF FF 03 0E 03 48 FF FF 0   0     81.672781      0.003817 .....H..
```

```
r xtd 01 10FAFADB 08 FF FF 03 0C 03 48 FF FF 0   0      8.088076      0.003900 .....H..
r xtd 01 10FAFADB 08 FF FF 03 2B 03 4D FF FF 0   0      8.109187      0.021111 ...+.M..
r xtd 01 10FAFADB 08 FF FF 03 1E 03 40 FF FF 0   0      8.113086      0.003899 .....@..
r xtd 01 10EEFADB 08 FF FF 03 0D 03 4B FF FF 0   0      8.159256      0.046170 .....K.. Address Claimed
r xtd 01 10FAFADB 08 FF FF 03 2C 03 4E FF FF 0   0      8.163106      0.003850 ...,.N..
r xtd 01 10FAFADB 08 FF FF 03 1B 03 4B FF FF 0   0      8.185946      0.022840 .....K..
```

CAN Module and Receiver Improvements

There are many types of CAN modules that can be used to output CAN signal from the XIAO ESP32C3. The "Seeed Studio CAN Bus Breakout Board" has a very small footprint, which may be worth exploring. Some of these may be more reliable as well.



There is no housing design for the receiver either. Since it will be in the tractor cabin, designing protection was less of a priority.

Integrated Testing and Characterization

Integrated testing on a tractor remains to be done. This includes the system's performance as well as the effectiveness of this documentation and the ease of use.

Integration will likely raise new compatibility issues that need to be considered. Once the system is working, it should be characterized as well. Particularly the following should be evaluated,

1. All the code files functionality
2. Ease of installation
3. Ease of use
4. Battery life
5. Signal strength
6. System integrity
7. Housing geometric compatibility
8. Housing strength
9. Etc.

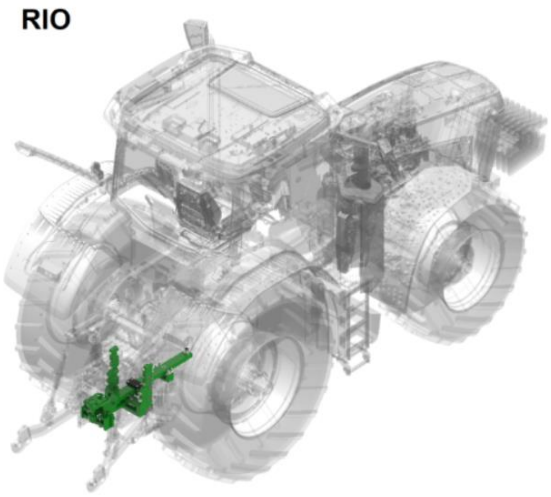
Andy Dequin
DequinAndy@JohnDeere.com

Appendices

Glossary of Terms

Drawbar and Drawbar Pin

Drawbar pin in hammer strap from John Deere Parts Catalog



Location of drawbar assembly on tractor

XIAO Microcontroller Pin Names

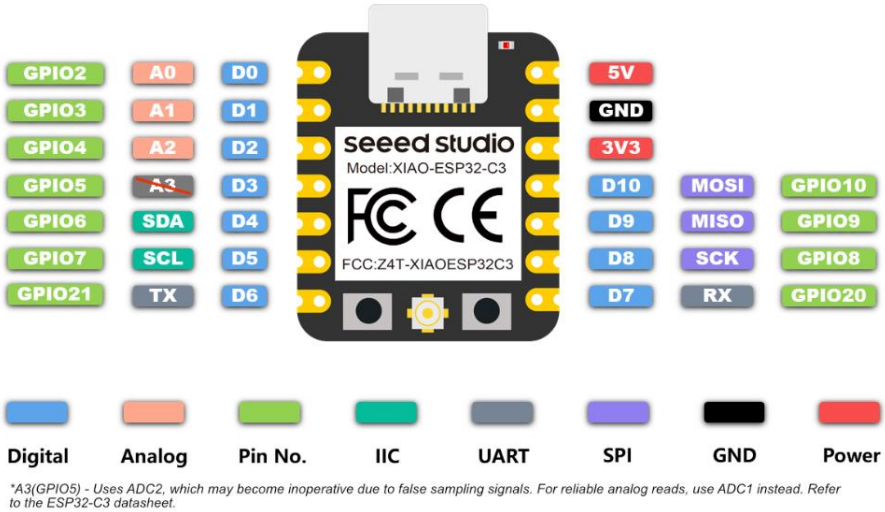


image from the seeed studio website

Each pin on the XIAO microcontroller can be referred to by multiple names. The name we are generally looking for is “D” for “Digital” pin, because we intend to read the value of the pin as either high (3.3V) or low (0V).

If we wanted to be able to read intermediate values like 1.2V, we would look for pins with an “A” for “Analog”. If we wanted to use different communication protocols like SPI or IIC, we would look for the respective names.

In Arduino, a handy pin mapping file (pin_arduino.h) has already been made for us so that even if use number “6”, “D4”, or “SDA”, it all maps back to the same pin on the core ESP32 chip (XIAO SAMD21 does not have this however).

External Links

Mouser Links

- 103020016 Seeed Studio | Mouser
- 474-SEN-13879
- 103020312 Seeed Studio
- 114991377 Seeed Studio
- ASR00012 TinyCircuits
- 110990036 Seeed Studio | Mouser

Product Information

- Cost Effective RISC-V MCU with Wi-Fi, BLE - XIAO ESP32C3 (seeedstudio.com)
- Grove - CAN BUS Module based on GD32E103 - Seeed Studio
- Grove Shield for Seeed Studio XIAO with embedded battery management chip - Seeed Studio
- Lithium Ion Polymer Battery 3.7V 1000mAh | TinyCircuits.com
- SparkFun Load Cell Amplifier - HX711 - SEN-13879 - SparkFun Electronics

Guides

- Getting Started with Seeed Studio XIAO ESP32C3 | Seeed Studio Wiki
- SERIAL CAN BUS MODULE - Longan Docs (longan-labs.cc)
- Grove Shield for XIAO with battery management chip | Seeed Studio Wiki
- Lithium-ion Polymer 1000mAh Battery Datasheet (mouser.com)
- Load Cell Amplifier HX711 Breakout Hookup Guide - SparkFun Learn

Other Components

- 103020016 Seeed Studio | Mouser
- 102010388 Seeed Studio | Mouser
- FIT0096 DFRobot | Mouser
- ICP501421PS Renata
- 113991254 Seeed Studio | Mouser

103030356 Seeed Studio | Mouser
321050009 Seeed Studio | Mouser

John Deere Parts Catalog

8R Drawbar Assembly

Arduino

Software
Installing Additional Arduino Libraries | Arduino

Seeed Studio

XIAO ESP32 Arduino Core URL
Getting Started with Seeed Studio XIAO ESP32C3 (Software) | Seeed Studio Wiki
Grove Shield for XIAO with battery management chip | Seeed Studio Wiki
Bluetooth Usage | Seeed Studio Wiki

Github Libraries

GitHub - bogde/HX711: An Arduino library to interface the Avia Semiconductor HX711 24-Bit Analog-to-Digital Converter (ADC) for Weight Scales.
GitHub - Longan-Labs/Serial_CAN_Arduino
GitHub - plerup/espsoftwareserial: Implementation of the Arduino software serial for ESP8266
GitHub - sparkfun/HX711-Load-Cell-Amplifier at V_1.1

Images

Grove Shield Diagram
XIAO ESP32C3 pinout diagram
Drawbar Assembly Diagrams
CSS Electronics CAN ID Explanation
See documentation folder for remaining images

FINAL CHECK:

Run all the code and compare to code files

Review notebooks and add any relevant bugs to troubleshooting

TO DO

Future enhancement section

Add section on bluetooth connection and debugging (app)

Add reference links

Add a list of acknowledgements?

dimensions, resolution, power, input and output ranges, max voltage, etc. or just link to separate datasheets :P