à la mods

EMBDZ19121

Embedded Wireless Host Controller

User Guide

Revision History

REV	DATE	DESCRIPTION
0	Feb, 2020	Initial release

Disclaimers

Information in this document is subject to change without notice and does not represent a commitment on the part of Made Systems, LLC. Made Systems provides this document "as is," without warranty, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Made Systems may make improvements, updates and/or changes in this manual or in the product and/or program(s) described in this manual at any time.

Copyright and Trademarks

à la mods and the à la mods logo are trademarks in the United States property of Made Systems, LLC. All other trademarks mentioned in this document are the property of their respective owners.

Support

Contact à la mods technical support through our website at www.alamods.com/support.html

Warranty

The à la mods product warranty can be obtained from the website at www.alamods.com/documents/warranty.html

Contents

Table of Contents

List of Tables

List of Figures

Introduction

The à la mods EMBDZ19121 embedded wireless host controller is designed as an embedded alternative host processing unit for the à la mods stacking I/O modules. Although it is designed to control the à la mods modules, it can be used in many other applications.

The EMBDZ19121 is based upon the ESP32 low-power System On a Chip (SoC) with integrated WiFi and dual-mode Bluetooth by Espressif Systems.

The ESP32 supports a host of peripheral I/O including:

- Several serial ports including: UART, SPI, I^2C
- Several PWM channels
- Several Analog channels including: 12-bit ADC & 8-bit DAC
- Several touch sensor inputs

The à la mods module adds many additional features including:

- 40 pin GPIO header (similar to Raspberry Pi ® header)
- GPIO expander to support the additional I/O
- USB UART bridge w/ auto reset compatible with Arduino IDE
- Dual buttons for Reset and Boot
- SD Card slot
- Dual indicator LEDs for Status and Power
- Mechanical dimensions equivalent to Raspberry Pi ® Zero

In addition to the multitude of hardware features expected for embedded system designers, the ESP32 ecosystem includes a development environment framework directly from Espressif Systems (IDF) and is compatible with the Arduino Software IDE. There is extensive documentation on the Espressif Systems website and a large community of users on the Internet providing a full support system.

NOTE:	The à la mods EMBDZ19121 is loaded with a template web-app application from the factory. Source code can be found at the à la mods website within the *Resources* page at: http://www.alamods.com/resources.html/embdz19121app

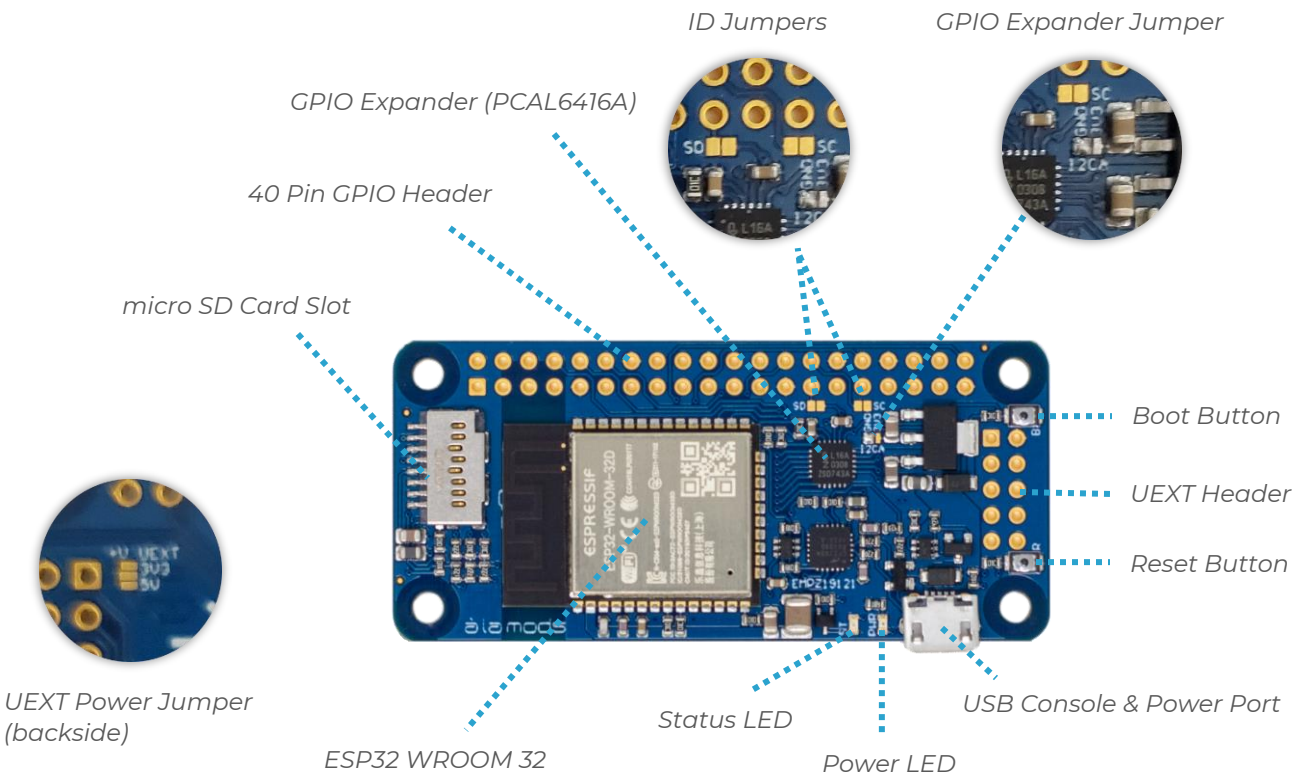


Figure 1 EMBDZ19121 Overview

Mechanical Specifications

The à la mods

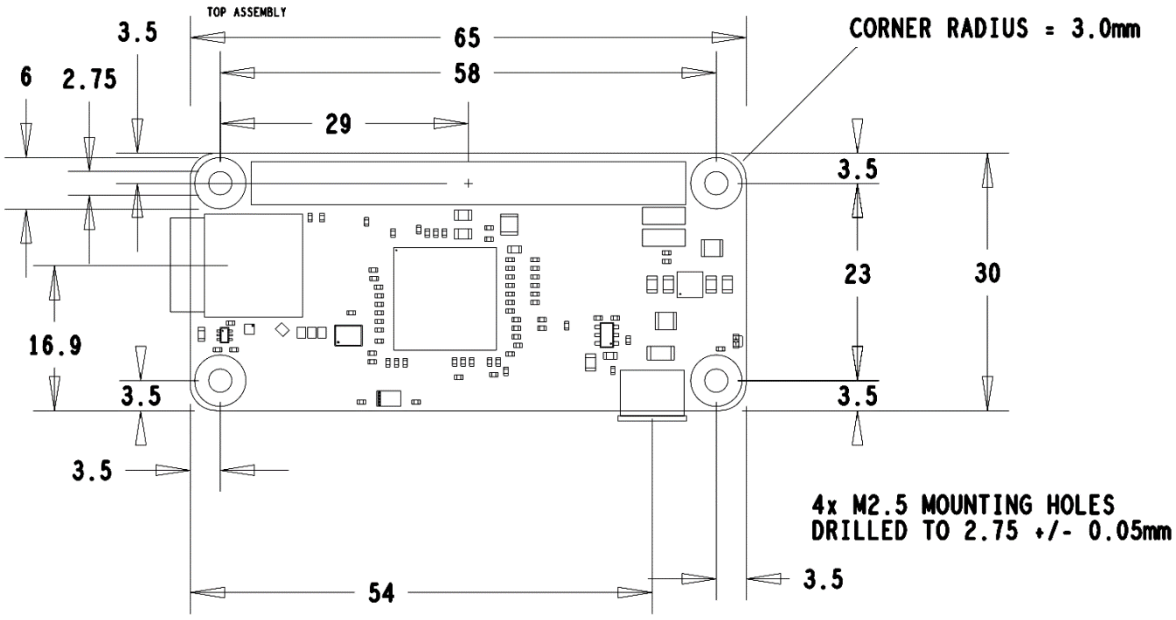


Figure 2 Mechanical Dimensions

Electrical Specifications

The à la

Table 1 Electrical Operating Characteristics

PARAMETER	PERFORMANCE			UNIT
	Min	*Typ*	*Max*	
Input Voltage *	4.75	5	5.25	V
Output 3V pin voltage	3.14	3.3	3.46	V
Output 3v pin current		500		mA
5V Supply Current - Deep sleep mode - Average - RF Transmit peak	10			µA
		90		mA
			500	mA
Operating Temperature	-40		85	°C

* at the USB connector or the 5V pins of the 40 pin header (pins 2 & 4)

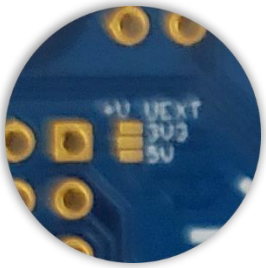
Hardware Configuration Options

Most all a la mods modules have jumper options to adapt the module to specific logic environments. These are special solder bridge and 0.1" header hybrid combination jumpers for added convenience. A simple solder bridge or the addition of a header can be used.

GPIO Expander I²C Address

UEXT Power Option

This option allows the designer to power an external device from the UEXT connector at 5V or 3.3V.



NOTE:	*The I/O signal lines of the UEXT connector are 3.3V compatible only.*

I²C ID 40 Pin Header Lines (pins 27 & 28)

I/O Pinouts

The I/O pins of the 40 pin header map to various I/O on the ESP32 and GPIO expander as shown in Table 2 40 Pin GPIO Header Pin Assignments Table 2 below. The names provide identification to which I/O device and pin each is connected to. Names that are of the form “E32_GPIOxx” are connected to the GPIO label of the ESP32 while names of the form “EXP_Px_y” are connected to the port “Px” and pin “y” of the GPIO expander.

Many lines from the ESP32 are shared such as the serial ports I²C, SPI and UART. All of these lines are connected to the 40 pin GPIO header and also connected to the UEXT port header.

GPIO Header – 40 Pin

Table 2 40 Pin GPIO Header Pin Assignments

PIN	DIRECTION	NAME	DESCRIPTION
1	Power	3V3	3.3 Volt power output
2	Power	5V0	5.0 Volt power input/output [1]
3	I/O	E32_GPIO32 / SDA	GPIO or I²C SDA
4	Power	5V0	5.0 Volt power input/output [1]
5	I/O	E32_GPIO33 / SCL	GPIO or I²C SCL
6	Power	GND	GND
7	I/O	EXP_P0_7	GPIO (expander port 0 bit 7)
8	I/O	E32_GPIO17 / TXD	GPIO or UART TXD
9	Power	GND	GND
10	I/O	E32_GPIO16 / RXD	GPIO or UART RXD
11	I/O	EXP_P0_6	GPIO (expander port 0 bit 6)
12	I/O	E32_GPIO2	GPIO [2]
13	I/O	EXP_P0_5	GPIO (expander port 0 bit 5)
14	Power	GND	GND
15	I/O	EXP_P0_4	GPIO (expander port 0 bit 4)

16	I/O	EXP_P0_0	GPIO (expander port 0 bit 0)
17	Power	3V3	3.3 Volt power output
18	I/O	EXP_P0_1	GPIO (expander port 0 bit 1)
19	I/O	E32_GPIO13 / MOSI	GPIO or SPI MOSI (HSPI ID)
20	Power	GND	GND
21	I/O	E32_GPIO12 / MISO	GPIO or SPI MISO (HSPI Q) [2]
22	I/O	EXP_P1_5 / Status LED	GPIO (expander port 1 bit 5) [2]
23	I/O	E32_GPIO14 / CLK	GPIO or SPI CLK (HSPI CLK)
24	I/O	E32_GPIO15	GPIO or SPI_CE0
25	Power	GND	GND
26	I/O	E32_GPIO25	GPIO or SPI_CE1
27	I/O	E32_GPIO32 / ID_SDA	Jumpered GPIO or SDA (default open)
28	I/O	E32_GPIO33 / ID_SCL	Jumpered GPIO or SCL (default open)
29	I/O	EXP_P0_2	GPIO (expander port 0 bit 2)
30	Power	GND	GND
31	I/O	EXP_P0_3	GPIO (expander port 0 bit 3)
32	I	E32_GPIO34	Input only – digital or ADC1_6
33	I	E32_GPIO35	Input only – digital or ADC1_7
34	Power	GND	GND
35	I	E32_GPIO36	Input only – digital or ADC1_0
36	I/O	EXP_P1_6	GPIO (expander port 1 bit 6)
37	I/O	EXP_P1_4	GPIO (expander port 1 bit 4)
38	I/O	EXP_P1_7	GPIO (expander port 1 bit 7)
39	Power	GND	GND
40	I	E32_GPIO39	Input only – digital or ADC1_3

1 These pins can be used to provide 5V power to the module or provide 5 Volts from the module.

2 These lines are pulled down with a 100k Ohm resistor

UEXT Header

Table 3 UEXT Header Pin Assignments

PIN	DIRECTION	NAME	DESCRIPTION
1	Power	3V3 or 5V0	Power output – jumpered [2]
2	Power	GND	GND
3	TXD	E32_GPIO17 / TXD	GPIO or UART TXD
4	RXD	E32_GPIO16 / RXD	GPIO or UART RXD
5	SCL	E32_GPIO33 / SCL	GPIO or I²C SCL
6	SDA	E32_GPIO32 / SDA	GPIO or I²C SDA
7	SPI MISO	E32_GPIO12 / MISO	GPIO or SPI MISO (HSPI Q) [1]
8	SPI MOSI	E32_GPIO13 / MOSI	GPIO or SPI MOSI (HSPI ID)
9	SPI CLK	E32_GPIO14 / CLK	GPIO or SPI CLK (HSPI CLK)
10	SPI SSEL	E32_GPIO15 / SSEL	GPIO or SPI_CE0

1 These lines are pulled down with a 100k Ohm resistor
2 **CAUTION!** The I/O lines (pins 3-10) are not 5V tolerant. External level shifting is required if setting pin 1 to 5V.

SD Card Slot

Table 4 SD Card Slot Pin Assignments

PIN	DIRECTION	NAME	DESCRIPTION
1	Pull-up	DAT2	2.2k pull-up resistor
2	I/O	E32_GPIO5 / CD/DAT3	10k pull-up resistor
3	I/O	E32_GPIO23 / CMD	10k pull-up resistor
4	Power	Vdd	3.3V
5	I/O	E32_GPIO18 / CLK	Clock signal
6	Power	Vss	GND
7	I/O	E32_GPIO19 / DAT0	DAT0 signal
8	Pull-up	DAT1	2.2k pull-up resistor
9	Pull-down	DET	100k pull-down resistor

Programming

The à la mods EMBDZ19121 has two common development environments that can be used to develop and load (flash) programs into the ESP32 device. Espressif Systems provides a complete IoT framework and toolchain and they have also provided a board package that can be loaded into the Arduino Software IDE.

There are many good tutorials on the Internet that walk through the setup and programming process step-by-step.

Both environments will require a USB (micro-b to A) cable.

ESP-IDF Programming Environment

This environment entails loading the Espressif Systems ESP-IDF toolchain. The official Espressif documentation can be found at:

https://docs.espressif.com/projects/esp-idf/en/release-v3.3/get-started/index.html#introduction

Arduino Software IDE

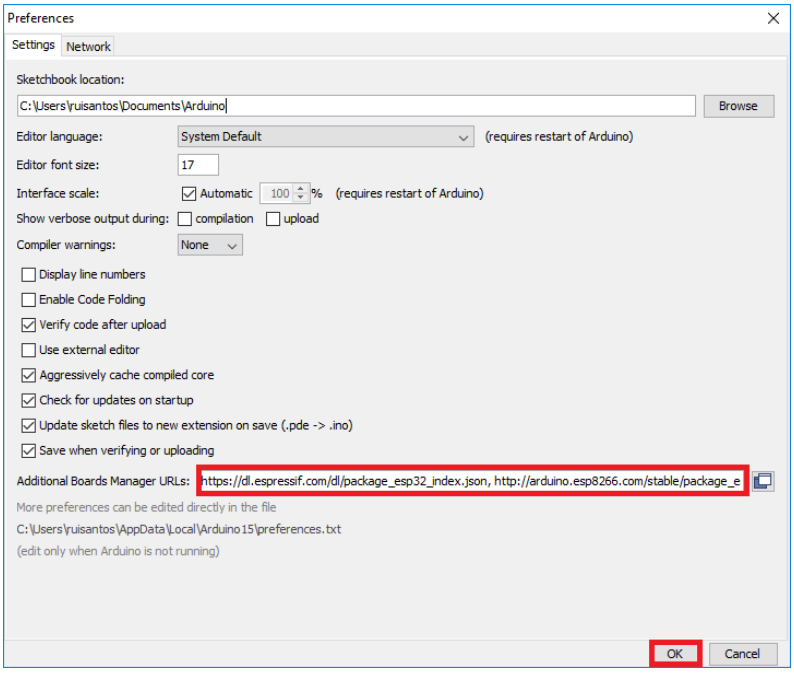
1. Load the latest version of the Arduino Software IDE

   This can be found at https://www.arduino.cc/en/main/software

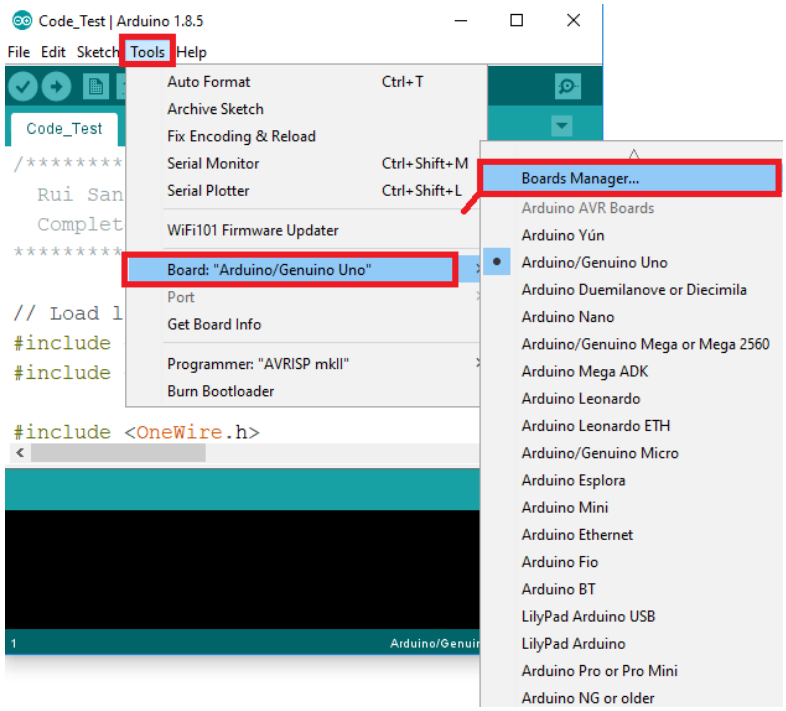
2. Install the ESP32 board package into the Arduino IDE

   a. Go to **File > Preferences**

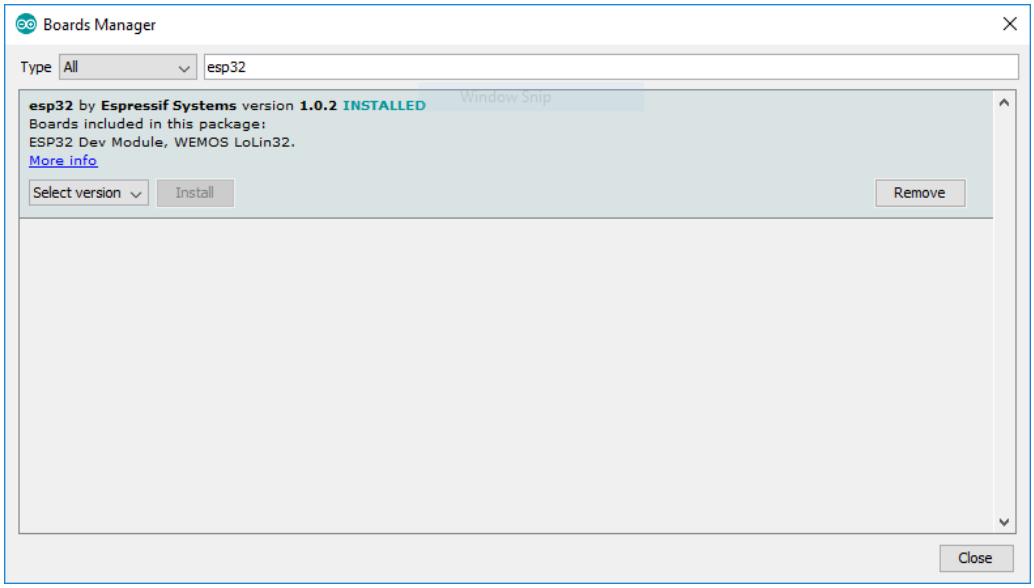
   b. Enter https://dl.espressif.com/dl/package_esp32_index.json into the "Additional Board Manager URLs" field.as shown below. Then select "OK".



c. Open the board manager. Go to **Tools > Board > Boards Manager...**



d. Search for ESP32 in the search field. Then select **Install**



e. The ESP32 environment should be installed in a few seconds.

3.

Module Operation

The à la mods EMDZ19121 host controller has a flexible I/O peripheral system that can be configured through software. Specific I/O pin capability such as the serial ports (I^2C, SPI and UART) are setup to map to the 40 pin GPIO header that is consistent with the Raspberry Pi® 2/3/4 header configuration, but it is completely up to the developer how the I/O configuration will function.

The à la mods EMDZ19121 is intended to be an embedded host processor option for the à la mods peripheral module family.

ESP32 WROOM 32 Microcontroller

The ESP32 WROOM 32 is the heart of the EMBDZ19121. This microcontroller module supports an extensive array of features including:

- Integrated WiFi & dual-mode Bluetooth
- Dual Xtensa LX6 processor cores
- Adjustable clock from 80 to 240 MHz
- Low Power co-processor to monitor specific peripherals
- Rich set of peripherals
  - Multiple serial ports (I^2C, SPI, UART)
  - Analog 12-bit ADCs & 8-bit DACs
  - Touch sensor inputs
  - Hall sensor input
  - SD card interface
  - Multiple PWM channels
- Well supported development environment including RTOS
- Volumes of documentation and large community support

For more detailed information regarding the ESP32 WROOM process consult the datasheet and documentation on the Espressif Systems website at –

https://docs.espressif.com/projects/esp-idf/en/release-v3.3/

GPIO Expander

The GPIO expander device is the NXP PCAL6416HF,128. This is a 16-bit general-purpose I/O expander that provides additional I/O via an I^2C bus. It provides additional I/O features such as programmable output strength, latchable inputs, programmable pull-up/pull-down resistors, maskable interrupt, interrupt status register, programmable open-drain or push-pull outputs. See PCAL6416A product data sheet for complete details.

SPI Port

SPI port setup code -

```
#include “SPI.h”
#include “Wire.h”

Static   const  int              spiClk                    = 1000000;
         union SPI_MSG_t         spiRxBfr;
         union SPI_MSG_t         spiTxBfr;

                  // setup the SPI structure pointer
SPIClass                         *spi                      = NULL;

                  // setup the SPI select lines
         pinMode(15, OUPUT);
         digitalWrite(15, HIGH);
         pinMode(25, OUPUT);
         digitalWrite(25, HIGH);

                  // instantiate the SPI object
         spi                                                = new SPIClass(HSPI);

                  // initialize the SPI port
         Spi->begin(14,12,13,15);
```

SPI port usage code –

This example simply sends a single byte out of the SPI bus and receives a single byte.

```
spi->beginTransaction(SPISettings(spiClk, MSBFIRST, SPI_MODE0));
                  // enable the slave select line SPI_CE0
digitalWrite(15, LOW);
                  // load the SPI bus message into the TX buffer
spiTxBfr.b[0]            = 0xaa;          // example byte to send
spiRxBfr.b[0]            = spi->transfer(spiTxBfr.b[0]);
```

```
                    // disable the slave select line
digitalWrite(15, HIGH);
spi->endTransaction();
```

This next example sends two à la mods commands to a connected à la mods **smart module** to retrieve its model number.

> *For more details regarding the à la mods module SPI bus communication structure go to the "Resources" page on the à la mods website.*

This first section sends the command to setup the module internal read address register. The data in the spiRxBfr will be discarded from this command because the data returned may not be from the internal register that is needed.

```
spi->beginTransaction(SPISettings(spiClk, MSBFIRST, SPI_MODE0));
                    // enable the slave select line SPI_CE0
digitalWrite(15, LOW);
                    // load the SPI bus message into the TX buffer
spiTxBfr.b[3]       = 0x01;         // msB  (bus message command byte –
                                            0x01 = read address register setup command)
spiTxBfr.b[2]       = 0x80;                 (bus message register address byte)
spiTxBfr.b[1]       = 0x00;                 (bus message data high byte)
spiTxBfr.b[0]       = 0x00;         // lsB  (bus message data low byte)
                    // send and receive bytes
spiRxBfr.b[3]       = spi->transfer(spiTxBfr.b[3]);
spiRxBfr.b[2]       = spi->transfer(spiTxBfr.b[2]);
spiRxBfr.b[1]       = spi->transfer(spiTxBfr.b[1]);
spiRxBfr.b[0]       = spi->transfer(spiTxBfr.b[0]);
                    // disable the slave select line SPI_CE0
digitalWrite(15, HIGH);
spi->endTransaction();
```

This next section sends the command to read the module internal register with the address setup in the previous section. The data returned and stored into the spiRxBfr will be the desired information.

```
spi->beginTransaction(SPISettings(spiClk, MSBFIRST, SPI_MODE0));
                    // enable the slave select line SPI_CE0
digitalWrite(15, LOW);
                    // load the SPI bus message into the TX buffer
spiTxBfr.b[3]       = 0x00;         // msB  (bus message command byte –
                                            0x00 = read command)
spiTxBfr.b[2]       = 0x80;                 (bus message register address byte)
spiTxBfr.b[1]       = 0x00;                 (bus message data high byte)
spiTxBfr.b[0]       = 0x00;         // lsB  (bus message data low byte)
```

```
                    // send and receive bytes
spiRxBfr.b[3]               = spi->transfer(spiTxBfr.b[3]);
spiRxBfr.b[2]               = spi->transfer(spiTxBfr.b[2]);
spiRxBfr.b[1]               = spi->transfer(spiTxBfr.b[1]);
spiRxBfr.b[0]               = spi->transfer(spiTxBfr.b[0]);
                    // disable the slave select line SPI_CE0
digitalWrite(15, HIGH);
spi->endTransaction();
```