Week 4 Lab 2: *Communications!*

Introduction

In today's lab, we have three main **goals** that we would like to achieve. The first goal is to solder our ADS1115 Analog-to-Digital Converter (ADC) to the pins that come with it. The second goal is to learn about how to wire up and then communicate with the **ADS1115**. The third goal is to try out the internet communication abilities of our **WeMOS**!

Your lab tutor will come to your table and bring you to the soldering stations to complete **Task 1**. In the meantime, you can try out **Task 2**. We only have a limited number of stations so this may take a little while.

Your tasks for today are the following:

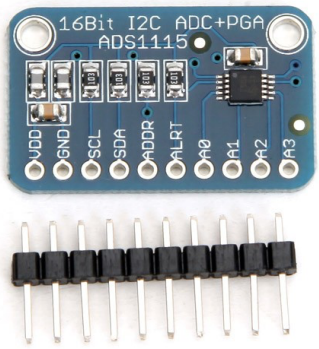
T1. No matter what, we must *solder* on!
T2. Trying out the Joystick
T3. Using I2C to directly communicate with our ADC
T4. Advanced Joystick Control
T5. A tiny little webserver
T6. WeMOS, phone home

T1. No matter what, we must *solder* on!

Our WeMOS board has some limitations, and one of these limitations is the number of analog-in pins. It only has one, and that analog-in can only work within the range of 0 to 3.3v (the operating voltage of the ESP8266). For your project you may require more pins so today we are going to get our ADC board working.

You may have noticed the two other boards in those little anti-static bags that we have provided you. One of the boards is the **MPU9250**, an Inertial Measurement Unit (IMU) for measuring motion, and the other board is the **ADS1115,** an Analog-to-Digital Converter that will give our WeMOS four extra analog-in pins, that work within a range of 0 to 5v (rather than 0 to 3.3v).

You may also notice that the boards, do not have handy pins soldered to them for plugging into a breadboard. Luckily, the pins have been included in the package and we just have to connect them up.

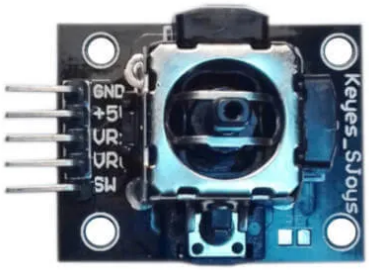


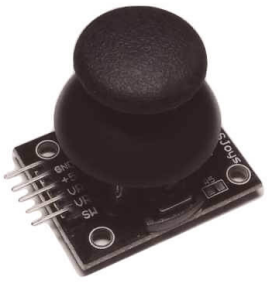
So, today, you will be doing a little soldering! Don't worry if you have not soldered anything before. I will give a safety briefing at the start of class, and then you will be taken to the soldering stations group by group.

You can find a video showing you the process of soldering this component on Canvas, but we will go through it during the lab brief as well.

T2. Trying out the joystick

In your box of sensors is a Joystick. This Joystick consists of two independent potentiometers (oversimplified, but you can think of a potentiometer as a device that allows us to measure a change in voltage as the joystick moves), one for each degree of freedom the Joystick has. That is, one for measuring how far it is moved up and down (y-axis), and one for measuring how far the joystick is moved left and right (x-axis). The joystick also has a clickable button. Neat.





This joystick has 5 pins we need to keep track of:

Joystick Pin	Purpose	WeMOS pin
GND	Ground	GND
+5v	Power Supply	3.3v (??)
VRx	Voltage Proportional to X axis	??
VRy	Voltage Proportional to Y axis	??
SW	For the click button state	??

So to use the Joystick we just need to measure the value of VRx and VRy to figure out which way the joystick is being pushed. But we have a problem, we only have one analog-in, and it only can give us measurements between 0 and 3.3v. No matter. Let's partially wire up our Joystick by connecting the Joystick **GND** to **GND**, **+5v** to **3.3v** (don't worry this is ok), and **VRx** to **A0**.

We will now use the built-in Analog-to-Digital Converter on our WeMOS by using the **analogRead** function. Copy the following program into a new Arduino sketch and try it out:

```
void setup() {
  Serial.begin(9600);
  pinMode(A0, INPUT);
}

void loop() {
  int value = analogRead(A0);
  Serial.print("Value : ");
  Serial.println(value);
  delay(1000);
}
```

Try out the Joystick, move it left to right and see what happens and then move it up and down. Change the **VRx** wire to be connected to **VRy** and try again. What do you notice?

You should see numbers between 0 and 1023. What are these numbers? Well, the built in ADC is 10 bits, so when it measures/samples a voltage on **A0**, it will return a number between 0 and 2^{10}-1 (i.e. 1023) that is proportional to the voltage on **A0**. So here, 1023 is 3.3v, and 0 is 0v, therefore 512 is approximately 1.65v.

Lab Report Task 1: What value do you get on VRx when it is at rest, pushed up, down, left, and right? Give both the reported value and convert it by hand to Volts. Do the same for VRy. (1 marks)

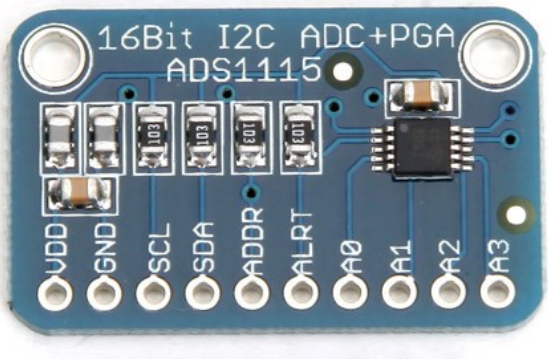
Optional Task: What happens if instead of connecting the Joystick to the 5v on the WeMOS? What numbers do we get back on the ADC? Why does this happen? Why does the Joystick still work even when connected to 3.3v (**Hint:** think about how potentiometers work)?

But we can't read both at once. So… not enough analog-in pins and an ADC that only allows us to convert a maximum of 3.3v. Indeed, we are quite limited *currently*. **Baddum-tish**. Sorry.

Anyway, here comes our *I2C 4 Channel 16-bit ADC* to the rescue. But you might need to solder it first!

T3. Using I2C to directly communicate with our ADC

Hopefully, by now you have soldered the pins to your ADC board. Now, let's take a good look at our board. It is a 4 channel ADC, which means it has four analog-in pins, **A0**, **A1**, **A2**, and **A3**. Also note that is says it is 16-bit, unlike the 10-bit ADC we used in Task 2. That means that it will return a value to us between 0 and $2^{16} - 1$ (or 65535) which is proportional to a voltage on one of the AX pins between 0 and VDD (with the default setup).



Notice that there is an **SCL** and an **SDA** pin. Does this ring any bells? Correct, this is an I2C device, and we can use I2C to communicate with it. But first let's wire up out device.

ADS1115 Pin	Purpose	WeMOS pin
GND	Ground	GND
VDD	Power Supply	3.3v
SCL	I2C Serial Clock	SCL
SDA	I2C Serial Data	SDA
ADDR	I2C Slave Address Select	Not needed
ALRT	Comparator output or conversion ready	Not needed

We can also connect our Joystick **VRx** to **A0** and **VRy** to **A1** on the ADS1115 and use it measure both **VRx** and **VRy** at the same time! Let's check to make sure it is working by using an I2C scanner (one can be found in Appendix 1 at the end of the lab sheet). If this works, it should tell you the address of your I2C component. Write it down!

We are going to use the library Wire which lets us communicate with I2C devices: https://www.arduino.cc/reference/en/language/functions/communication/wire/

The way that we will interact with the ADS1115 is by interacting with the chips internal registers, namely, writing the **Config** register, and then reading from the **Conversion** register using I2C. We are going to follow the quick start guide on page 35 of the ADS1115 Techsheet (with some changes) to read only one of the analog-in pins.

1. Write to Config register:
   - First byte: (7-bit I2C address followed by a low R/W bit)
   - Second byte: 0b00000001 (points to Config register)
   - Third byte: 0b11000000 (MSB of the Config register to be written)
   - Fourth byte: 0b10000011 (LSB of the Config register to be written)
2. Write to Address Pointer register:
   - First byte: (first 7-bit I2C address followed by a low R/W bit)
   - Second byte: 0b00000000 (points to Conversion register)
3. Read Conversion register:
   - First byte: (7-bit I2C address followed by a low R/W bit)
   - Second byte: the ADS111x response with the MSB of the Conversion register
   - Third byte: the ADS111x response with the LSB of the Conversion register

Let's first understand what the config register does. Take a look at the ADS1115 Techsheet (**Hint:** page 28-29), and identify the configuration that we have sent in the above register:

MSB: 0b11000000
LSB: 0b10000011

Which analog-in pin will the above read from (note that these pins are different to the A0 on the WeMOS)? Which Joystick pin does that analog-in pin connect to? **VRx** or **VRy**?

Note in the program on the next page, I have written all numbers in binary notation, rather than in hexadecimal which is more common and concise. Feel free to change it back ☺

Copy the following program into a new sketch and try it out:

```
#include <Wire.h>
byte address = <put the I2C address here>;
uint8_t MSByte = 0, LSByte = 0;
uint16_t regValue = 0;

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}

void loop()
{
    Wire.beginTransmission(address);
    int error = Wire.endTransmission();
    if (error) {
    }else {
      Serial.println("Success");
      Wire.beginTransmission(address);
      Wire.write(0b00000001); // Config Register
      Wire.write(0b11000000); // MSB of Config Register
      Wire.write(0b10000011); // LSB of Config Register
      Wire.endTransmission();

      Wire.beginTransmission(address);
      Wire.write(0b00000000); // Conversion Register
      Wire.endTransmission();

      Wire.requestFrom(address,2);
      if(Wire.available()){
        MSByte = Wire.read();
        LSByte = Wire.read();
      }
      regValue = (MSByte<<8) + LSByte;
      // How do we get this result below?
      float result = (regValue * 1.0 / 32768) * 6.144;
      Serial.print("Register Value: ");
      Serial.println(regValue);
      Serial.print("Volts: ");
      Serial.print(result);
      Serial.println("v");
    }
    delay(1000);
}
```

Check that the Joystick is behaving correctly. Move it and confirm that the ADC is correctly changing based on your input.

Change this program, to use the other channel of your Joystick. **Hint:** your only need to change one value in the above program.

Lab Report Task 2: Explain the meaning of the config register values that we sent in the first version of the program. What analog-in pin is it allowing us to read from? How did you change the program to read from the other analog-in pin? Describe or show your change. (2 marks)

Optional Task: Try changing the I2C address of the ADS1115.

T4. Advanced Joystick Control

Now as you can see, all this can be quite tedious to do. It took me quite some time to get the previous example working for your lab. Let's do ourselves a favour and use a library. For a lot of the sensors/actuators you have, libraries already exist which you are free to use in your projects. Though you should reference them in any report you write up.

We are going to use an ADS1115 library called ADS1115_WE. You know the drill by now:
Tools > Manage Libraries > Search: ADS1115_WE > Install

Now, let's open the example called Continuous:
File > Examples > ADS1115_WE > Continuous
Have a look through it and try running it on your WeMOS. Much easier no?
It gives you the results of all four analog-in pins.

Ok now, let's edit this program to be more useful for us:
1. It should only give us the output of the two analog-in pins we are using.
2. Write a function to determine which direction the stick is being pushed in (At Rest, Up, Up-Right, Right, Down-Right, Down, Down-Left, Left, and Up-Left), and print out that direction.
3. In addition, the Joystick also clicks in as a button, you should detect this state and print it out as well.

Lab Report Task 3: Describe your changes briefly in your lab report. Write a function to determine which direction the stick is in. Include this function and any relevant code in your report with a small description. (3 marks)

Optional Task: Can you get your Joystick to control something on your computer? Pong perhaps?

T5. A tiny little webserver

Now for a change of pace. Let's try out the HTTP functionalities of our little WeMOS. Open the WebServer.ino file and read through and try and see how it works. For this task, you will need to use your Smartphones mobile hotspot. You will see the SSID and password variables right at the start of the program, and these should be your Smartphone credentials. Edit these and try and get your WeMOS to connect to your Smartphone. While it is trying to get connected, it will print ".". If it connects it should look something like this:

```
Connected to Note10+AP
IP address: 192.168.37.15
MDNS responder started
HTTP server started
```

You now need to get your laptop to connect to the same network (your smartphone). Once connected you can try opening a web browser and connecting to the WeMOS's IP address. In the above example, it is 192.168.37.15. You should see a "WeMOS on the Web!" message. Now try to go to another endpoint or route, such as "inline", by pointing your browser to 192.168.37.15/inline (with your WeMOS IP address).

Let's try creating new end point (toggleLED). This endpoint should toggle an LED on and off (you can use an external LED or the built in one). It should also send back the LED status ("LED On" or "LED Off"). That is, when you go to <ip-address>/toggleLED, the WeMOS should:

1. Toggle an LED
2. Report the LED status back to the web browser

So now we have seen some idea of how to affect actuation over the internet, but this is rather primitive. We will come back to this in our backend lab later in the semester.

Lab Report Task 4: Describe your changes briefly in your lab report. If you defined any other function or made source code changes please include it in your report. (2 marks)

T6. WeMOS, phone home

Ok one last task for this lab! Let's try sending information to a simple server running on your computer from the WeMOS device. Again, your laptop and your WeMOS device both need to be connected to your smartphone hotspot. For this task we are going to use a simple python HTTP framework/server called **flask**. You will need to install **python3** on your laptop and then use pip to install flask (`pip install flask`). If you are having trouble, ask!

Have a look at the small_webserver.py script. You can see that it defines routes/endpoints in a similar way to the webserver in Task 5. Here we only have one endpoint, called "data". You can also see that it is expecting a payload (data1 and data2). We are going to use a HTTP GET request to send a payload from the WeMOS device to your laptop server.

One you have python and flask installed you can run the script:

```
python small_webserver.py
```

You should see something like:

```
* Serving Flask app 'small_webserver' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:3237
 * Running on http://192.168.37.119:3237 (Press CTRL+C to quit)
```

Ok great. Now we need to get your WeMOS device to use an HTTP Client to connect to our webserver. To do this open the Http_Request.ino Arduino sketch. Again, you will have to specify the SSID and password for your hotspot, and now you will also have to include your laptops IP address. Try this program out. If successful, you should see data being printed out like so every 5 seconds:

```
Data from WeMOS:  31.2 76.0
```

But where does this data come from? Well let's look into the source of Http_Request.ino.

You can see the example HTTP GET payload in this line:

```
"data?data1=31.2&data2=76"
```

This payload is defined as the following:

```
<name of the route>?<var_name>=<value>&<var_name>=<value>&..
```

Here we are sending two values 31.2 and 76. However, they are currently hardcoded.
Pick a sensor and send its data back to your laptop using the HTTP Client. Which sensor is up to you!

Lab Report Task 5: Describe your changes briefly in your lab report, which sensor you used, and describe how you send the data back to the laptop. (2 marks).

Now, we have only looked at a very simple way to send data from your WeMOS to the Cloud (or rather your laptop). We will cover more on this in a later part of the module!

Appendix 1: I2C Scanner

```
#include <Wire.h>

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}

void loop()
{
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");
      nDevices++;
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```