



6 – Arduino based Coax tester

This project will provide a basic continuity tester and ohmmeter. It will give us the opportunity to explore how to construct an Arduino project including logic, hardware, and software.

Contents

THEORY OF TESTING PROCEDURE	1
BILL OF MATERIALS	3
START WITH THE ESP32 MICROPROCESSOR.....	4
ADD A DISPLAY	5
ADD THE TESTING COMPONENTS.....	6
INSTALL LIBRARY	6
PROGRAM THE CONTINUITY TESTER.....	8
HOW TO TEST A COAX CABLE	8
CODE WALKTHROUGH	9
WHAT NOW?.....	13
APPENDIX	14

THEORY OF TESTING PROCEDURE

Have you tried to test a coax cable? How did you do it?

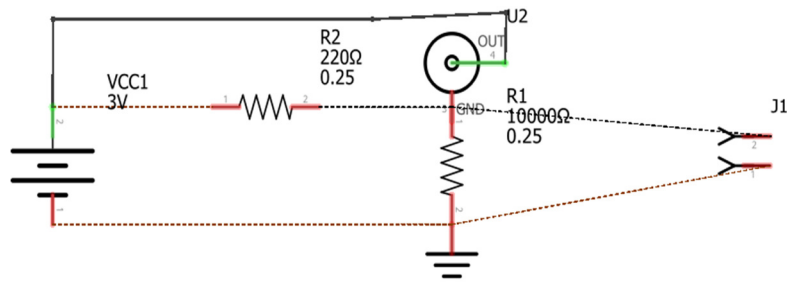
When I have tested a coax cable, I use a multimeter. I test that there is no continuity between the center conductor and the shield or ground conductor, and then I test that there is a low resistance between the center conductor on one end and the center conductor on the other end, and a low resistance between the shield on one end and the shield on the other end.

This is fine if the two ends of the coax are near enough to each other that my test leads can stretch between the ends of the coax. But what can I do if one end of the coax cable is in my shack and the other end is outside at the antenna? As long as you can reach each end of the coax cable then you can use a COAX TESTER like the one we are going to build. You may have to take down your antenna enough to reach the end of the coax cable, or climb up your tower and reach the end that way.

This circuit will provide a way to test for an open condition and a short condition, and some measure of something in-between using Ohm's Law.



6 – Arduino based Coax tester



fritzing

Resistor R2 is not an actual resistor but is the resistance across the U2 coax connector.

If we connect a test point between the ground terminal of the coax connector and the negative battery lead, we can measure the voltage across the R1 resistor. If the coax cable connected to the coax connector is good, that is, not shorted or with some continuity between the two sides of the cable, and not connected at the far end of the coax cable, then the voltage across R1, or Vout, is 0 and this represents an open circuit across the coax cable. R2 would be infinite and we would call this OPEN.

If we connect the inner conductor to the outer shield of the coax cable at the far end of the coax cable, then Vout is 3.3v, from our ESP32. R2 would be 0 and we would call this SHORT.

We can use these two test results to determine if our coax cable is good.

But what if the coax cable is compromised and the resistance is neither zero nor infinite? We can use Ohm's Law to compute R2. The formula is

$$V_{out} / V_{in} = R1 / (R1 + R2)$$

Cross multiply:

$$V_{out} * (R1 + R2) = R1 * V_{in}$$

Divide both sides by Vout:

$$R1 + R2 = (R1 * V_{in}) / V_{out}$$

Subtract R1 from both sides:

$$R2 = ((R1 * V_{in}) / V_{out}) - R1$$

Simplify the right side:

$$R2 = (V_{in} / V_{out} - 1) * R1$$



6 – Arduino based Coax tester

Our sketch computes R2 and displays both R2 and Vout. We would need a very precise Vout to accurately measure R2 when it was either near zero or infinity, and unfortunately, the analog-to-digital converter in our ESP32 is not precise enough, but it does give us an approximation.

BILL OF MATERIALS

Let's gather the components needed to build our Coax Cable Tester.

- HiLetgo ESP-WROOM-32 ESP32 ESP-32S Development Board, available at Amazon for \$11. https://www.amazon.com/HiLetgo-ESP-WROOM-32-Development-Microcontroller-Integrated/dp/B0718T232Z/ref=sr_1_3



<< Upload button

<< Reset button

- Computer to host software and power the ESP32
 - Windows, Mac, Linux, including Raspberry Pi, are supported
- USB cable to connect them together: MicroUSB to USB-A
https://www.amazon.com/dp/B0719H12WD/ref=dp_iou_view_item?th=1
- 10 uF electrolytic capacitor to make it easier to upload a sketch to the ESP32.
https://www.amazon.com/Projects-Radial-Electrolytic-Capacitor-10uF/dp/B07YBCF9NG/ref=sr_1_9
- Breadboard and jumper wires: This is a possible set but any will do.
From Amazon for \$11: DEYUE 3 Set Standard Jumper Wires Plus 3 Set of Solderless Prototype Breadboard 830 tie Points Breadboard | 3 Set of M/F, M/M, F/F - Each 40pin Electronic Jumpers Wire
<https://www.amazon.com/Standard-Jumper-Solderless-Prototype-Breadboard/dp/B07H7V1X7Y/>
 - Breadboard full size
 - Jumper wires
 - Male-male if all parts are on breadboard
 - Male-female if some parts are not on breadboard
 - Female-female if all parts are not on breadboard
 - Or you can just use wires that you cut, strip, and insert into the breadboard
- Display:



6 – Arduino based Coax tester

HiLetgo 1.3" IIC I2C Serial 128x64 SSH1106 OLED LCD Display

<https://www.amazon.com/HiLetgo-Serial-SSH1106-Display-Arduino/dp/B07BHHV844/>

These displays use an SSH1106 chip and are compatible with the Adafruit-SH110X library.

- A 10 K ohm resistor (If you don't have one, don't go buy just one. I have a bag of them.)
- Two female coax connectors to match with the coax cable we want to test. If you want SO-239 connectors, you can use <https://www.amazon.com/DHT-Electronics-Female-Chassis-Connector/dp/B071KK22V2/>
If you have clipleads (<https://www.amazon.com/WGGE-WG-026-Pieces-Colors-Alligator/dp/B06XX25HFX/?th=1>), you can use them to connect the connectors to the breadboard.

OR You will need to solder wires to these connectors. I will have a soldering iron at the workshop if you don't have one. For one of the connectors, you will need to connect the center conductor to the outer conductor. For the other connector, you will need to solder wires onto the two contacts so that you can connect each to the breadboard.

START WITH THE ESP32 MICROPROCESSOR

For this project, we use the ESP32 Microprocessor.

If you have not yet installed the Arduino IDE program on your computer, see the steps in the Appendix to do so.

These steps will connect your ESP32 to your Arduino IDE so we can begin to develop our project.

1. Plug your ESP32 board into your computer using a USB cable. Its red power LED should light.
2. Open the Arduino IDE application that you just installed.
3. Select "File > Preferences"
4. In the Additional Boards Managers URL field, enter https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json on one line. You may need to enter this link manually so it is formatted properly.
5. Select "Tools > Board > Boards Manager..."
6. Scroll down to "esp32". If it is not yet installed, click Install. Wait for download and install to complete then click Close.
7. Select "Tools > Board > ESP32 Arduino > ESP32 Dev Module". This loads the code needed for this particular processor, and is a necessary step for each different type of Arduino.



6 – Arduino based Coax tester

8. Determine what port your ESP32 is connected to. In Windows, right-click Start, select Device Manager. Select Ports (COM & LPT). Look for Silicon Labs CP210x and note the COM port number. In my case, it was COM4.

NOTE: If you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers from:

<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>

Click on Downloads and download the driver appropriate for your operating system environment. For Windows, use the CP210x Universal Windows Driver. After downloading, unzip the zip file and run the appropriate exe, probably

CP210xVCPInstaller_x64.exe

for a 64 bit Windows 10 system.

9. In Arduino IDE, select "Tools > Port > (the COM port number from step 12)"

NOTE: If you don't see the Port option, your firewall may have blocked the port manager. If so, allow "serial-discovery.exe" to execute.

10. Test that the ESP32 is properly installed by using the test script at <https://randomnerdtutorials.com/installing-esp32-arduino-ide-2-0/>

ADD A DISPLAY

For this project, we will use a small monochrome LCD display. It has a low power requirement while providing a crisp looking display.

I suggest using the 1.3" display with the SSH1106 driver chip as specified in the BOM.

Here are the connections needed between the ESP32 and the display:

FUNCTION	DISPLAY	ESP32
Ground	GND	GND (device or bus)
Power 3.3v	VCC	3.3V (device or bus)
I2C Clock	SCL	GIOP 22 / Pin 36
I2C Data	SDA	GIOP 21 / Pin 33



6 – Arduino based Coax tester


It is generally easier to connect the GND and 3.3V pins on the ESP32 to the Blue and Red bus pins on the breadboard so they are available for other purposes as we will see.

I2C is the specification that provides a way to connect a processor such as ESP32 to a device such as the Display, It only requires two wires. It can connect multiple devices to our processor, but we only need to connect one device.

INSTALL LIBRARY

You will need to add a library to make the tester display functional. A library is a collection of C++ code that someone wrote to provide a particular capability that you can include in your sketch without you having to write all of the code needed.

When you install a library, several files are included and you will always find at least 2 files in a library. The “.h” file is the “header” file that describes what are all of the functions included in the library. The “.cpp” file expands each function description to include all of the code needed for the function.

To install a library, in the Arduino IDE, select Tools > Manage Libraries. The library manager will appear at the left of the current sketch. You will also see the icon  appear, and you can click this icon to open the library manager.

In the Search box, type the name of the library that you want. We need three libraries for our project. For the I2C communications, we need the Arduino I2C library, called <wire.h>. For the display to provide graphics and text, we need the library <Adafruit_GFX>. The third library we need is <Adafruit_SH110x.h> which provides device specific code to manage the display which includes a 1106 control chip.

If you type in the name of a library, the library manager will show the status of this library. If it is already installed, you will see **INSTALLED** under the library name. If there is a newer version or if you have not installed the library yet, you will see **INSTALL** and you can click this to install this library.

Now you can test that the display works by using the example code included with the Adafruit library:

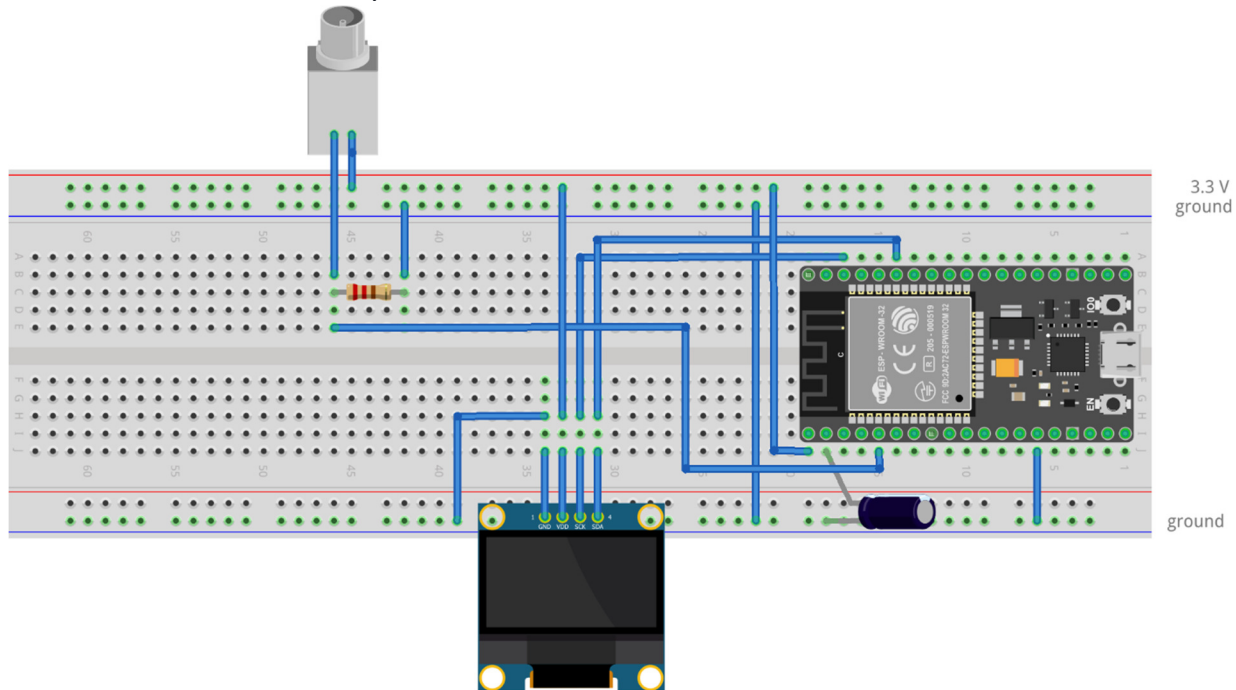
File > Examples > Adafruit SH110X > OLED_QTPY_SH1106 > SH1106_128x64_i2c_QTPY.ino



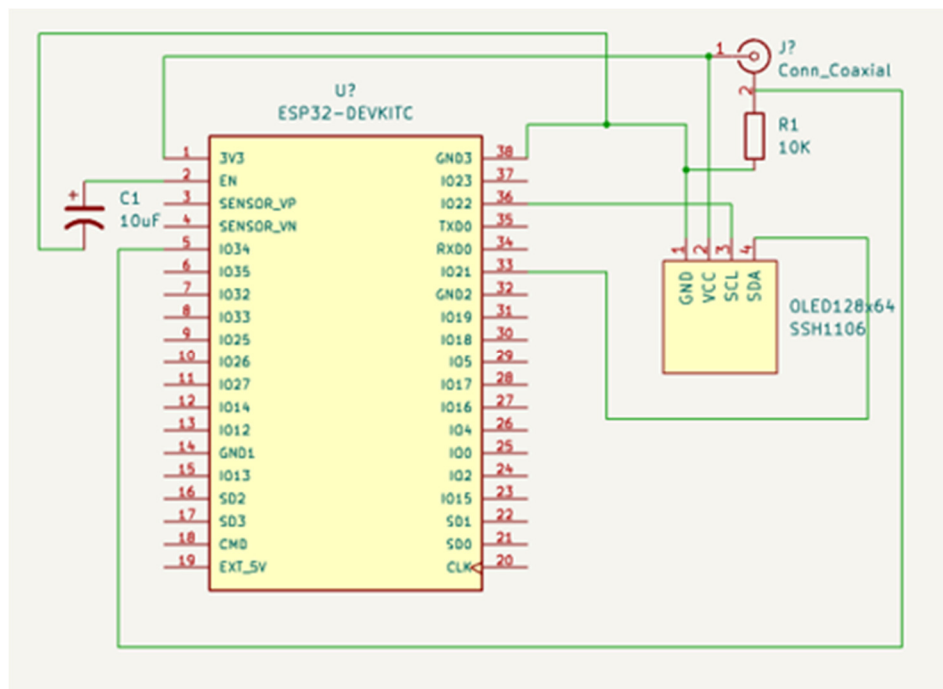
6 – Arduino based Coax tester

ADD THE TESTING COMPONENTS

The circuit for this continuity tester is:



fritzing



The schematic and wiring diagram show a BNC connector to use for connecting to the coax. I could not find a UHF connector in the parts library, but I used an SO-239 connector for my



6 – Arduino based Coax tester

circuit as that is what I use to connect to my coax cables. You can use any type of connector based on what cables you want to test.

The other added component is a 10K Ohm resistor. That is all we need to complete the tester.

PROGRAM THE CONTINUITY TESTER



The circuit includes the ESP32 and the display. It also includes a bridge, or voltage divider, circuit that we connect our coax cable to and it measures the voltage at the outer shield of our coax. It also approximates the resistance of the circuit.

Our sketch, or program, is named

`coaxTester_SSH1106.ino`

and works with the 1.3" display with the SSH1106 control chip.

You can find this sketch at Github.

1. To download it to your Arduino IDE, Navigate to:
https://github.com/hwdornbush/6-CoaxTester/blob/main/coaxTester_SSH1106.ino
2. Find  at the right. Click . This will copy this sketch to your clipboard.
3. In Arduino IDE, click File > New Sketch.
4. A new sketch window will open with a skeleton sketch in it. Delete the skeleton sketch by keying Ctrl-A then Delete.
5. Key Ctrl-V to paste the touch demo sketch into the IDE window.
6. Key File > Save As. Enter "coaxTester-SSH1106" and press Save. This will save this sketch so you can use it again as needed.

HOW TO TEST A COAX CABLE

Now connect a coax cable to the connector attached to the breadboard.

First, we need to test that there is no continuity between the inner conductor and the outer conductor, or shield. The tester displays OPEN.

Next, we can connect the far end of the coax cable to a connector that shorts the inner conductor to the outer conductor and then test the continuity between inner and outer conductor. The tester displays SHORT.

If the continuity is between short and open, the tester will display a RAW number which is between 4095 and 0. A value of 4095 indicates a very low resistance while a value of 0 shows a very high resistance. The analog-to-digital converter in the ESP32 is nonlinear so it cannot detect resistance accurately. The lowest it can detect is about 20 ohms.



6 – Arduino based Coax tester

Wiggle your coax cable connector ends. If you see any change in the display, then you may have a problem with that cable.

CODE WALKTHROUGH

Let's walk through the sketch. It starts with comments describing what we are doing. You should always comment liberally as you develop a sketch so you can recall what you did and can explain it to others.

The sketch starts with comments to describe what the sketch is used for and some information I like to record, such as sources of information I used in my research to build the sketch. All of the text between `/*` and `*/` are treated as comments. You will also see additional comments which start with `/**`

```
/* coaxtester_SSH1106.ino
   Tests coax cable to determine if there is a short or open circuit, or
   something in between
   The in-between is only accurate if the resistance is about the same
   value
   as the resistor in the test circuit.
   AA6BD hwd 2023-08-08
   derived from https://www.circuitbasics.com/arduino-ohm-meter/
   https://github.com/bhall66/LED-
   tester/blob/main/LED\_tester/LED\_tester.ino
   https://roboticsbackend.com/arduino-push-button-tutorial/
   http://www.gammon.com.au/switches pulldown resistor
   https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/
   https://www.youtube.com/watch?v=WJXr9LEE0vk Build a Coax Tester
*/
```

These next statements

```
#include <Wire.h> // Arduino I2C library
#include <Adafruit_GFX.h> // Adafruit graphics
library
//#include <Adafruit_SSD1306.h> // Adafruit OLED
library for SSD1306
#include <Adafruit_SH110X.h> // Library for SH1106
controller
```

include the needed libraries for the display. The commented-out statement is for the SSD1306 display as we are using the SH1106. You will see other comments in the code for the SSD1306 version. Only one of these pairs of statements should be included.



6 – Arduino based Coax tester

The I2C communications protocol supports multiple devices on the bus. The display is typically at address 0x3c but you can use a different address if needed.

```
/* Uncomment the initialize the I2C address , uncomment only one, If you
get a totally blank screen try the other*/
#define i2c_Address 0x3c //initialize with the I2C addr 0x3C Typically
eBay OLED's
//#define i2c_Address 0x3d //initialize with the I2C addr 0x3D Typically
Adafruit OLED's

//if you can't find I2C device, use this code to find valid I2C address
//https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/
```

The next statements define some values for our display: width and height, and include a reset value needed by the library.

```
#define SCREEN_WIDTH      128                // OLED display width,
in pixels
#define SCREEN_HEIGHT     64                // OLED display height,
in pixels
#define OLED_RESET -1                // QT-PY / XIAO
```

This declaration statement creates a display object to be used whenever we want to use the display.

```
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
//Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
// Declaration for an SSH1106 display connected to I2C
Adafruit_SH1106G display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

We have done the setup needed, and now the heart of the code begins. We declare some variables we will be using.

```
int analogPin = 34; //GPIO34 ADC6 physical5 to measure voltage

int raw = 0;
float Vin = 3.3;      // Vin to coax center pin
float Vout = 0;       // Vout from coax ground
float buffer = 0;
float R1 = 10000;     // R1 for open measuring
float R2 = 0;         // R2 resistance of coax
```

After variables and libraries are declared, an Arduino sketch contains two functions named “setup()” and “loop()”. As functions, they must have a “type” because C++ allows you to pass



6 – Arduino based Coax tester

values to a function, but in this case, “void” indicates that nothing will be passed to these functions. The syntax includes () where variables would be if we were passing them in, and an opening brace, {. All statements between the opening { and the closing } are executed when the function is called. Setup() is called once when the sketch starts, and loop() is called right after setup() ends. Loop() is executed to its end and then repeats forever or until power is removed.

Setup() initializes the serial monitor where we will show the progress of the sketch, and initializes the display device.

```
void setup(){
  Serial.begin(115200);           // initialize serial monitor
  delay(250);                     // wait for the OLED to power up
  //display.begin(SSD1306_SWITCHCAPVCC, i2c_Address); //initialize
  SSD1306 display
  display.begin(i2c_Address, true); //initialize the SSH1106 display
  delay(2000);
}
```

The loop() is the rest of the sketch and runs from its beginning to its end and then repeats.

```
void loop(){
```

The display statements call “methods” in the “display” library to do the things we want it to do. In this case, these statements show the title of our device “Coax Tester” at the top center of the display.

```
Display.clearDisplay();
display.setTextSize(1);
//display.setTextColor(WHITE);           // for SSD1306
display.setTextColor(SH110X_WHITE);     // for SSH1106
display.setCursor(20, 0);
// Display “Coax Tester”
display.println(“Coax Tester”);
```

We read the voltage on the analog Pin which we declared above attached to GPIO pin 34. We print that voltage on the serial monitor and then show it on the display. The “print” method will print the value while the “println” method will print the value and then print a Return to the next line.

```
Raw = analogRead(analogPin);           //read voltage
Serial.print(“Raw:” );                  // and display it
Serial.println(raw);
display.setTextSize(1);
```



6 – Arduino based Coax tester

```
display.setCursor(0, 35);  
display.print("Raw: ");  
display.println(raw);
```

We want to know how to interpret the raw voltage. It can range between 0 and 4095 which are the analog to digital values provided by the ESP32. If the value is close to the top of the range, we indicate that the coax center conductor and outside conductor are connected together to form a SHORT circuit. This is because the outer conductor is pulled to the 3.3 volts and grounded through the pulldown resistor.

```
if (raw>4000){ // if coax is shorted,  
  Serial.println("SHORT"); // display this  
  display.setTextSize(2);  
  display.setCursor(30,15);  
  display.println("SHORT");  
}
```

On the other hand, if the raw voltage is near 0, then the outer conductor is pulled to ground because it is not connected to the 3.3 volts through the center conductor. We display this as an OPEN circuit.

```
if (raw<100) { // if coax is open circuit,  
  Serial.println("OPEN"); // display this  
  display.setTextSize(2);  
  display.setCursor(30,15);  
  display.println("OPEN");  
}
```

Now, we do a little Ohm's Law computation, and we convert the analog to digital value to volts, and then compute the resistor value that would provide this voltage. The formula is shown on the first page of this document.

```
buffer = raw * Vin; // compute Vout  
Vout = (buffer)/4095.0;  
buffer = (Vin/Vout) - 1;  
R2= R1 * buffer;
```

Once we compute Vout and R2, we display them.

```
Serial.print("Vout: "); // display Vout  
Serial.println(Vout);  
Serial.print("R2: ");  
Serial.println(R2); // display R2  
display.setTextSize(1);  
display.setCursor(0,45);
```



6 – Arduino based Coax tester

```
display.print("Vout: ");  
display.print(Vout);  
display.setCursor(0,55);  
display.print("R2=");  
display.print(R2);
```

The display does not show anything until we tell it to do so with this statement.

```
display.display();           //display all of the lines
```

We wait for 1000 milliseconds, or one second.

```
delay(1000);                 //wait for a second then go back and do  
it again
```

This closing brace is the end of the loop() function. Once we reach it, we return to the first statement after loop() and execute them again, over and over. This way, our display refreshes once a second to show the value at that time. If we start with our coax cable open, and then short the inner and outer conductors, the display will reflect this.

```
}
```

Sketches can be much more complicated but they all contain these basics: the opening declarations, and the setup() and loop() functions.

The accuracy of the R2 calculation is not very good because we are comparing it to the 10K ohm resistor, and if the value of R2 is not too close to 10K ohms, the value will be quite off. This is due to the analog to digital converter (ADC) only having 4096 possible values and due to the nonlinearity of the converter which is particularly off at the top and bottom of its range. To improve this, we would need to switch the 10K ohm resistor, we use an external A-to-D converter with a greater resolution. Since we are primarily interested in determining if the circuit is OPEN or SHORT, we can accept this inaccuracy.

There are also issues with noise and power supply which affect the accuracy of the ADC.

WHAT NOW?

We could design and build a printed circuit board to make this project more permanent than on a breadboard.

The original article <https://www.circuitbasics.com/arduino-ohm-meter/> used the Ohm's Law approach to sort resistors.



6 – Arduino based Coax tester

You can build a USB CABLE TESTER by adding three more circuits so that you can test all 4 of the wires in a USB cable.

What other uses can you think of for this approach?

How could we improve the precision and accuracy of the resistance measuring circuit?

This approach provides a go-nogo test. Is it shorted or not? This is a good test to determine if you have good or loose connectors on your coax as you can wiggle the connectors while testing.

The test is at DC. To test more thoroughly, we could use an instrument like a NanoVNA to test at various RF frequencies. For example, see <https://www.youtube.com/watch?v=mU71rGUKIBI>

APPENDIX

Follow this procedure to install the Arduino IDE (Integrated Development Environment)

You may find this tutorial useful as it shows screenshots of this process:

<https://randomnerdtutorials.com/installing-esp32-arduino-ide-2-0/>

Follow these steps:

11. On your computer, start your internet browser and go to <https://arduino.cc/en/main/software>
There you will find the Arduino software download choices for Windows, macOS, and Linux. Click on the link for Arduino IDE (V2.x.x) that applies to your development environment. For Windows, use “Windows Win 10 and newer.”
12. These notes are for the Windows 10 and later environment.
13. You will be asked to support the Arduino project. This is optional. Click either “Just Download” or “Contribute and Download”
14. Click on the file name to install. Follow the usual process and take the defaults.
15. Plug your ESP32 board into your computer using a USB cable. Its red power LED should light.
16. Open the Arduino IDE application that you just installed.
17. Select “File > Preferences”
18. In the Additional Boards Managers URL field, enter https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json on one line. You may need to enter this link manually so it is formatted properly.
19. Select “Tools > Board > Boards Manager...”
20. Scroll down to “esp32”. If it is not yet installed, click Install. Wait for download and install to complete then click Close.
21. Select “Tools > Board > ESP32 Arduino > ESP32 Dev Module”. This loads the code needed for this particular processor, and is a necessary step for each different type of Arduino.



6 – Arduino based Coax tester

22. Determine what port your ESP32 is connected to. In Windows, right-click Start, select Device Manager. Select Ports (COM & LPT). Look for Silicon Labs CP210x and note the COM port number. In my case, it was COM4.

NOTE: If you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers from:

<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>

Click on Downloads and download the driver appropriate for your operating system environment. For Windows, use the CP210x Universal Windows Driver. After downloading, unzip the zip file and run the appropriate exe, probably

CP210xVCPInstaller_x64.exe

for a 64 bit Windows 10 system.

23. In Arduino IDE, select "Tools > Port > (the COM port number from step 12)"

NOTE: If you don't see the Port option, your firewall may have blocked the port manager. If so, allow "serial-discovery.exe" to execute.

24. Navigate to <https://randomnerdtutorials.com/installing-esp32-arduino-ide-2-0/>
25. Find "Testing the Installation"
26. Below the code you see, click View Raw Code. A new browser window will open with this code:

```
/*****
  Rui Santos99
  Complete project details at https://RandomNerdTutorials.com/vs-code-
  platformio-ide-esp32-esp8266-arduino/
  *****/

#include <Arduino.h>


#define LED 2

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
}
```



6 – Arduino based Coax tester

```
void loop() {  
  // put your main code here, to run repeatedly:  
  digitalWrite(LED, HIGH);  
  Serial.println("LED is on");  
  delay(1000);  
  digitalWrite(LED, LOW);  
  Serial.println("LED is off");  
  delay(1000);  
}
```

27. Select the code using ctrl-a, then copy the code using ctrl-c
28. In Arduino IDE, click File > New. A new window will open with a skeleton sketch. Delete the code in this window.
29. Press Ctrl-v and the code will be copied into the window.
30. Press File > Save
31. The Arduino IDE will prompt you to name and save the sketch and its directory. Name it "sketch_ESP32blink"
32. In The Arduino IDE, a program is called a "sketch" and has a file type of .ino It is saved in a directory with the same name.
33. In the toolbar, select Upload (the  arrow).
34. The sketch will compile and then try to upload to the ESP32. You can see the progress in the status section at the bottom of the IDE window.
35. When you see "Connecting..." press and hold the button next to the USB port on the opposite side from the red LED, on the right in the above photo of the ESP32, until the IDE shows "Leaving...".
To eliminate having to press this button, connect a 10 uF capacitor between the EN pin and ground.
36. The blue LED should start blinking on and off once per second. If you don't see this, press the reset button, the button on the same side as the red LED.
37. Power can be removed at any time. When it is restored, the program will run, with the blue LED blinking.
38. You can open this sketch later by selecting "File> Sketchbook > sketch_ESP32blink"

Examine the sketch.

1. Comments begin with "//" or are between pairs of "/*" and "*/" so you don't have to enter "//" on each line.
2. After the comments, shown starting at /* and ending at */, you will see

```
#include <Arduino.h>
```

This statement indicates that you want to include a code library named "Arduino.h" with your sketch. We often will include code libraries developed by others so we don't have to code all of the functions ourselves.



6 – Arduino based Coax tester

3. Next, you see

```
#define LED 2
```

This statement defines an integer constant with the name of LED and the value of 2. The reason for 2 is that this is the GPIO pin where the built-in LED is attached on an ESP32. If you read the circuit diagram that shipped with your ESP-32S, the LED section shows the wiring of the LED and shows it connected to IO2.

4. Next, you see

```
void setup() {
```

Note there is no ; at the end of this line as it is a function definition and encloses a set of statements that each end with ;

In the Arduino IDE, the “setup()” function is executed once when the sketch first starts.

This block of code ends with

```
}
```

5. Next, you see

```
void loop() {
```

All statements within this function are executed repeatedly as long as the sketch runs.

When the sketch gets to the } at the end of the block, it goes back to the statement right after:

```
void loop() {
```

6. The setup and loop function definitions have no type, such as integer, so they are defined as a type of void, or no type. We do not need to pass any parameters to them. Some functions expect you to pass a variable to them, which we explain later.
7. In the setup() function, there is a statement

```
pinMode(LED, OUTPUT);
```

This statement sets the mode of the LED pin, which we previously declared as pin 2, to be OUTPUT. We can then set the value of that pin.

pinMode() calls a built-in method to set the mode of the pin. This method expects two parameters, the pin number and the direction, INPUT or OUTPUT.

8. The indents are purely to make the sketch easier to read, and are not required, but they greatly improve the readability of the sketch.



6 – Arduino based Coax tester

9. In the `loop()` function, there is first the statement

```
digitalWrite(LED, HIGH);
```

This statement connects a high voltage, 3.3 volts, to the LED pin so that the LED lights.

This is followed by

```
delay(1000);
```

which delays the sketch for 1000 milliseconds, or 1 second, before moving on.

10. Can you see the purpose of the remaining statements?

11. When the sketch reaches the end of the statements within the

```
void loop()
```

function, it returns to the first statement after the

```
void loop()
```

and continues through the statements – forever or, more likely, until power is removed or you upload another sketch.

12. This two-function structure is the structure of all Arduino sketches written in the C++ language. Note that C++ allows other structures but the Arduino IDE uses this structure exclusively.