Dave's Arduino Kit and merit badge program

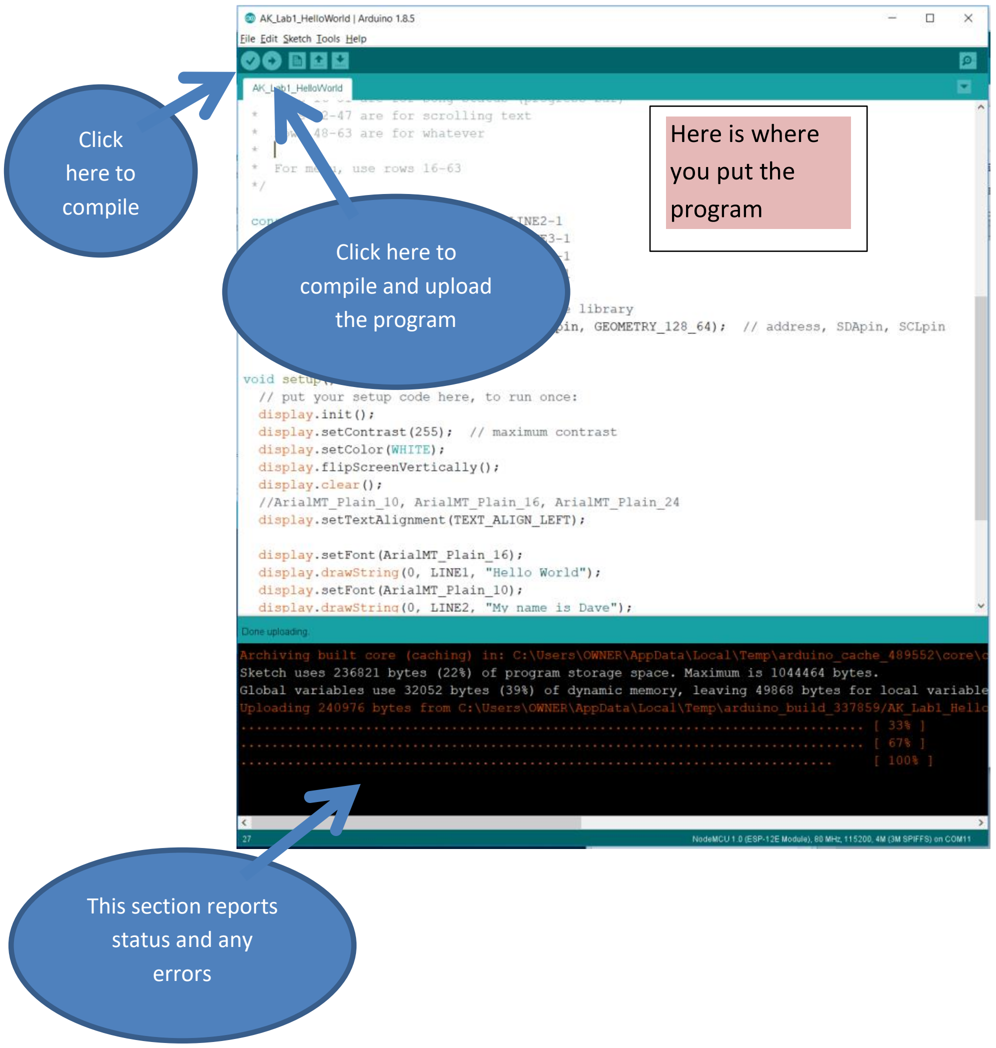
Table of Contents

Introduction

Congratulations on your purchase of a WiFi enabled Arduino with display. The Arduino world is very large with millions of hobbyists using them to control lights, switches, radios, and more.

To tell your device what to do, you write what is called a program. A program is a list of instructions that tells the computer what to do. The instructions must follow a specific set of rules called syntax. The syntax depends on what language the code is written in. For our device, we will use C or C++.

To develop your program, you typically use what is called an IDE or Integrated Development environment. There is an IDE designed specifically for writing Arduino code. This is what the IDE looks like as of this writing.


AK_Lab1_HelloWorld | Arduino 1.8.5
File Edit Sketch Tools Help
AK_Lab1_HelloWorld
Click here to compile
Click here to compile and upload the program
Here is where you put the program
This section reports status and any errors

```
 *    2-47 are for scrolling text
 *    w 48-63 are for whatever
 *
 *  For me  u, use rows 16-63
 */

con                              INE2-1
                                  E3-1
                                    -1

                                   e library
                                 pin, GEOMETRY_128_64);  // address, SDApin, SCLpin


void setup
  // put your setup code here, to run once:
  display.init();
  display.setContrast(255);  // maximum contrast
  display.setColor(WHITE);
  display.flipScreenVertically();
  display.clear();
  //ArialMT_Plain_10, ArialMT_Plain_16, ArialMT_Plain_24
  display.setTextAlignment(TEXT_ALIGN_LEFT);

  display.setFont(ArialMT_Plain_16);
  display.drawString(0, LINE1, "Hello World");
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, LINE2, "My name is Dave");
```

```
Done uploading.
Archiving built core (caching) in: C:\Users\OWNER\AppData\Local\Temp\arduino_cache_489552\core\c
Sketch uses 236821 bytes (22%) of program storage space. Maximum is 1044464 bytes.
Global variables use 32052 bytes (39%) of dynamic memory, leaving 49868 bytes for local variable
Uploading 240976 bytes from C:\Users\OWNER\AppData\Local\Temp\arduino_build_337859/AK_Lab1_Hello
................................................................................ [ 33% ]
................................................................................ [ 67% ]
............................................................................     [ 100% ]
```

```
27                    NodeMCU 1.0 (ESP-12E Module), 80 MHz, 115200, 4M (3M SPIFFS) on COM11
```

The Arduino kit comes with an ESP8266 Wifi enabled microcontroller with a built in display.

Getting Started

The first step, if not already done, is to download and install the IDE. You can do this at the Arduino website, https://www.arduino.cc/en/Main/Software. Alternatively, you can use their WEB editor found at the same page.

Next, download the ESP8266 package. Instructions are found at

https://arduino-esp8266.readthedocs.io/en/latest/installing.html

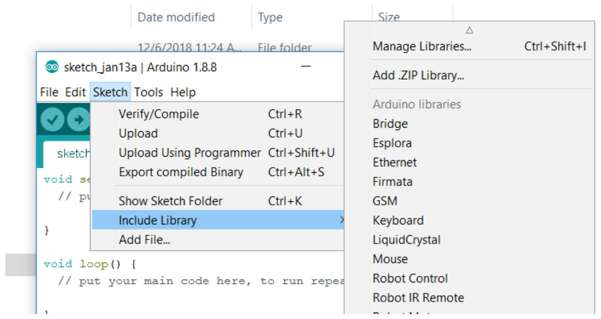
More information can be found on the GitHub web site for ESP8266,

https://github.com/esp8266/Arduino

Finally, install the libraries and examples for this kit. To do this go to

https://github.com/Spydee/ArduinoKit

Download Arduinokit.zip and AK_Starter.zip. To use, you first install the library. This is done in the Sketch menu in the IDE. Select Sketch -> Include Library -> Add .ZIP Library. Navigate to the ArduinoKit.zip file and select Open.



Then, unzip the AK_Starter.zip file in the Arduino folder (for normal install this is under Documents – Arduino. For portable install it is D:\arduino-1.8.8\portable\sketchbook where D is the portable USB drive). You are ready to start your first program.

Your First Program – "Hello World"

Typically, the first program a developer writes is a "Hello World" program. This program simply writes a short message to the screen or display to make sure it is working. We will follow this tradition. The example code has lots of comments in it to help understand the purpose of each section of the code. We will only be this verbose on the first program.

To load, open the IDE and select File->Open. If you installed the files in the correct location, you should see a list of folders including one called AK_Lab1_HelloWorld. Open this folder. You now see a file called AK_Lab1_HelloWorld.ino. This is the default extension for Arduino files. It is really a C/C++ program but the .ino tells Arduino it is the top file for the program.

The first part of the program tells the Arduino what prebuilt code (called libraries) to use.

```
/************************************************************************
* Global includes and variables
* include the library files with the functions we will use
************************************************************************/
#include "AK_SH1106.h"

/*
*  Just like your home, we need an address to send data to the display
*  SH1106 is the protocol used by this display
*  The SDA pin is the arduino pin the code uses for data
*  The SCL pin is the clock pin used to tell the device when to read the data
*/
const uint8_t SH1106_addr = 0x3c; // I2C address of display
int SDApin = D1; //D2;
int SCLpin = D2; //D1;
```

The text is green is a comment. Comments start with /* and end with */. Or you can use a single line comment by typing // followed by the comment. Comments are meant for humans and are ignored by the program.

```
#include "AK_SH1106.h"
```

This line tells the Arduino compiler to include a file called AK_SH1106.h. This file contains definitions needed to talk to the display. Typically we use .h extension for definition files and .c or .cpp (or .ino for arduino) extension for the actual program.

```
const uint8_t SH1106_addr = 0x3c; // I2C address of display
int SDApin = D1; //D2;
int SCLpin = D2; //D1;
```

These lines define constants or variable to be used by the program. Let's break it down. We will first look at the address definition. It has 5 fields. Each device, the display, the radio, and the touch pad use I2C protocol which requires an address. For the display, the address is 0x3C.

Field	text	Description
1	`const`	Tells the compiler this cannot be changed. Unlike variables, this is a constant
2	uint8_t	This is an 8-bit integer. “u” means unsigned as in always positive. “int” stands for integer, “8” means it is an 8-bit number, and “_t” is a standard way of saying it is a type of variable
3	SH1106_addr	This is the variable or constant name. From now on the compiler will replace this with the value assigned to it.
4	0x3c	This is the value of the constant. 0x tells the compiler the number is in HEXIDECIMAL format. To convert to decimal we multiply the 3 by 16 (HEX is base 16) then add “C” which is 12 (HEX uses 0-9 then ABCDEF for numbers 10 to 15). So the decimal equivalent of 3C is 60.
5	`; // I2C address of display`	The semicolon tells the compiler this is the end of the code for this line. The // tells the compile to ignore everything that follows it because it is a comment

The next line does not start with const so it is a variable and can be changed by the program anytime. The keyword int tells the compiler to use its generic integer format and that negative numbers can be used. SDApin = D1 tells it to replace SDApin with D1 wherever it sees it.

```
int SDApin = D1; //D2;
```

The next section of the code is similar. The comments tell us the SH1106 display is 128 pixels wide by 64 pixels high. To make it easier to write code later, we define text lines as LINE1 = 0 (top of the display), LINE2 = 16 (starts at the 16th pixel from the top) etc. Because we don’t want to change this, we assign them as constants. Finally, the keyword int16_t tells the compiler it is a 16-bit signed integer.

```
/*
*  Display is 128 x 64
*  Rows 0-15 (top) are for status of battery, antenna, etc
*  Rows 16-31 are for song status (progress bar)
*  Rows 32-47 are for scrolling text
*  Rows 48-63 are for whatever
*
*  For menu, use rows 16-63
*/

const int16_t LINE1 = 0; // 0 to LINE2-1
const int16_t LINE2 = 16; // 0 to LINE3-1
const int16_t LINE3 = 32; // 0 to LINE4-1
const int16_t LINE4 = 48; // 0 to LINE5-1

// Initialize the OLED display using Wire library
SH1106 display(SH1106_addr, SDApin, SCLpin, GEOMETRY_128_64);  // address, SDApin,
SCLpin
```

The last line of this sections tells the compiler to create an instance of the object SH1106 and call it display. We pass 4 variables to it, the address, the data pin number, the clock pin muber, and another number that tells it the display resolution.

Next we see the Arduino specific function called, setup. This is run one time when the Arduino wakes up. Inside are several functions for the display.

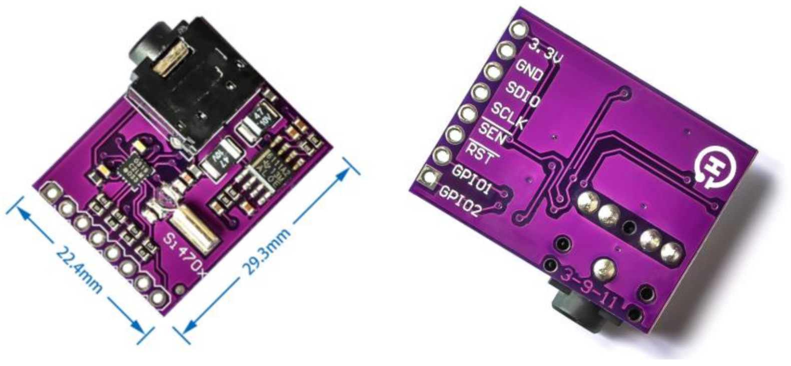
```
void setup() {
       // put your setup code here, to run once:
       display.init();
       display.setContrast(255);  // maximum contrast
       display.setColor(WHITE);
       display.flipScreenVertically();
       display.clear();
       //ArialMT_Plain_10, ArialMT_Plain_16, ArialMT_Plain_24
       display.setTextAlignment(TEXT_ALIGN_LEFT);

       display.setFont(ArialMT_Plain_16);
       display.drawString(0, LINE1, "Hello World");
       display.setFont(ArialMT_Plain_10);
       display.drawString(0, LINE2, "Lilly is my friend");

       display.display();


}
```

Finally we have the loop section of the code. This is run over and over again. Put the code here that is to be repeated over and over. For this program, we leave this section blank.

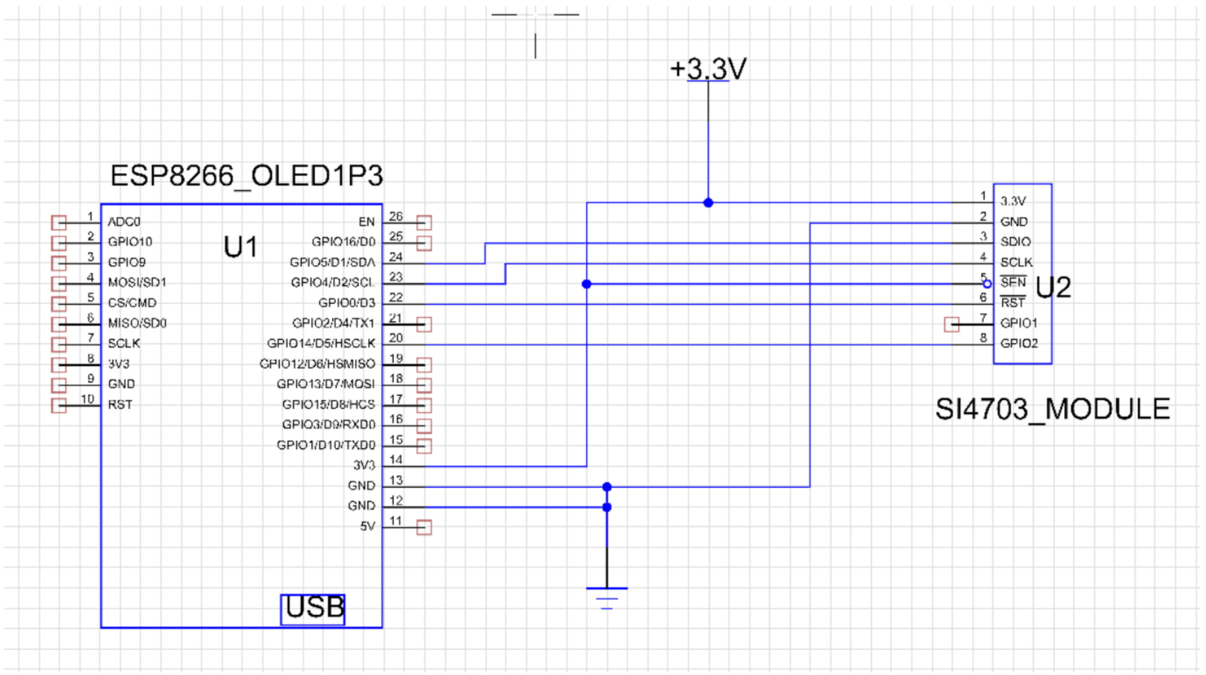
```
void loop() {
       // put your main code here, to run repeatedly:

}
```

SI4703 Radio

This kit includes an FM radio that can be programmed using the Arduino. The first step is to connect the radio breakout board to the Arduino.



Here is the schematic.

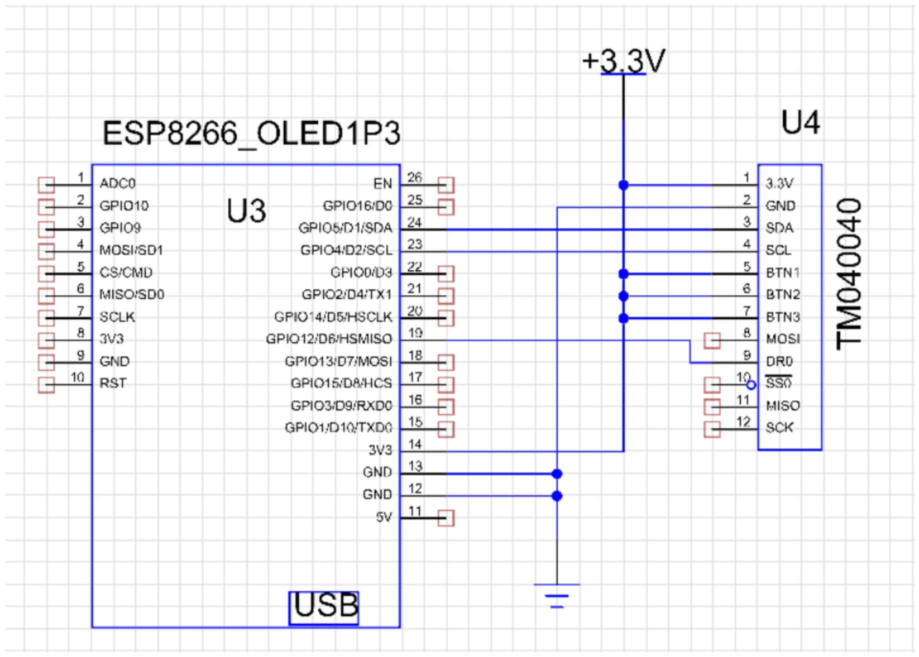


The connection description is provided in the table below. The SI4703 radio talks to the Arduino using the I2C protocol. The SI4703 requires the SEN pin to be high when RST goes from low to high. That is why it is connected to 3.3V.

Si4703 pin	Arduino pin	Name	Description
1	3.3V	3.3V	This is the positive power supply pin
2	GND	GND	This is the negative power supply pin or ground
3	24 (D1)	SDA or SDIO	This is the data pin which contains the commands from the Arduino and the data sent back from the SI4703.
4	23 (D2)	SCL or SCLK	This is the clock pin that tells the SI4703 when to read the data on SDA
5		SEN	This pin must be connected to 3.3V in order for the SI4703 to talk to the Arduino using I2C protocol
6	22 (D3)	RST	This pin is used to reset the SI4703 to a known state. It is low to reset and high for normal operation.
7		NC	Not used
8	20 (D5)	INT	This is an interrupt pin. It is used by the SI4703 to tell the Arduino there is data to be read or that a process is complete

Cirque TM040040 Touch Pad

The Cirque TO040040 touch pad is the same touch pad as used in many game controllers including virtual reality games. It is a circular touch pad and can talk to the Arduino using I2C or SPI protocol. To connect it to the Arduino, use the schematic below.



Si4703 pin	Arduino pin	Name	Description
1	3.3V	3.3V	This is the positive power supply pin
2	GND	GND	This is the negative power supply pin or ground
3	24 (D1)	SDA or SDIO	This is the data pin which contains the commands from the Arduino and the data sent back from the TM040040.
4	23 (D2)	SCL or SCLK	This is the clock pin that tells the TM040040 when to read the data on SDA
5	BTN1	NC	This pin is used for an optional button.
6	BTN2	NC	This pin is used for an optional button.
7	BTN3	NC	This pin is used for an optional button.
8	MOSI	NC	Not used. This pin is used for SPI protocol only
9	DR0	19 (D6)	DR) stands for Data Ready. It is an interrupt pin used by the TM040040 to tell the Arduino there is data to be read or that a process is complete
10	SS0	NC	Not used. This pin is used for SPI protocol only
11	MISO	NC	Not used. This pin is used for SPI protocol only
12	SCK	NC	Not used. This pin is used for SPI protocol only