SEICHE 2023
Intermediate Arduino Programming for IoT

Instructor: Paul Frommeyer

www.paulfrommeyer.com

Corporate Sponsor: DXC Technology

REMINDER

YOU MUST HAVE A *WORKING* LED MATRIX DISPLAY TO TAKE THIS CLASS!

If displays are lost or damaged, replacements are $70 with 1-week lead time for replacement

Lesson Plan Overview

Lesson 1: 7Feb23 – Review, Addressing, and Pointers

- Overview of lesson plan
- Class Exercise: Validation of sketch uploading
- Class Exercise: Validation of binary create and upload
- More on the Preprocessor
- Memory Organization and Variables
- Review of addressing and pointers
- Using pointers to variables
- Declaring pointers in function calls
- Call-by-value Function Calls
- Call-by-reference Function Calls

Lesson 2: 14Feb23 – More Gory Details

- Class Exercise: Call-by-reference variables
- Introduction to Complex Data Structures
- Using complex data structures
- Storage declarations and Arduino

Lesson 3: 21Feb23 – Fonts Redux

- MD_Parola library font usage review
- The MD_Parola font format
- Designing your own fonts
- Declaring your own fonts
- Using your own fonts with MD_Parola

Lesson 4: 28Feb23 – Intro to Visual Studio Code

- Introduction to VSC
- https://code.visualstudio.com/docs/introvideos/overview
- VSCode Installation
- VSCode Integration with Arduino and ESP8266

Lesson 5: 7Mar23 – Filesystems

- Introduction to mass storage filesystems
- The SD FAT, FAT16, and FAT32 filesystems
- The exFAT filesystem
- Introduction to SPIFFS
- Class Exercise: Using SPIFFS

Lesson 6: 14Mar23 – WiFi

- Review of WiFi technology (but no gory details)
- Class Exercise: Review of WiFi *access* with WiFi Manager
- Creating access points with ESP8266
- Class Exercise: Creating an access point

Lesson 7: 21Mar23 – Web Technology

- Introduction to HTML
- Web server and client architecture
- Creating web servers with ESP8266
- Class Exercise: A basic web server for time and temperature

Lesson 8: 28Mar23 – Arduino Web Services

- Storing web pages using SPIFFS
- HTML forms
- Form processing with Arduino
- Class Exercise: Displaying text entered on a web page
- Obtaining weather information from the Internet
- Class Exercise: Displaying weather information

Lesson 9: 4Apr23 – IoT Messaging and MQTT

- Introduction to IoT network messaging and MQTT
- Subscribing to an MQTT service
- Publishing to an MQTT service
- Class Exercise: Using MQTT and WiFi

Lesson 10:11Apr23 – Version control and Git

- The need for document version control
- The Git version control system
- Installing Git
- Using Git
- Introduction to Github
- Using Github
- Class Exercise: Signing up for a Github account

Lesson 11:18Apr23 – Using VSC

- VSC plugin review
- Using VSC with Arduino
- Using VSC with Git

Flash Drive Contents Reminder

- When I update parts of your flash drives, you can either *recopy the entire section*, or *just copy the updated part*.
- However, **you will need to explicitly copy any new files so that they are locally accessible on your laptops**
- Copying just *lesson* files will– wait for it!– *copy only the lesson files.*

Make certain you have copied everything in RED!

You should already have previously copied everything else!

FLASH DRIVE

- Archive–BasicProgrammingWith Arduino-Fall2022
- Archive-IntroductionToArduino-Spring2022
- IntermediateProgramming-Software
- IntermediateProgramming-Lessons
- IntermediateProgramming-Documentation

IntermediateProgramming-Software

- Kubuntu
  - *code_1.75.0-1675266613_amd64.deb*
  - arduino-ide_2.0.4_Linux_64bit
- Windows
  - *VSCodeSetup-x64-1.75.0.exe*
  - *VSCodeUserSetup-x64-1.75.0*
  - arduino-ide_2.0.4_Windows_64bit

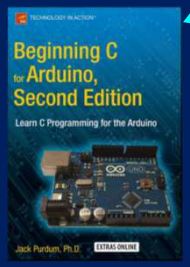
IntermediateProgramming-Lessons

- IntermediateProgramming-Lesson1
- IntermediateProgramming-Lesson2
- IntermediateProgramming-Lesson3
- IntermediateProgramming-Lesson4

IntermediateProgramming-Documentation

- ArduinoReference
  - **Beginning C for Arduino, 2nd Edition**
- BoardsGuides
- D1 Mini Reference
- ElectronicsReference
- IoT Reference
- LED Matrix Display Architecture
- HackSpace Magazine

Can't do reading homework without the files!



Lesson 4 – Visual Studio Code

Part A

- Arduino IDE Limitations
- Introduction to VS Code
- VS Code introductory videos

Part B

- Installing VS Code
  - **Kubuntu**
  - **Windows**
- Linking VS Code with the Arduino IDE
- Assuring we have ESP8266 available in VS Code
- Uploading a sketch using VS Code

Arduino IDE - Limitations

https://learn.sparkfun.com/tutorials/efficient-arduino-programming-with-arduino-cli-and-visual-studio-code/all

The Arduino IDE lacks a number of "professional" code-assistance features, like:

- **Code navigation -- Whether it's find-by-reference (instantly navigating to the definition of the function you're using), search-by-symbol (quick navigation to function or symbol definitions within a file), or a quick link to a compilation error, code navigation is critical to managing large code bases.**
- **Auto-Complete -- This feature can, of course, help complete long constant names, but it can also provide insight into the parameters that a function may expect.**
- **Version control integration -- Whether you're using git or SVN, many modern IDE's provide source-control integration that can show, line-by-line, the changes you've made since your last commit.**
- **Refactoring -- Need to overhaul a function's naming scheme? Or convert a common block of code into a function that can be more widely-used throughout your application? Sounds like a refactoring job! A modern IDE can help with that.**
- **Integrated Terminal -- Whether you use bash or the Windows CMD, an integrated terminal can save you loads of time. This tool allows you to run "make", "grep", or any of your favorite terminal commands without ever swapping windows.** *[Especially true on Linux – PLF]*

Once you take the time to learn these tools they make programming in C/C++ (or any language, really) so much more efficient. They help produce better code faster.

Visual Studio Code

https://code.visualstudio.com/docs

- Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, C#, Java, Python, PHP, Go, .NET).

- Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

VSCode – Introductory Videos

https://code.visualstudio.com/docs/introvideos

- **Overview**
  https://code.visualstudio.com/docs/introvideos/overview
- **Basics**
  https://code.visualstudio.com/docs/introvideos/basics
  https://youtu.be/B-s71n0dHUk
- **Code Editing**
  https://code.visualstudio.com/docs/introvideos/codeediting
- **Extensions**
  https://code.visualstudio.com/docs/introvideos/extend

VSCode – Arduino Setup

- Bye Bye Arduino IDE – Welcome VSCode – *"RoboCircuits"* https://www.youtube.com/watch?v=WVZxK2MEbE4
- Setup and program NodeMCU esp8266 using Arduino VSCode extension – *"Dr. Sensor"* https://www.youtube.com/watch?v=UOsv7dwXuaA

VSCode – Arduino Setup

[Note: These are reference pages, not videos]

- Using VS Code for Arduino
  https://maker.pro/arduino/tutorial/how-to-use-visual-studio-code-for-arduino

- **Arduino CLI and VSCode**
  https://learn.sparkfun.com/tutorials/efficient-arduino-programming-with-arduino-cli-and-visual-studio-code/all
- **Programming ESP and other Arduino compatible chips using VS Code on MacOS**
  https://blog.anoff.io/2020-12-programming-arduino-esp-with-vs-code/

Formal End of Lesson 4

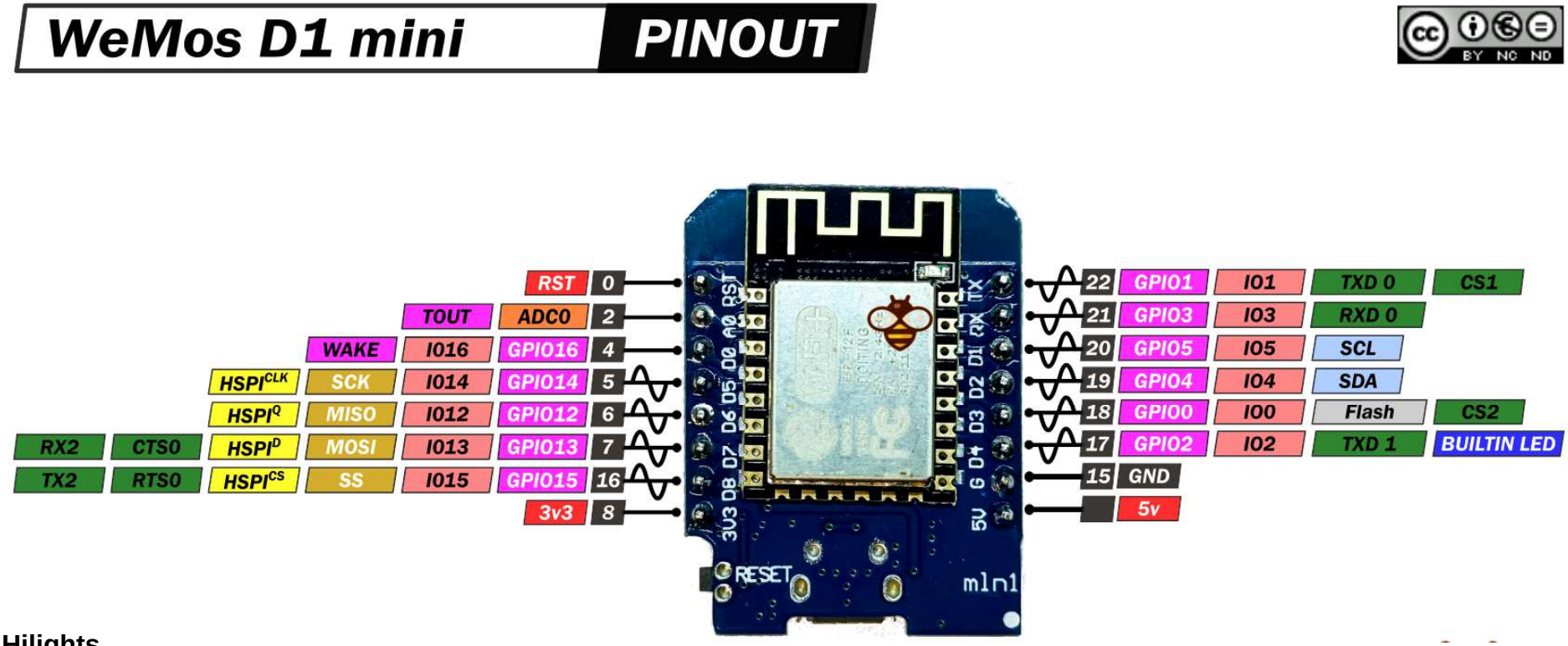
HOMEWORK

- **Check the Internet for more articles or videos on Visual Studio Code and Arduino**
- **For the adventurous, you can search for articles on using PlatformIO with VSCode.**

In next week's exciting episode

- Video – Installing Visual Studio Code
- Video – Using Visual Studio Code with Arduino
- Class Exercise – Installation of Visual Studio Code
- Instructor's time permitting – More on Parola font usage

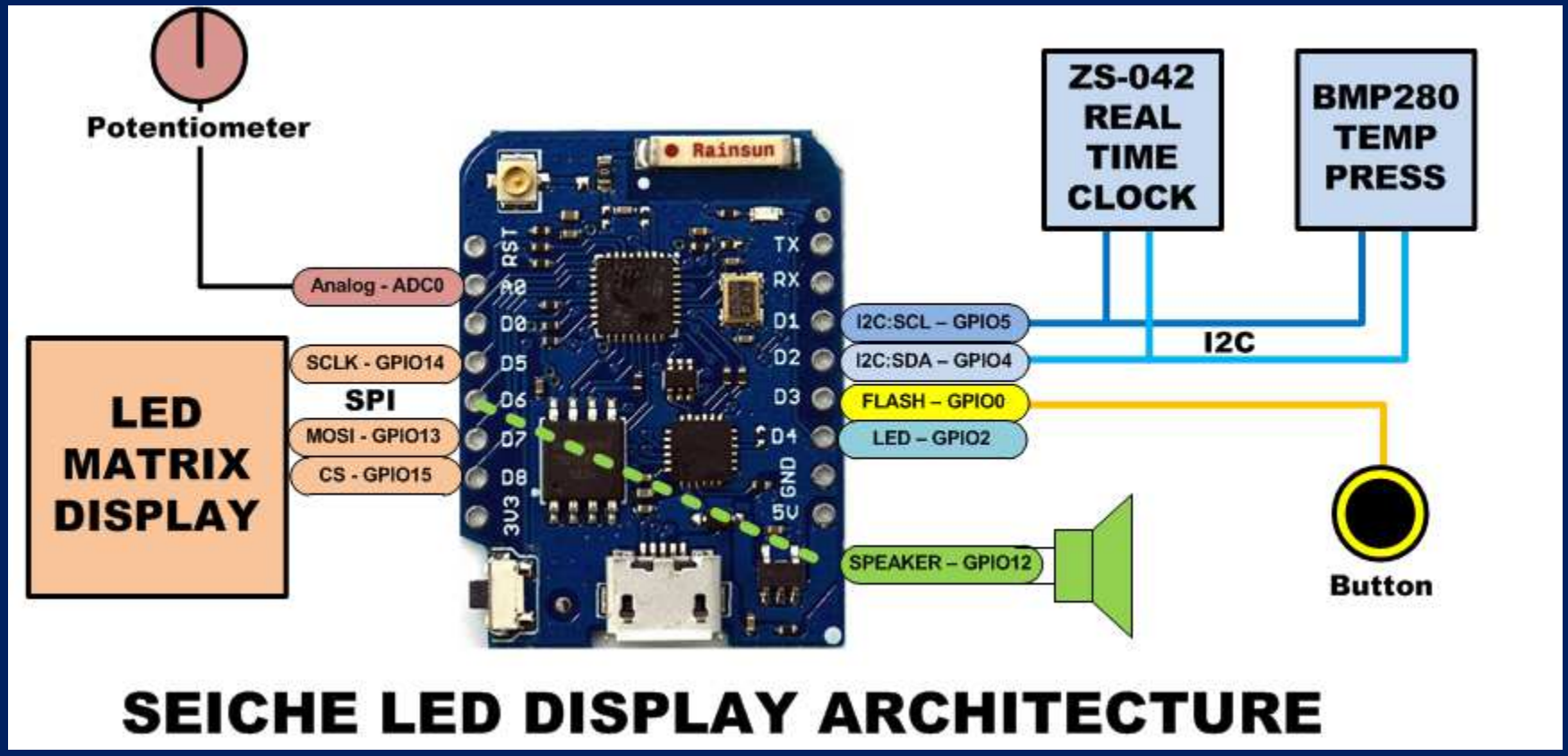
Our Microcontroller: The WeMos D1 Mini



Hilights

- **11 digital IO: all are interrupt and pwm capable (except D0/GPIO16)**
- **1 analog input (3.2V max input): A0/ADC0**
- **Micro USB or Type-C USB Port (clones usually have micro USB)**
- **Two SPI interfaces (one is used for on-board flash memory), one I2C interface, two serial ports**
- **Built-in WiFi (client or standalone access point modes) and Bluetooth**
- **Compatible with MicroPython, Arduino, NodeMCU**
- **Uses the CH340 USB-to-serial driver (installation usually needed on Windows)**
- **Extremely low cost (approx $3.00 US on Amazon; one of the two least expensive components in your kits)**

SEICHE LED Display Architecture



- Red MAX7219 8x32 LED matrix display (SPI)
- ZS-042 real-time clock module (I2C)
- BMP280 Temperature and Pressure Sensor (I2C)
- Piezoelectric speaker (PWM)
- 10KΩ Potentiometer (Analog-to-Digital Converter)
- Button (Pullup and interrupt)

`sprintf()` Function Syntax

- `sprintf()`'s formal syntax is:
  `sprintf(`char ***`buffer`**, const char ***`format`**, *variable list*`)`
- **`buffer`** is an array of type char (the parameter when passed can also be a pointer [address] for such an array) which will contain the formatted output of the function
- **`format`** is an unmodifiable (constant) array of char containing the format descriptors for all variables subsequently passed to the function, which is to say, it's the *format string*.
- The variable list are just the comma separated variables that are to be formatted
- All variables passed in a single call to `sprintf()` are combined into a single output string
- `sprintf()` automagically puts a terminating NULL (\0) at the end of the ASCII output

`sprintf()` Format Strings

- `sprintf()` format strings contain conversion specifiers that specify how each individual variable is to be converted
- `sprintf()` conversion specifiers have the following general syntax (all begin with a percent-sign):

  %[flags][minimum field width][.][precision][length][conversion character]

  - % - special token that indicates the start of a conversion specifier
  - Flags – these modify the behavior of the specification
  - Minumum field width – as it says on the tin; this the minimum number of characters to be converted
  - . – The period is a separator between field width and precision
  - Precision – means one of the following depending on the variable type and conversion specifier
    - The maximum number of characters to be generated from a string
    - The number of digits after the decimal point for type float conversions (e, E, or f)
    - The zero-filled minimum number of digits for an integer
  - Conversion character – A single character which determines the output type of the conversion specifier; a single character that specifies the type of output format for the corresponding data or variable

`sprintf()` Conversion Specifiers

Below are the general conversion specifiers and what they do.

Specifier	What it does
d, i	int - integer; signed decimal notation
o	int – unsigned octal (no leading zero)
x, X	int – unsigned hexadecimal, no leading 0x
u	int – unsigned decimal
c	int – single character, after conversion to unsigned char
s	char * - characters from string are printed until \0 (NULL) or *precision* is reached
f	double – decimal notation of form [-]mmm.ddd where number of decimals is specified by precision; precision of zero (0) suppresses the decimals altogether
e, E	double - exp notation; default precision of 6, 0 suppresses
g, G	double – Use %f for <10^4 or %e for >10^4
p	void * - print output as a pointer, platform dependent
n	Number of characters generated so far; goes into output
%	No conversion, put a % percent sign in the output

`sprintf()` Conversion Specifiers

Below are the flags and what they do.

Flag	What it does
-	Left justification
+	Always print number with a sign
spc (space)	Prefix a space if first character is not a sign
0 (zero)	Zero fill left for numeric conversions
#	Alternate output form depending on conversion character o – first digit will be zero x or X – 0x or 0X (respectively) prefixed to non-zero results e, E, f, g and G – Output will always have a decimal point g and G – trailing zeroes will never be removed