SEICHE 2022
Basic Arduino Programming

Instructor: Paul Frommeyer

www.paulfrommeyer.com

Corporate Sponsor: DXC Technology

Lesson Plan Overview

Lesson 1 – Intro and Setup
[may require 2 classes]

- Introduction to class format
- Overview of lesson plan
- Presentation format (monitor, camera, screen, whiteboard)
- Review of microcontrollers and types of boards
- SEICHE LED display architecture
  - ESP8266 pinout
  - High level architecture

Lesson 2 – Laptop operation review – Windows and Linux

- Inventory of USB drives
- Installation of Arduino IDE software
- Installation of CH340/ESP8266 serial port drivers (Windows only)
- Control panel/settings location
- Home directories and folder hierarchy
- Arduino file locations
- Search functions
- (Windows) Device Manager
- (Linux) Konsole
- Copying flash drive contents [critical]
- Open questions and issues
- **IDE essentials**
- Starting the Arduino IDE
- Basic Arduino sketch (program) structure
- Loading example sketches
- Loading and configuring new boards
- Connecting boards
- Identifying the microcontroller serial port
  - Linux
  - Windows

Lesson 3 – Libraries, Sketch structure, Serial Monitor, Variables, Binary Number System Pt1

- Libraries
- Sketch structure (A note on brace formatting)
- The serial port monitor
- Printing to the serial port monitor
- Variables and the assignment operator
- Binary number system Pt. 1.

Lesson 4 – Expressions, Conditionals, Blocks and Functions

- Arithmetic Expressions and Operators
- Incrementing and Decrementing Variables
- Truth Values in C++
- The If-Then Statement
- Code Blocks
- Functions

Lesson 5 – Binary Images, Arrays, Characters, Strings, Loops

- Loading Binary Images
- Arrays
- Characters and Character Codes
- Strings
- Conditional Loops Part 1

Lesson 6 – Loops (cont.), LED Matrix Displays, Nested Loops Advanced Functions, Binary Numbers Part 1

- For-Next Loops
- SPI Peripherals
- Using a MAX7219 LED Matrix Display
- Lighting and clearing individual pixels
- Advanced Functions
- Nested Loops

Lesson 7 – The Binary Number System (may take 2 lessons)

- Numerals vs numbers
- Review: the base 10 system and digit place values
- New: the base 2 system and digit place values
- Bits and bytes and nybbles
- Binary addition and subtraction
- Formatting printed output
- Shifting and exponents

Lesson 8 – Producing Sound

- Review of sound wave theory
- Analog vs Pulse Width Modulation
- Producing sound tones with an Arduino microcontroller

Lesson 9 – Reading pins

- Reading buttons
- Millis() and debouncing buttons
- Reading analog values from a potentiometer

Lesson 10 – The I2C Bus and Peripherals

- I2C Bus Operation
- Initializing the I2C bus
- Accessing an I2C temperature sensor
- Displaying text on the LED matrix
- Default fonts

Lesson 6 – Loading Binary Images; Arrays, Characters, Strings and Loops

Part A – Homework Exercises

- Write a sketch to increment a number until reaches 200, then decrement it until it reaches 0, then start incrementing it again. You can use either the ++/-- operators or selfassignment for variable updating. You'll execute the sketch next week.
- Take your homework sketch from last week and modify it to print the contents of an integer array of 99 elements, starting from the first element to the last, then from the last to the first. Use while instead of if-then. You'll execute the sketch next week.

Part B

- The For Loop
- SPI Peripherals
- Using a MAX7219 LED display
- Lighting and clearing individual pixels
- Advanced Functions
- Nested For- Loops

Homework Review 1

- Write a sketch to increment a number until reaches 200, then decrement it until it reaches 0, then start incrementing it again. You can use either the ++/-- operators or selfassignment for variable updating. You'll execute the sketch next week.
- Who was able to come up with a solution to this?
- Remember that you can use the IDE to *validate* your code, even if you do not have a microcontroller connected to *run* it on!
- Discussion of different approaches to solving the problem

Homework Review 2

- Take your homework sketch from last week and modify it to print the contents of an integer array of 99 elements, starting from the first element to the last, then from the last to the first. Use while instead of if-then. You'll execute the sketch next week.
- Who was able to come up with a solution to this?
- Notice that you were *not* required to *initialize* the array (fill it with data)
- Discussion of different approaches to solving the problem
- What did you see when printing out the values stored in the array?

The `for` Loop

- The `for` loop is a control structure that is more efficient than a `while` loop and, in C++, incredibly flexible
- It is intended to combine the conditional iteration of a `while` loop with iterative variable modification
- The formal syntax of the `for` loop is:

```
    for (start statement; conditional test; repeating statement)
            statement;
```

- `For-next` is almost always used with a code block

```
for (i=0; i < 256; i++)
{
            Serial.print("ASCII: "); Serial.println(i);
            Serial.print ("DEC: "); Serial.println(i, DEC);
}
```

- Historical note: In BASIC and other languages, the `for` loop is implemented more simply as a FOR-NEXT combination using only a single variable, and typically only single increment, thus:

```
FOR N = 1 TO 100
NEXT N
```

The For Loop – Let's Give It A Go

Load Sketch 6A, then modify it to use a `for-next` loop instead of `while`. Remember that ; semicolon statement terminators are *never* optional, and C is *case-sensitive*!

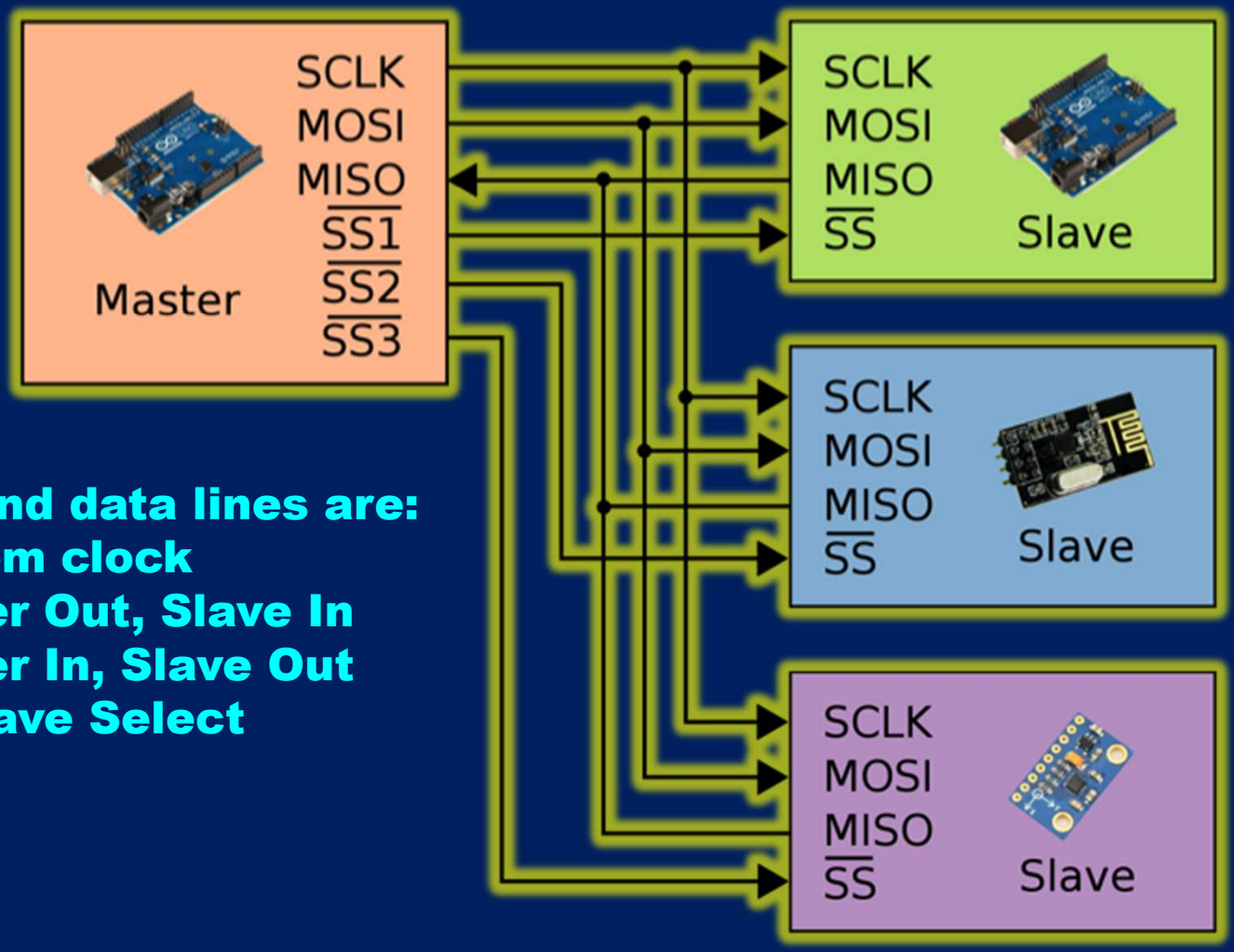
```
char c;
void setup()
{
  // Setup port for serial monitor
  Serial.begin(9600);
}

// Still printing the ASCII character set!
void loop()
{
  c = 0;
  while(c < 256)
  {
     Serial.print("ASCII: "); Serial.println(c);

     Serial.print ("DEC: "); Serial.println(c, DEC);
     c++;

     delay(500);
  }
}
```

Did it work? If not, what went wrong?

The SPI Protocol

- Serial data transmission protocols transmit the bits of a number (which may also be a character) *sequentially*
- SPI, the *Serial Peripheral Interface*, is a *very* fast serial protocol used for communication between microcontrollers and peripherals (devices that are attached to the uC).
- SPI uses a so-called master-slave architecture (not a social statement!) where the microcontroller (master) selects the peripheral (slave) it wants to talk to and tells it what to do. Peripherals *cannot* initiate communication or issue to commands to the microcontroller.
- *More than one* SPI peripheral can be attached to a given set of SPI control and data wires, making SPI a *bus* based architecture

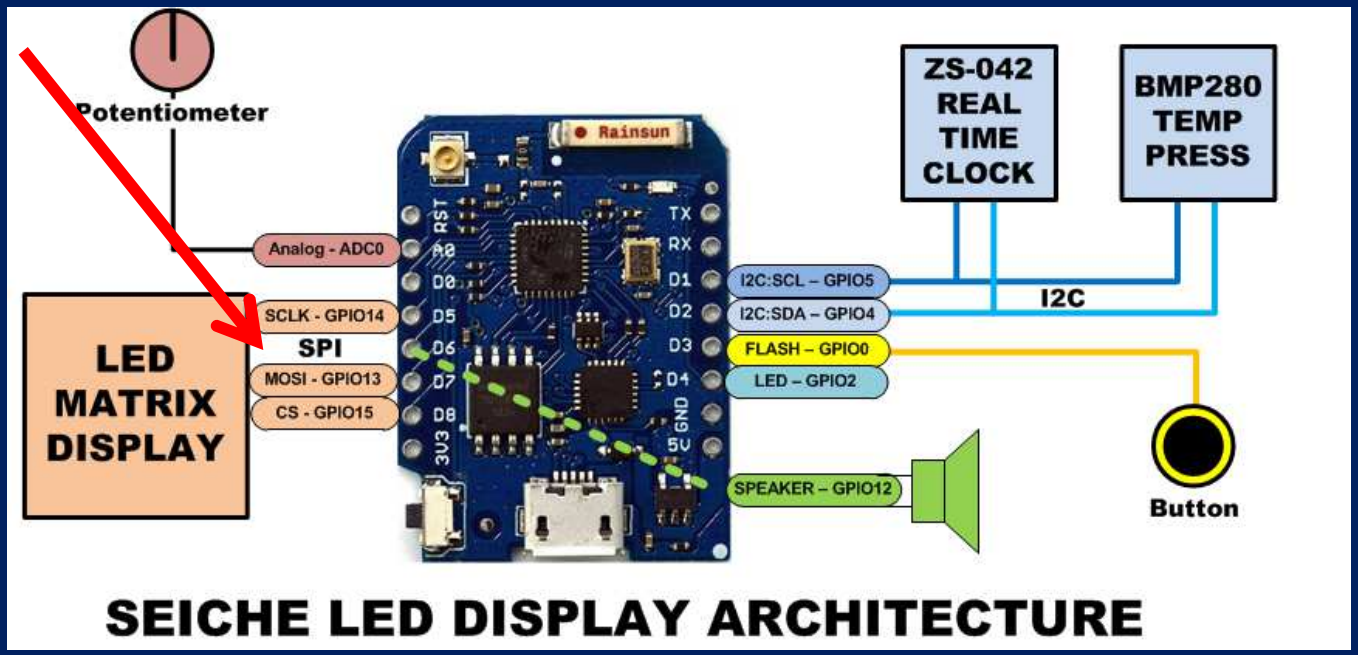
The SPI Protocol



SPI control and data lines are:
SCLK – System clock
MOSI – Master Out, Slave In
MISO – Master In, Slave Out
SS or CS – Slave Select

The SPI Protocol

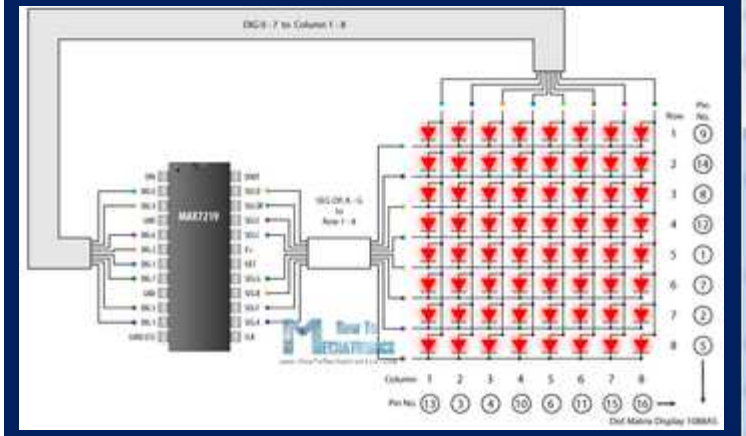
You must know the pins an SPI device is using in order to use it!



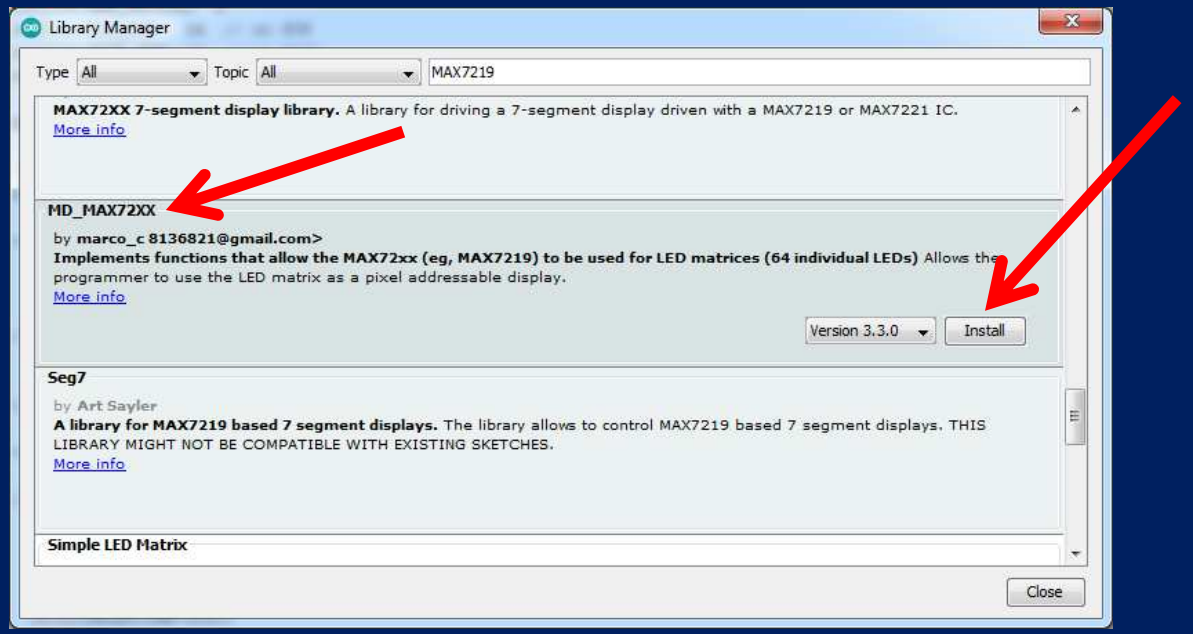
- In your LED displays, SPI is the protocol used to connect the MAX7219 display to the ESP8266 microcontroller
- Note the pin numbers that are used for this! *You will need them in your sketches!*

The MAX7219 LED Display

- Your displays use the MAX7219 chip to drive the LED display. The MAX7219 abstracts the 8x8 LED array into a coordinate addressable X-Y matrix.
- Using a MAX7219 requires a special library, which we must load using the Library Manager
- **Go ahead and install** the MD_MAX72XX library



MAX7219 LED Array



Working with LED Matrices

- Each LED in a matrix, or any X-Y display, is called a *pixel* from “picture element”
- The MD_MAX72XX library provides ways to individually control pixels
- LED matrices are first created as software objects:
  ```
  MD_MAX72XX ledmx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, MAX_DEVICES);
  ```
  Note that this is where you need those GPIO pin numbers!
- Next we have to initialize the LED array:
  ```
  ledmx.begin();
  ```
  And tell it to continuously update the display
  ```
  ledmx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
  ```
- Now we are ready to actually display information on the LED matrix!

Clearing and Setting Pixels

- The command to illuminate a single pixel using MD_MAX72XX is

```
ledmx.setPoint(5,5,true);
```

  whose parameters are

```
(x-coordinate, y-coordinate, true|false)
```

- To clear/turn off a single pixel, we use the same statement only with false as the state parameter:

```
ledmx.setPoint(5,5,false);
```

- Some libraries require the programmer to manually “refresh” the display, however MD_MAX72XX allows us to specify “continuous” updates (see previous slide)

MAX7219 LED Matrix
Let's Give It A Go

Load and execute Sketch6B:

```
#include <MD_MAX72xx.h>

#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES  4
#define CLK_PIN   14  // or SCK
#define DATA_PIN  13  // or MOSI
#define CS_PIN    15  // or SS

MD_MAX72XX ledmx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, MAX_DEVICES);

// We always wait a bit between updates of the display
#define  DELAYTIME  100  // in milliseconds

void setup() {
  // put your setup code here, to run once:
  ledmx.begin();
  ledmx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
//  ledmx.clear();
  Serial.begin(9600);
  Serial.println("MAX7219 TEST");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(DELAYTIME);
  ledmx.setPoint(5,5,true);
  delay(DELAYTIME+500);
  ledmx.setPoint(5,5,false);
  delay(DELAYTIME+500);
}
```

Did it work? What do you see on your display?

Function Definitions Part 2

- We already covered how to define functions to group frequently re-used code together.

```
some_function()
{
        // code goes here
}
```

- Now we will learn about how to receive a value back from a function, and how to pass information to a function
- Functions can have return values. The type of data that is returned is specified in the declaration of the function. So

  ```
  int some_function()
  ```

  has been defined so that it can return an integer value. However, it doesn't have to. If no return value is specified, it will default to zero (false).
- The way we return information from a function is with the– wait for it!– return statement:

  ```
  return(return_value);
  ```

Function Definitions (cont.)

- When functions are invoked we say they are called. So from the calling code, using a return value would look like this:

```
int bar;
bar = foo();
int foo() { return(2+2); }
```

This code will ultimately place the value of 4 in the variable bar. See how that worked?

- Return values are typically used for determining success, failure, or some error condition of a function. This is really useful, and you will see it used a lot:

```
if(Serial.begin(9600))
{ Serial.println("Serial monitor started"); }
else { abort (); } // Halt (And Catch Fire)
```

Function Definitions (cont.)

- So how do we define a function so it can accept *parameters*? Like this:

```
int foo(int bar) { bar = bar + 8; return(bar); }
```

- This code accept an integer parameter at the time the function is called, place that value in the variable bar, then add 8 to bar, then return the value of bar to the calling function.
- Thus, `baz = foo(24)` would return the number 32 and place it in the variable `baz`.
- If a variable is passed as a function parameter, *it is not modified!* Only *the value of the variable* is passed to the function at the time the function is called.

```
Int baz = 16, rvalue=0;
rvalue = foo(baz);
Serial.println(baz,DEC); Serial.println(rvalue,DEC);
```

  would print **16** and then 24. Baz is not modified.
- Also– and this is critical– all local variables in a function are reset every time it is called; this works just like the loop() function in that way.

Nested Loops

- It is possible to “nest” loops, that is, have one inside the other like those Russian matrioshka dolls.
- This is an extremely common construct, and the combination of while, if-next-else, and for loops can get quite deep.
- Here is but one example; you will see, and use, many others:

```
int i=99;
byte j=88;

void loop()
{
  for (i=0; i<16; i++)
  {
    for(j=1; j<16536; j = j*2;)
    {
       Serial.println(i*j,DEC);
    }
}
```

Formal End of Lesson 6

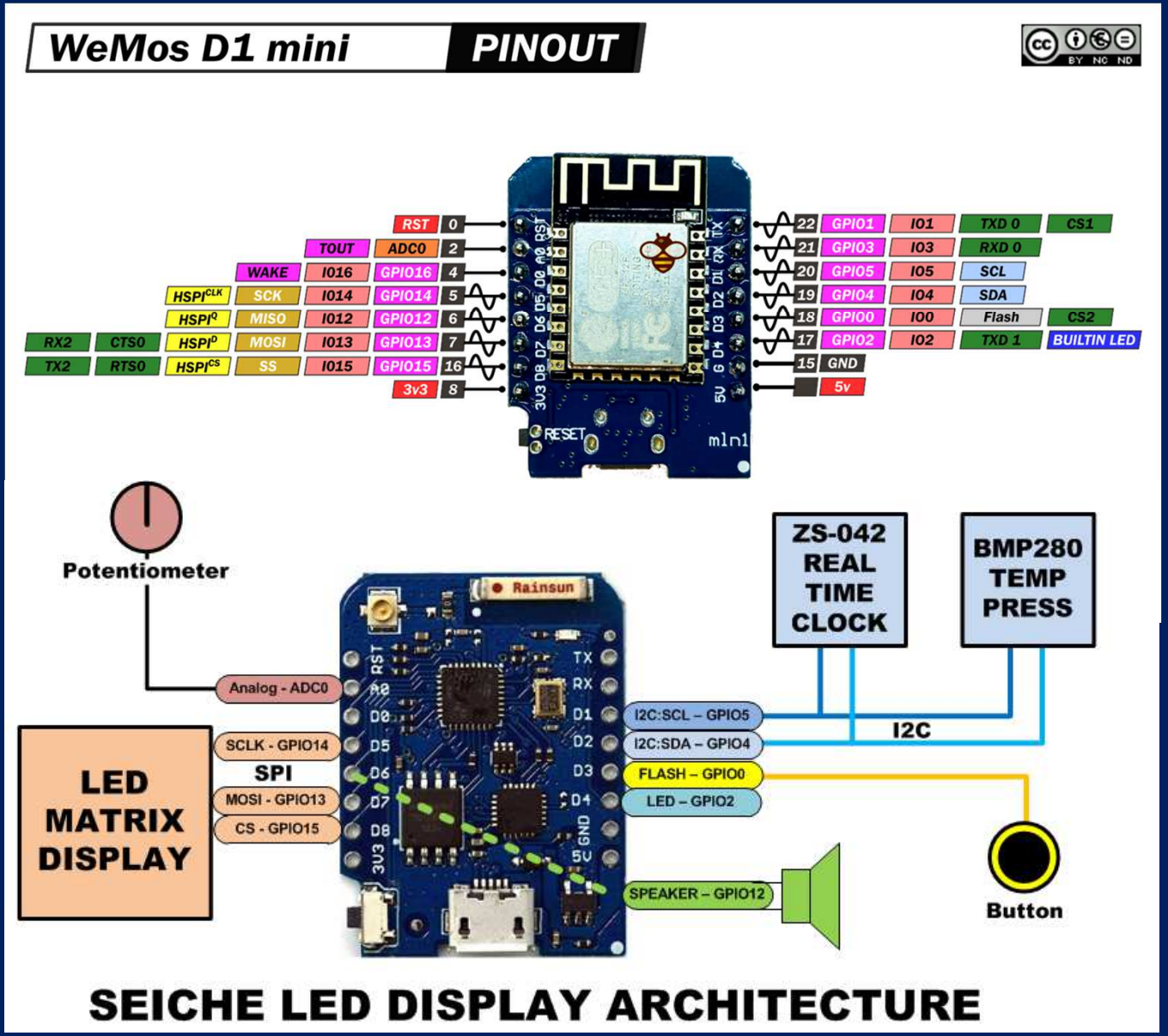
HOMEWORK!

- **Arduino Cookbook – You can begin reading Chapter 4; we will not be covering all of Chapter 4 in class**
- **Another programming assignment! Yay!**
  Modify Sketch6B to use one or more for loops to light all the pixels at the edge of the first LED square
  Remember that you can validate your sketch without a microcontroller attached!

In next week's exciting episode

- Homework review and testing
- The binary number system (should finally be ready)

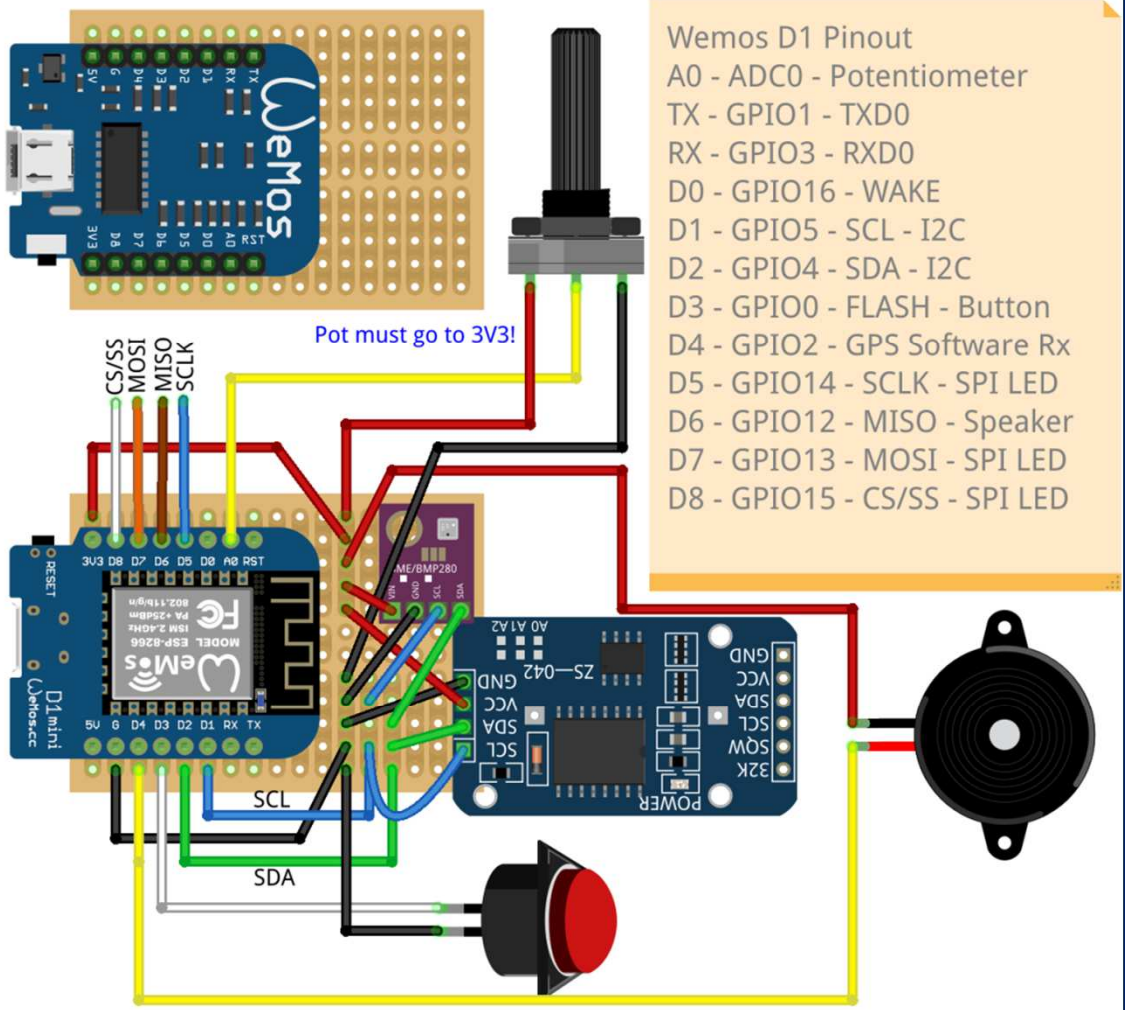
LESSON REFERENCE



LESSON REFERENCE

Pin Assignment Notes
GPIO16/D0 - HIGH at boot - No interrupt, no PWM or I2C - Unused
GPIO2/D4 - HIGH at boot - Input pulled up, output to onboard LED - - probable GPS RX software serial
GPIO12/D6 - Piezo Speaker (not used in SPI LED Matrix)
GPIO[12],13,14,15/D6,D7,D5,D8 - MISO,MOSI,SCLK,CS - SPI
GPIO4,5/D2,D1 - SDA,SCL - I2C
ADC0/A0 - Analog Input - Potentiometer 3.3V divider
GPIO0/D3 - Input pulled up - FLASH button, boot fails if pulled low - button to ground

SPI - LED matrix : 12,13,14,15
I2C - RTC,BMP280 : 4,5
Serial RX - GPS : 16 SS
Input pullup with interrupt - Button : 0
Piezo Speaker : 2
Analog Input - Potentiometer : ADC0



Wemos D1 Pinout
A0 - ADC0 - Potentiometer
TX - GPIO1 - TXD0
RX - GPIO3 - RXD0
D0 - GPIO16 - WAKE
D1 - GPIO5 - SCL - I2C
D2 - GPIO4 - SDA - I2C
D3 - GPIO0 - FLASH - Button
D4 - GPIO2 - GPS Software Rx
D5 - GPIO14 - SCLK - SPI LED
D6 - GPIO12 - MISO - Speaker
D7 - GPIO13 - MOSI - SPI LED
D8 - GPIO15 - CS/SS - SPI LED