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

Lesson 8 – Producing Sound

- Formatting printed output in Serial Monitor
- Shifting and exponents
- Bitwise operations and masking
- Displaying text on the LED matrix display
- 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 8 – Binary Operations, Matrix Text, and Sound Output

- Formatting numbers in Serial Monitor
- Binary shifting and exponents
- Bitwise operations
- Displaying text on the LED matrix – MD Parola Library
- Classroom Exercise – Scrolling text display!
- Review of sound wave theory
- Analog vs pulse width modulation
- Producing tones with a microcontroller
- Classroom Exercise

Formatting Serial Monitor numbers

- We've studied the three major number systems used in computing: binary, decimal, and hexadecimal
- It's possible to "force" the output of a number printed in Serial Monitor to be one of these data types
- You've already seen us use decimal:

```
Serial.println(foo,DEC);
```

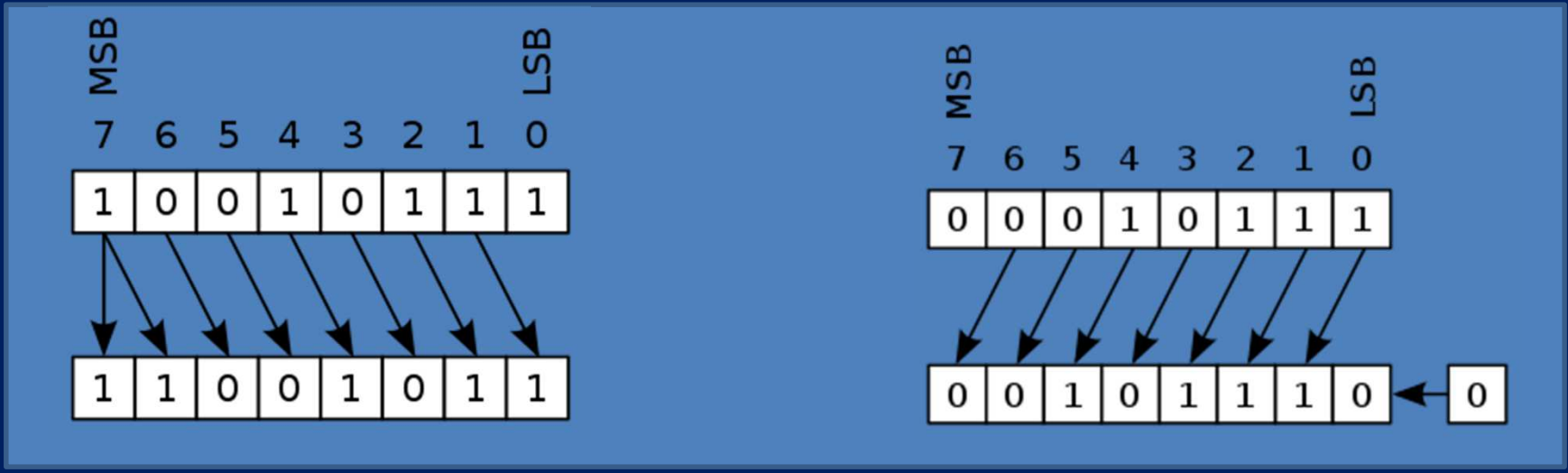
- But there are two more options:

```
Serial.println(foo,BIN);
Serial.println(foo,HEX);
```

- Go ahead and modify SKETCH**6**A to print in binary and hex, as well as decimal!

Binary Shifting and Exponents

- In computer programming, it is frequently useful to perform “bitwise shifts” on numbers, also called “arithmetic shifts”
- In a left shift, all the bits in a number, a byte or integer, say, are shifted one position to the left, and “ones-filled” from the left
- In a right shift, all the bits in a number, a byte or integer, say, are shifted one position to the right, and “zero-filled” from the left
- Because binary is a base-2 radix number system, a left shift multiplies the number by two, and a right shift divides it by two



Bitwise Operations

- The C language allows us to perform “bitwise comparisons” of numbers, comparing each bit one at a time using certain logical operators
- The primary bitwise logical operators are AND, OR, and XOR (exclusive OR)
- In C, the operators for these are &, |, and ^

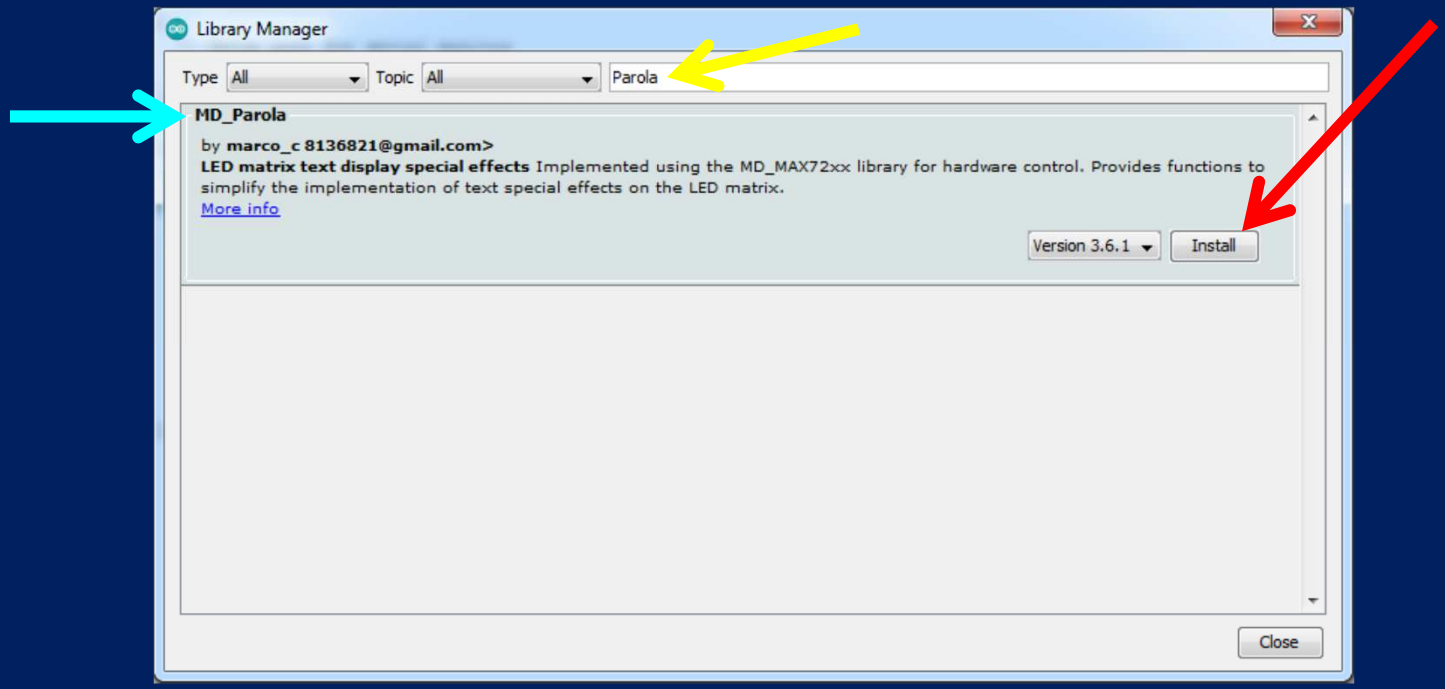
Number 1	1	0	1	0	1
Number 2	1	1	1	0	0
AND	1	0	1	0	0
OR	1	1	1	0	1
XOR	0	1	0	0	1

a	b	a&b	a|b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Caution: Arithmetic exponent and bitwise XOR use the same symbol!

LED Matrix – Displaying Text

- Due to a manufacturing error with the circuit board silk screening, I installed our LED matrices “upside-down” ☹
- As a result, when displaying text, we have to “flip” the display 😐
- This is *very* difficult to do with the MD_MAX72XX library we’ve been using so far
- However, it’s pretty trivial with the MD_Parola library (which itself uses MD_MAX72XX)
- So y’all need to install MD_Parola; go ahead and do that now!



LED Matrix – Sketch 8A

- The Parola library works similar to MD_MAX72XX; you must define your SPI pins, then call a constructor to create an MD_Parola display object.
- In this sketch, the LED display object is called “pmx”

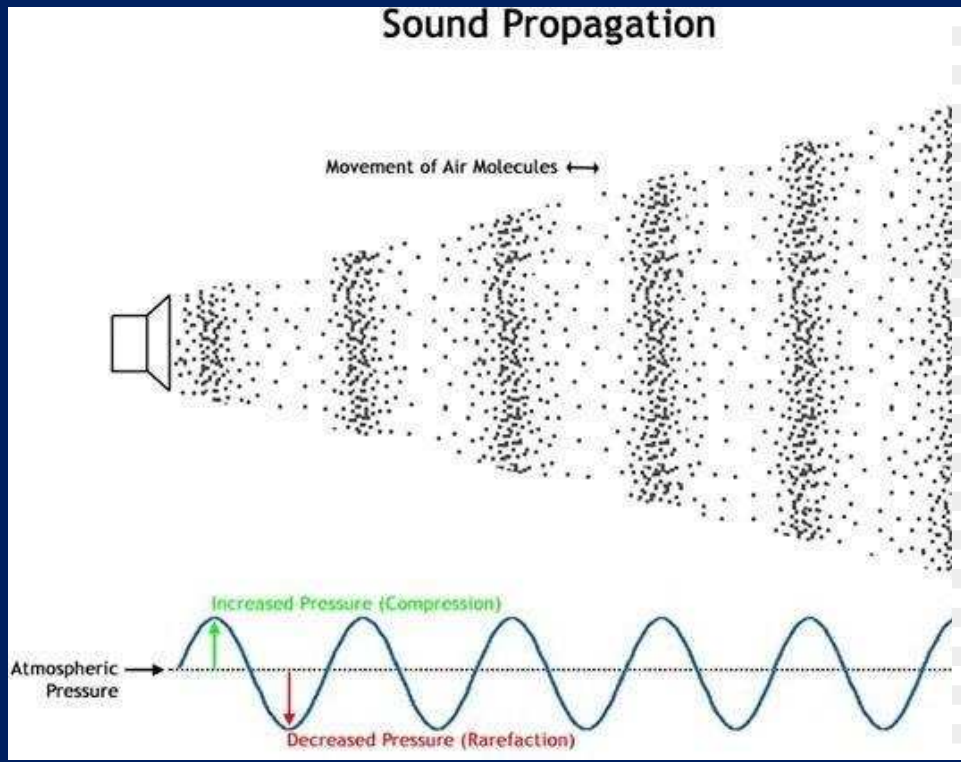
```
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES  4
#define CLK_PIN   14  // or SCK
#define DATA_PIN  13  // or MOSI
#define CS_PIN    15  // or SS
// MD_MAX72XX ledmx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, MAX_DEVICES);
MD_Parola pmx = MD_Parola(HARDWARE_TYPE, CS_PIN, 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:
  pmx.begin();
  pmx.setZoneEffect(0, true, PA_FLIP_LR);
  pmx.setZoneEffect(0, true, PA_FLIP_UD);
}

void loop()
{
  if(pmx.displayAnimate())
    pmx.displayText("SEICHE 2022",PA_LEFT,50,0,PA_SCROLL_LEFT,PA_SCROLL_LEFT);
}
```

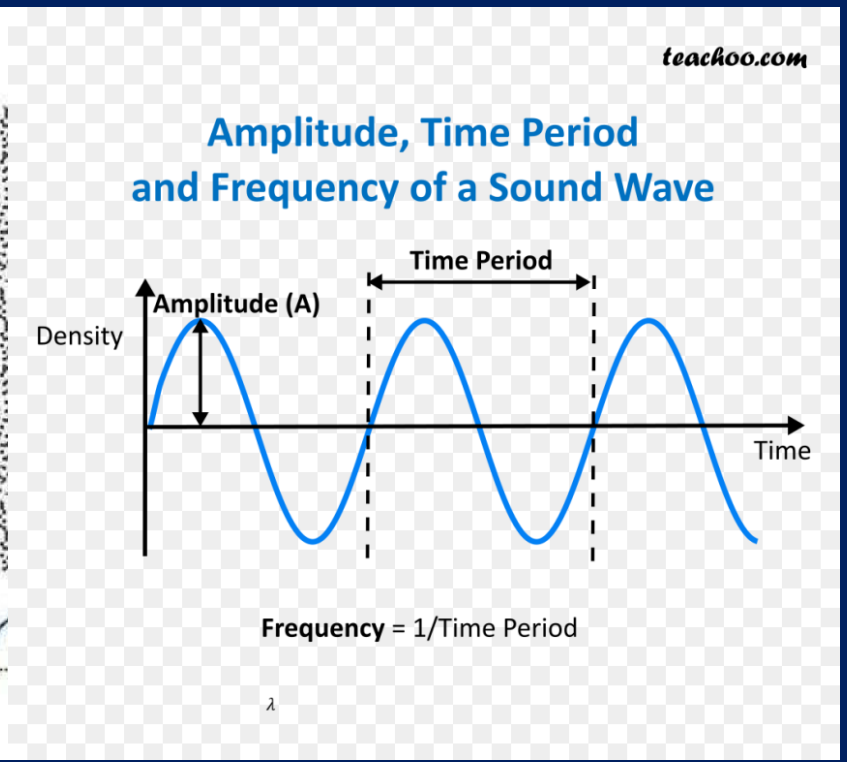
LED Matrix – Parola

- The Parola library is capable of some truly sophisticated animations, both graphics and text
- But the most typical use case is some sort of “scrolling text”, like a so-called “ticker tape display” seen in various public venues
- Take some time now to modify SKETCH8A to display additional scrolling text of your own choice
- There are many other animations available with MD_Parola, feel free to experiment https://majicdesigns.github.io/MD_Parola/_m_d___parola_8h.html#acf3b849a996dbbe48ca173d2b0b82eda
- And you may also want to check out the examples for the MD_Parola library in the IDE
- We will work a lot more with Parola next semester!

Sound Wave Theory - Review

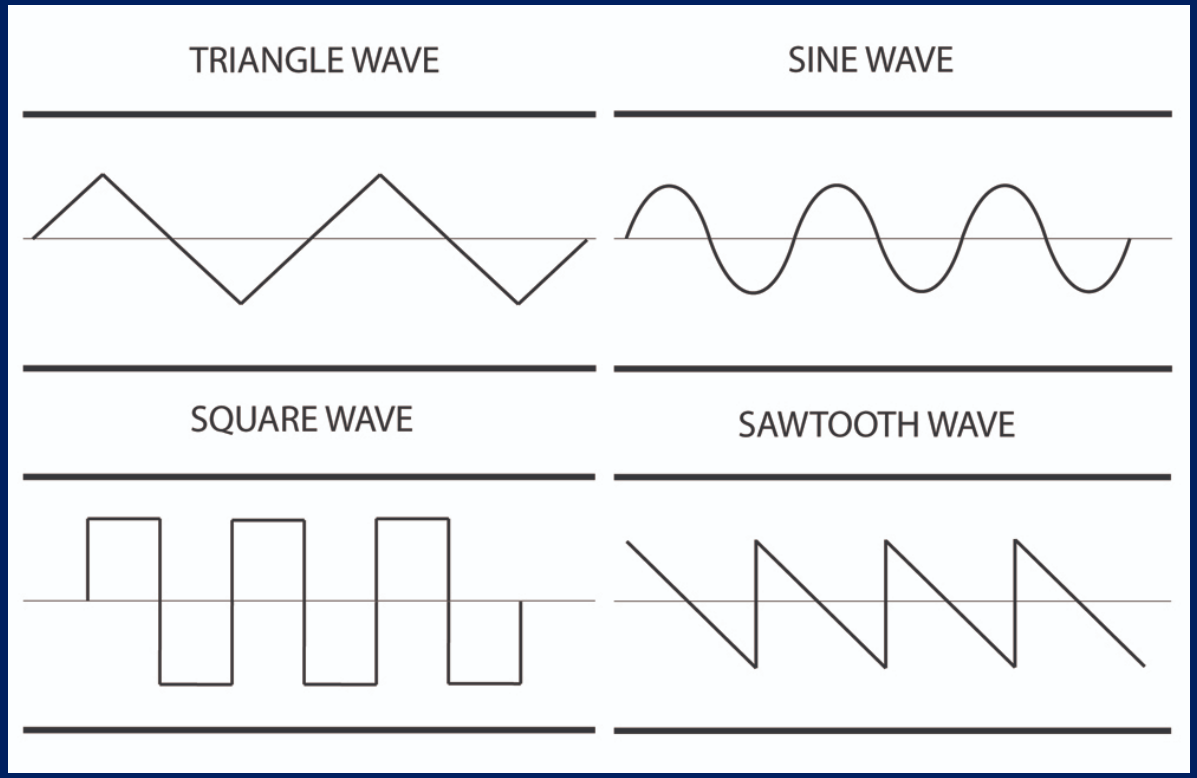
- Sound waves are successive pulses of air pressure
- They have both amplitude (loudness) and frequency (pitch)
- Electronics produce sound waves using some sort of audio transducer, like a speaker





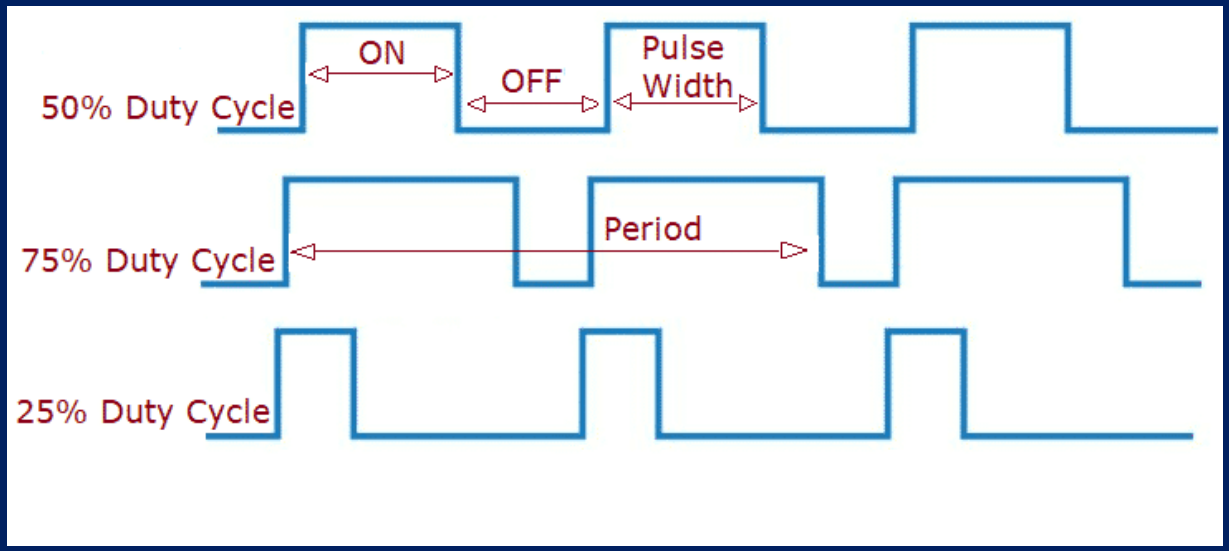
Types of Sound Waves

- There are different types of sound waves, depending on their waveform
- The easiest, and most typical sound wave produced by microcontrollers is the square wave



Pulse Width Modulation

- The method which microcontrollers use to produce sound is called pulse width modulation, or PWM
- The ratio of the amount of time a signal is high vs how long it is low is called duty cycle
- PWM works by changing the duty cycle of an electrical signal
- When this signal is then fed to a speaker, it produces a square wave of varying duty cycle



The Arduino tone() function

- The Arduino C++ language offers a handy function for generating square waves of 50% duty cycle
- This is the `tone()` function
- The formal format of the tone() function is:

```
tone(frequency, duration)
```

- **Frequency** is specified in *Hertz*, or cycles-per-second
- **Duration** is specified in *milliseconds*
- On certain microcontrollers, tone() is non-blocking: processing *continues after the tone is started*.
- What this means in practice is that *each call to tone() must be followed by a delay() of equal duration to that used in the tone() command.*
- But, y'know, feel free to leave that delay out and see what happens! ☺

Let's Make Some Waves!

- Go ahead and open up SKETCH8B and upload it to your displays
- Now, go ahead and adjust the sketch to change any or all of the tones being generated
- A list of frequencies for octaves 3-5 is included at the top of the sketch; can anyone make their display produce a melody using this information?

```
#define note_F2 87.31
#define note_C3 130.81
#define note_C3s 138.59
#define note_D3 146.83
#define note_D3s 155.56
#define note_E3 164.81
#define note_F3 174.61
#define note_F3s 185
#define note_G3 196
#define note_G3s 207.65
#define note_A3 220
#define note_A3s 233.08
#define note_B3 246.94
#define note_C4 261.63
```

```
#define note_C4s 277.18
#define note_D4 293.66
#define note_D4s 311.13
#define note_E4 329.63
#define note_F4 349.23
#define note F4s 369.99
#define note_G4 392
#define note_G4s 415.30
#define note_A4f 415.30
#define note_A4 440
#define note_A4s 466.16
#define note_B4f 466.16
#define note_B4 493.88
#define note_C5 523.25
```

```
#define note_C5s 554.37
#define note_D5s 622.25
#define note_D5 587.33
#define note_E5 659.25
#define note_F5 698.46
#define note_F5s 739.99
#define note_G5 783.99
```

CLASS ASSIGNMENT

Combine SKETCH8A and SKETCH8B

- **Using SKETCH8A and SKETCH8B as references, create a sketch which generates sounds when certain events happen on the display**
- **What those events are, and what tones you generate, are up to you!**

This shouldn’t be terribly tricky, however I’ve structured this as an in-class assignment so that I’m available for any questions

Formal End of Lesson 8

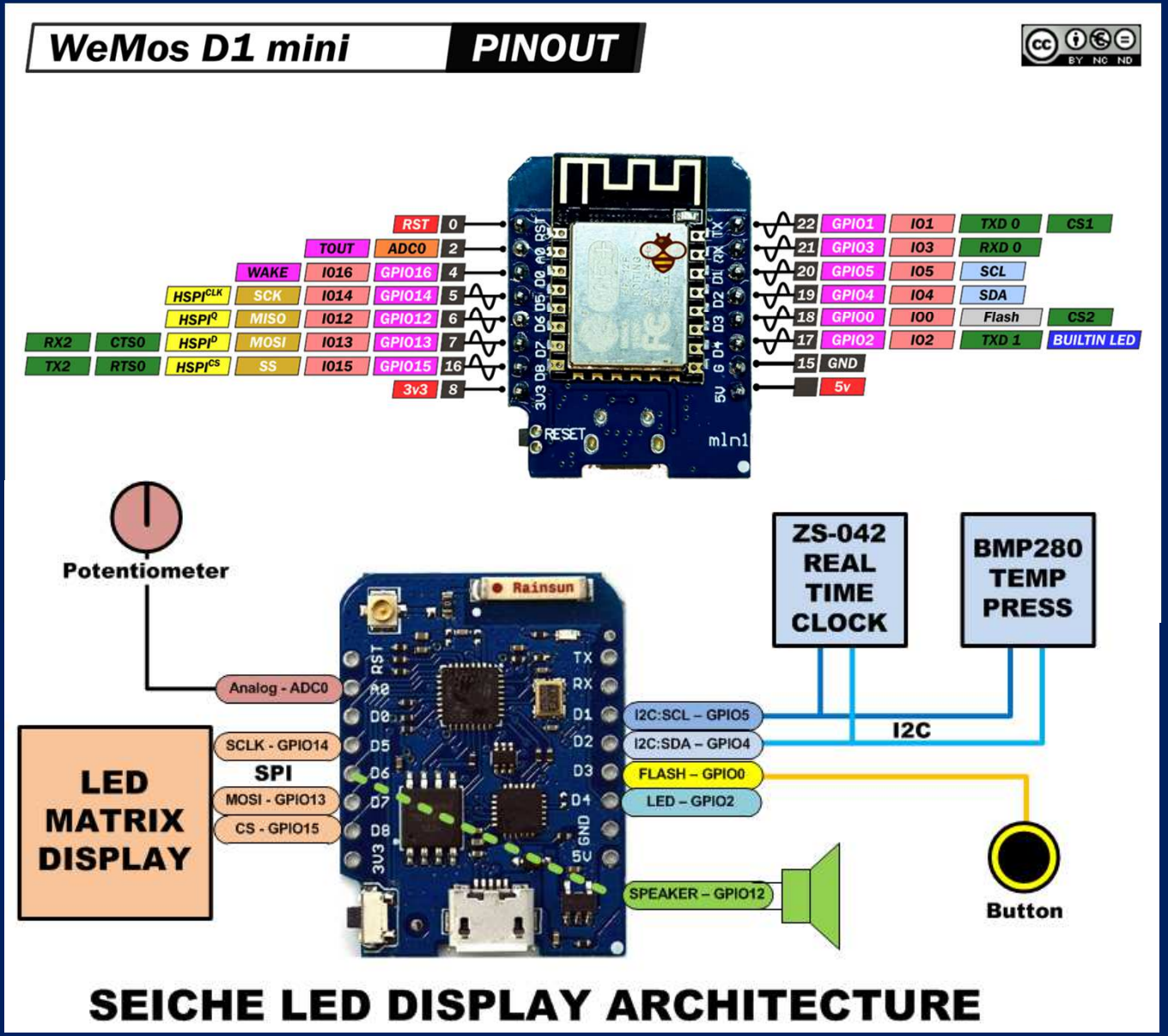
HOMEWORK!

- **No homework this week!**

In next week's exciting episode

- Reading pin values
- Debouncing buttons
- Reading analog values from a potentiometer
- The I2C bus and I2C sensors
- Reading I2C sensors

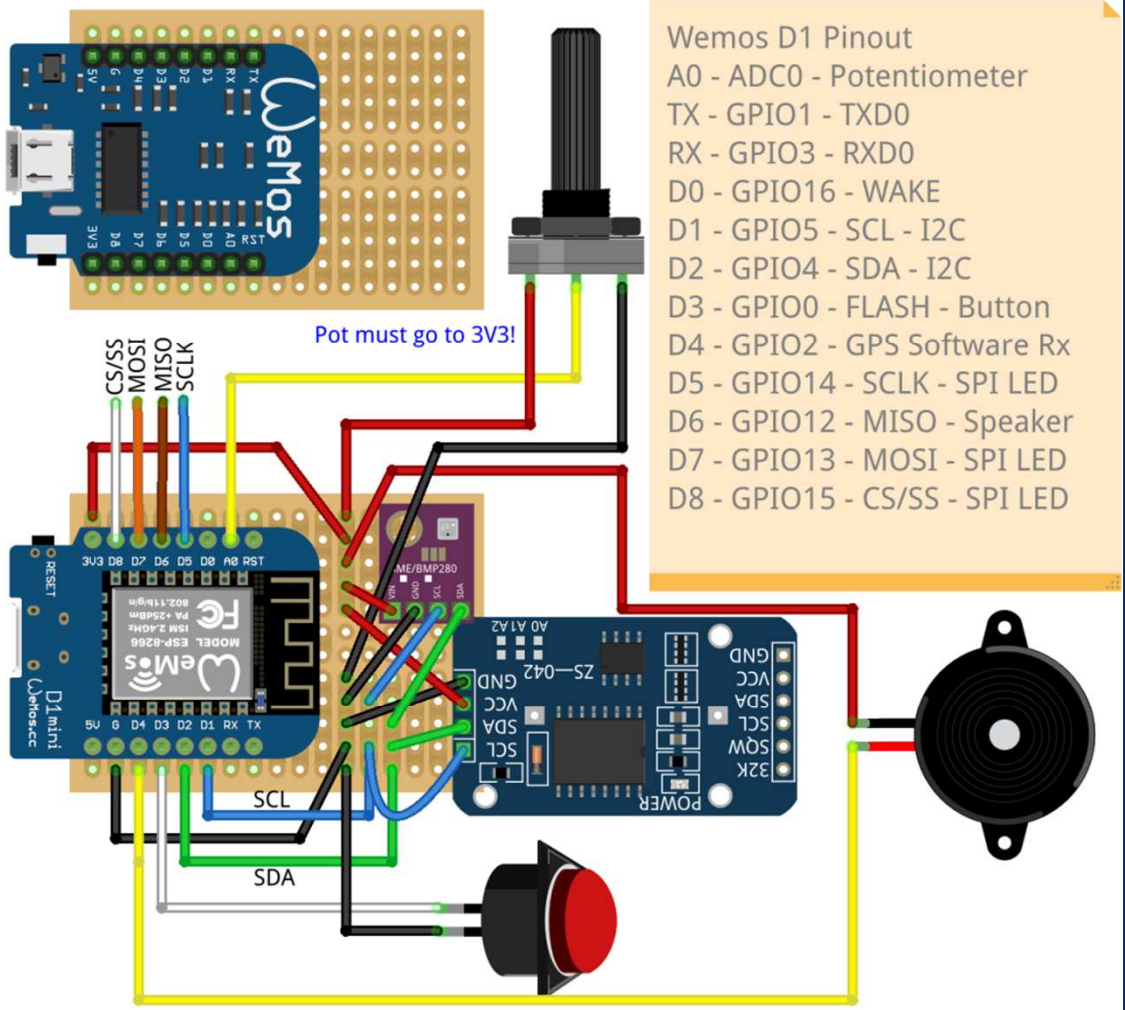
LESSON REFERENCE



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