SEICHE 2022
Basic Arduino Programming

Instructor: Paul Frommeyer

www.paulfrommeyer.com

Corporate Sponsor: DXC Technology

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 Analog and Digital pins

- Millis
- Reading buttons
- 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
- Real Time Clocks
- Accessing a DS3231 RTC

Lesson 11 – NTP and Text Management

- Time representations and conversions
- Network Time Protocol
- Displaying the time on an LED matrix display
- Changing the default font
- Text Effects
- Using multiple display zones

Lesson 10 – I2C and Sensors

- Parola followup – library syntax
- Displaying plain text with the Parola library
- The I2C bus – Theory of operation
- LED Matrix Display – I2C review
- Classroom Exercise – Temperature sensor
- Real Time Clocks
- Accessing a DS3231 RTC
- Classroom Exercise – DS3231 Display

MD_Parola Syntax Review

- The MD_Parola library method `setIntensity(uint8_t intensity)` changes the brightness of your LED display
- **This breaks down as:**

uint8_t	**This is the data type of the parameter**
intensity	**This names the parameter itself**

- ***Methods must be called from/in their objects!***
- **So as an example:**

```
uint8_t  display_intensity=6;
pmx.setIntensity(display_intensity);
```

- **"pmx" is the Parola display object that you created earlier in the sketch; it's what you set intensity *for*.**

Plain text with Parola

- As we saw previously, there are many text effects available with Parola. But what if we just want to put some text on the LED matrix without any fancy animations?
- **For that, we use the PA_PRINT text effect. This effect "just prints" the specified text**
- **An example of PA_PRINT:**
  `pmx.setTextEffect(PA_PRINT, PA_NO_EFFECT);`
- **That function call specifies the plain PRINT effect as the "text on" effect for all display zones and NO_EFFECT for "text off" for all display zones**

Display Plain Text – Sketch 10A

- Go ahead and load SKETCH10A into your IDE and upload it
- Once again, feel free to modify the text you are displaying to your personal preference

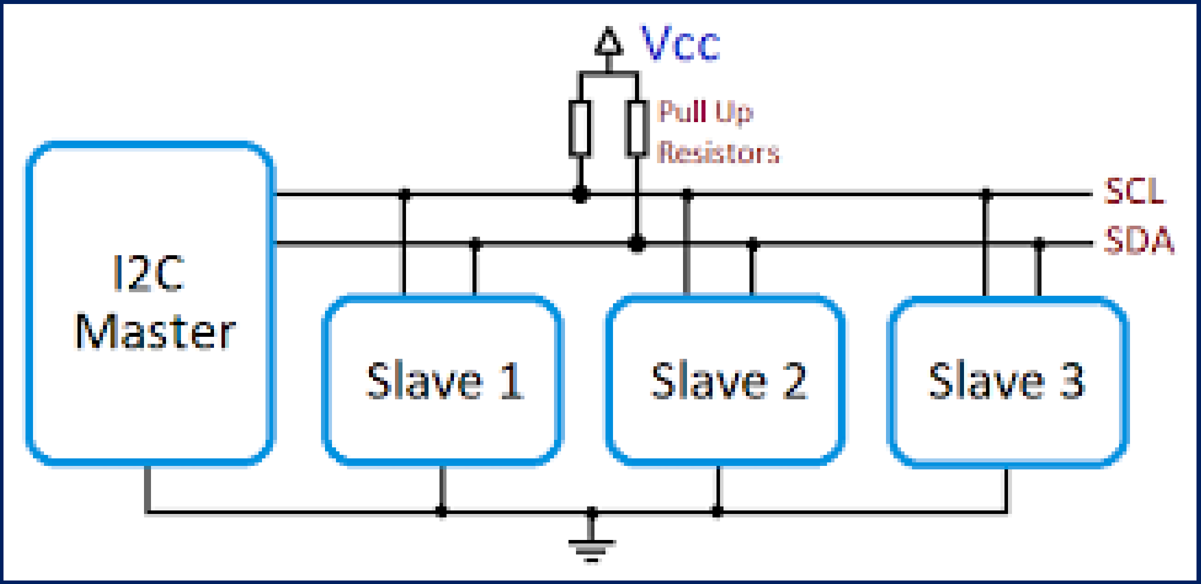
```
#include <SPI.h>
#include <MD_MAX72xx.h>
#include <MD_Parola.h>

#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);
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",PA_CENTER,50,0,PA_PRINT,PA_NO_EFFECT);
}
```

The I2C Bus

- Recall that a “bus” in electronics is a group of wires/circuits all carrying related signals that work together to jointly perform certain operations
- Last semester, we looked at WS2812 smart LEDs (“Neopixels”) which used the so-called OneWire or Wire single-wire signal bus to transmit color and position data to a strip of smart LEDs
- We have also looked at the SPI (Serial Peripheral Interface) bus which comprises five wires to access and control active devices connected to microcontrollers (such as our LED matrix displays)
- Today we are going to learn how to use the I2C bus. The I2C bus is a two-wire bus. I2C is an abbreviation (ofc) for Inter-Integrated Circuit. This name was chosen because the I2C bus is designed to allow active electronic components to communicate with each other over (relatively) short distances, namely the distances within a single electronics device such as a computer, MP3 player, or game console.
- The I2C bus uses data and clock wires, and I2C is a master-slave architecture (no social commentary intended or implied) where a master control unit handles all communications on the bus.

The I2C Bus

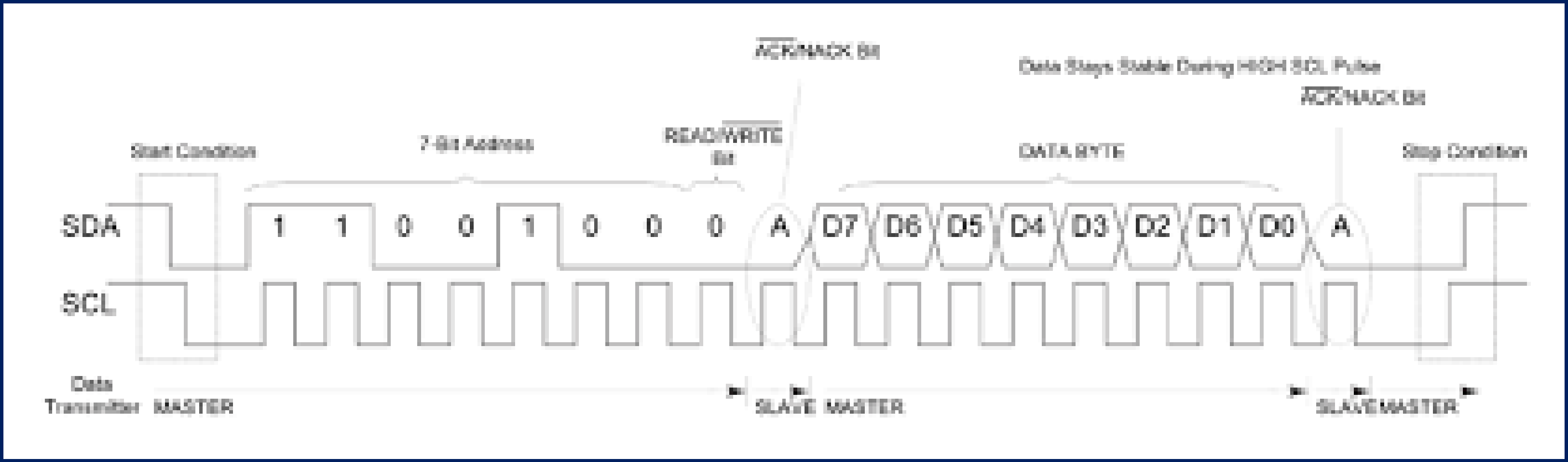
- The master unit is responsible for generating the clock signal on the Synchronous Clock (SCL) wire, issuing commands on the Synchronous Data (SDA) wire, and receiving data back (when applicable) from slave units



- Slave units have addresses (registers holding binary data) and they listen for commands from the master sent to their address, then they execute those commands

The I2C Bus

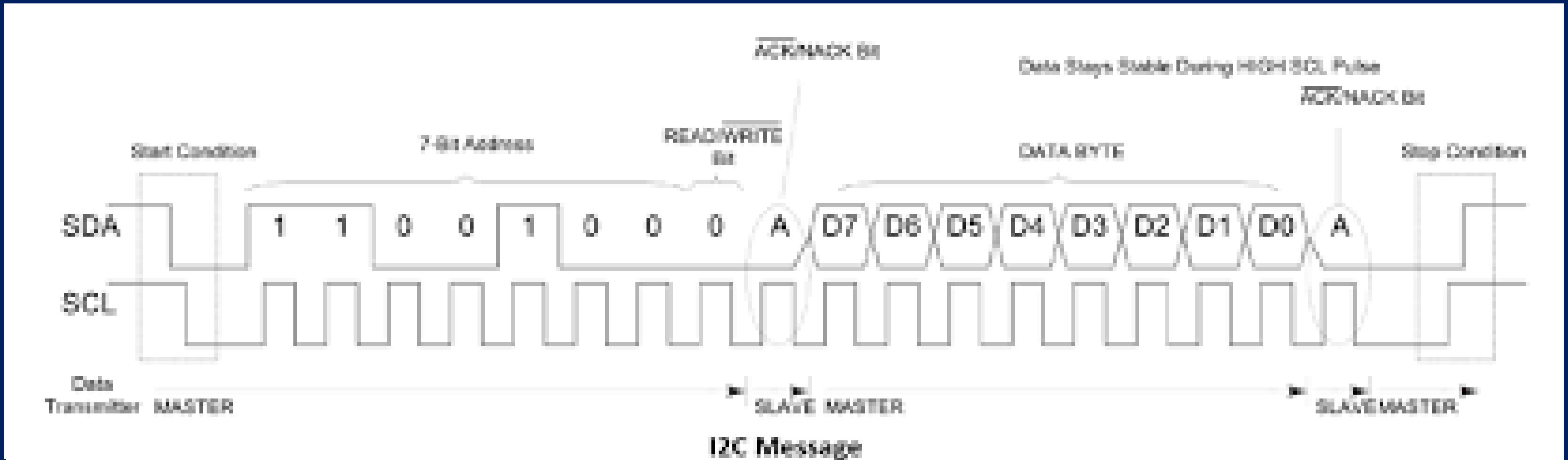
- The below diagram, which we will not be reviewing in detail ☺, shows how the SCL clock works with commands sent from the master and data returned from the slaves

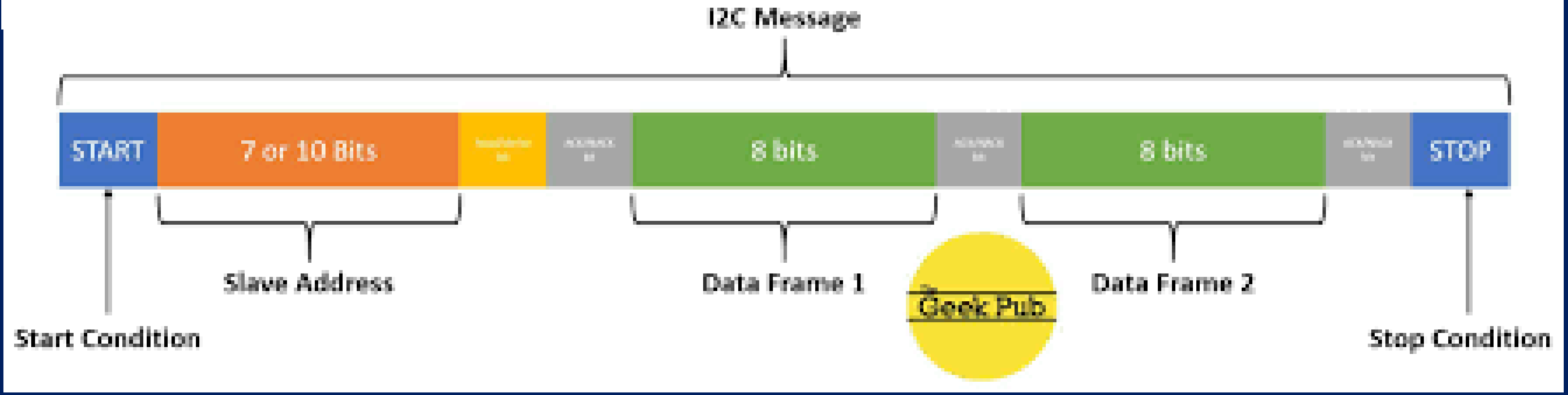


- The TL;DR is that during certain clock states the master sends data, and during other clock states the slaves send data

The I2C Bus

- The below diagrams, which we will *not* be reviewing in detail ☺, shows how the SCL clock works with commands sent from the master and data returned from the slaves
- The TL;DR is that during certain clock states the master sends commands, and during other clock states the slaves send back data



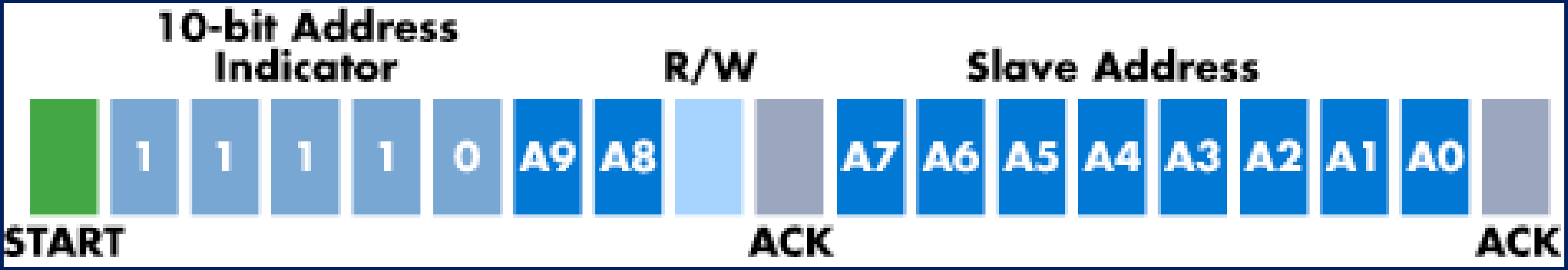


The I2C Bus

- The critical information you need as a programmer/developer in order to use I2C peripherals is how slave addressing works
- Originally, only 8 bits were used for slave addresses, permitting (wait for it!) up to 255 devices to exist on a single bus
- Quite predictably, this proved woefully insufficient for large devices such as server motherboards, so a hack was contrived that allowed the use of 10-bit addresses, with the possibility of expanding to even larger address ranges in the future

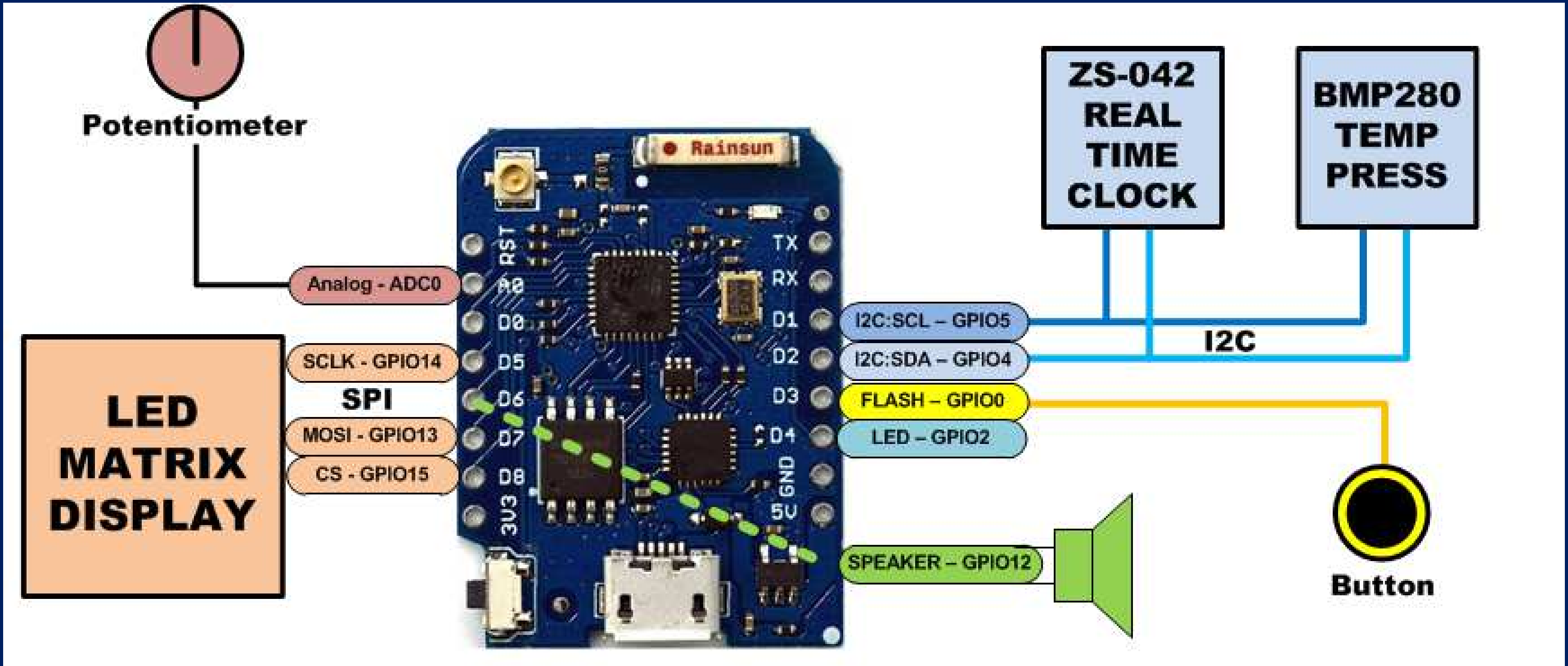
The I2C Bus - Addressing

- The way the 10-bit addressing hack works is that not one but two addressing fields are transmitted by the master.
- A bit-challenged slave will ignore the first field because it’s invalid, and only pay attention to the second
- While a modern, sophisticated slave device is smart enough to parse both fields and extract the full 10-bit address.
- With this scheme, only the master has to be fully aware of how the 10-bit addressing hack works.
- 8-bit and 10-bit slaves will happily co-exist together, as long as the cardinal rule of unique device addresses is followed



LED Matrix Display – I2C Bus

- Review the graphic for the signal architecture used in your LED matrix displays, and notice how the I2C bus is structured
- What pins does the I2C bus use on the ESP8266?
- What devices are connected via I2C?



SEICHE LED DISPLAY ARCHITECTURE

LED Matrix Display – I2C Bus

- Temperature Sensors
  - Your displays exclusively use the BMP180 sensor. (Yeah, the graphic is wrong– that's because I ended up changing components at the last second)
  - BME680, BME280, BMP280, BMP180, and BMP085 sensors *can* use the same address, *however they require completely different libraries.*
  - So if you have more than one on a single bus, the addresses *must* be different, even if they are different model sensors

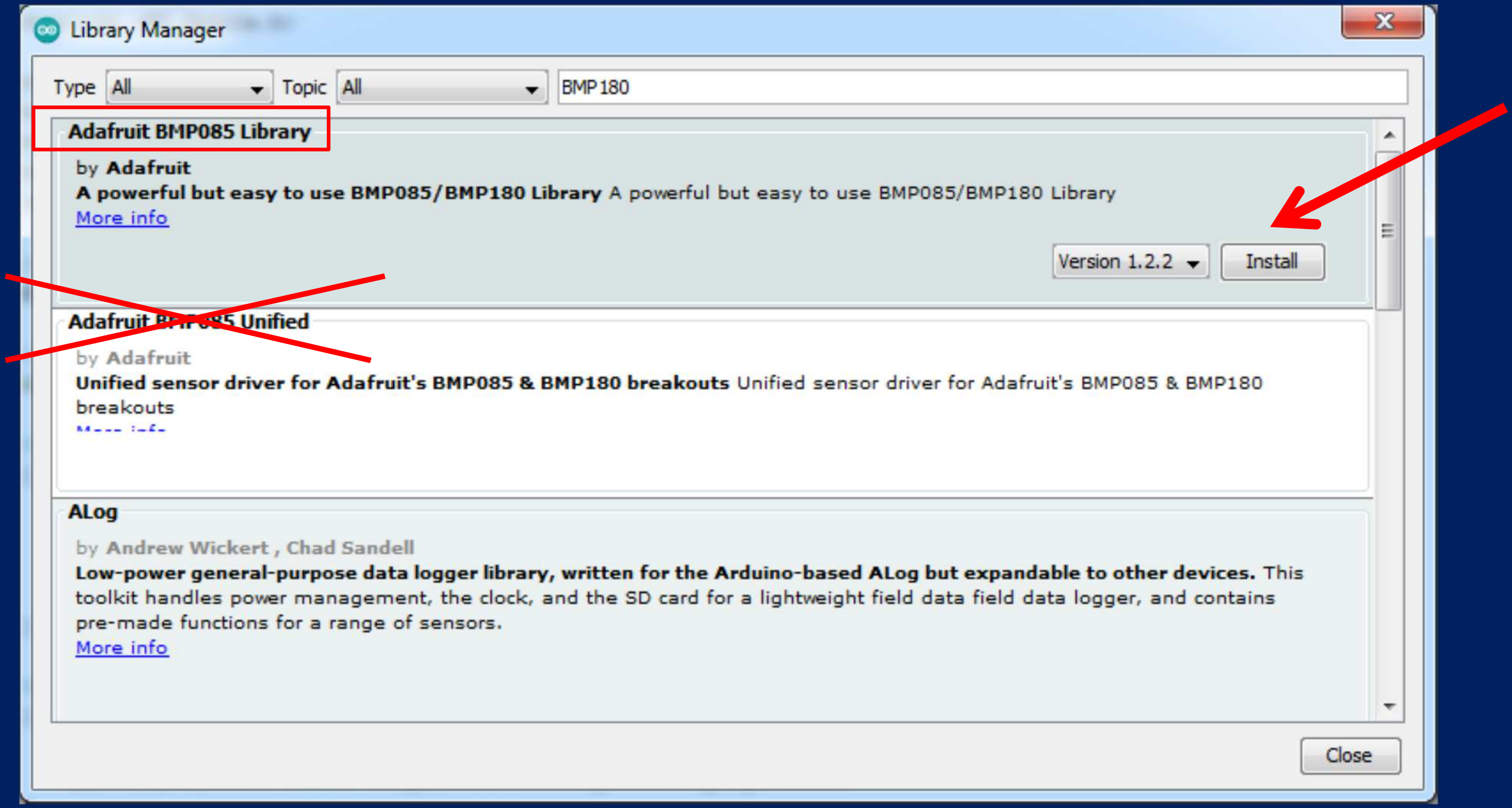
Sensor Model	Addr Width	Type	I2C Address
BMP085	**7-bit**	**Tmp**	**0x77**
BMP180	**7-bit**	**Tmp, Bar**	**0x77**
BMP280	**10-bit**	**Tmp, Bar**	**0x76 or 0x77 (selectable)**
BME280	**10-bit**	**Tmp, Bar, Hum**	**0x76 or 0x77 (selectable)**
BME680	**10-bit**	**Tmp, Bar, Hum, VOC**	**0x76 or 0x77 (selectable)**

Ref: https://randomnerdtutorials.com/dht11-vs-dht22-vs-lm35-vs-ds18b20-vs-bme280-vs-bmp180/#:~:text=The%20BME280%20and%20BMP180%20are,be%20used%20to%20estimate%20altitude.

Read and Display Temperature Sketch 10B

- Go ahead and load SKETCH10B into your IDE (*do* ***not*** upload it!)
- How is your BMP180 temperature sensor initialized and accessed according to what you see in the code?
- What happens when you ***validate*** the sketch?
- If you receive an error message, do you remember how you solve such a message?

Sketch 10B



BMP180

Adafruit BMP085 Library

by **Adafruit**

A powerful but easy to use BMP085/BMP180 Library A powerful but easy to use BMP085/BMP180 Library

More info

~~**Adafruit BMP085 Unified**~~

by Adafruit

Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts

ALog

by Andrew Wickert , Chad Sandell

Low-power general-purpose data logger library, written for the Arduino-based ALog but expandable to other devices. This toolkit handles power management, the clock, and the SD card for a lightweight field data field data logger, and contains pre-made functions for a range of sensors.

More info

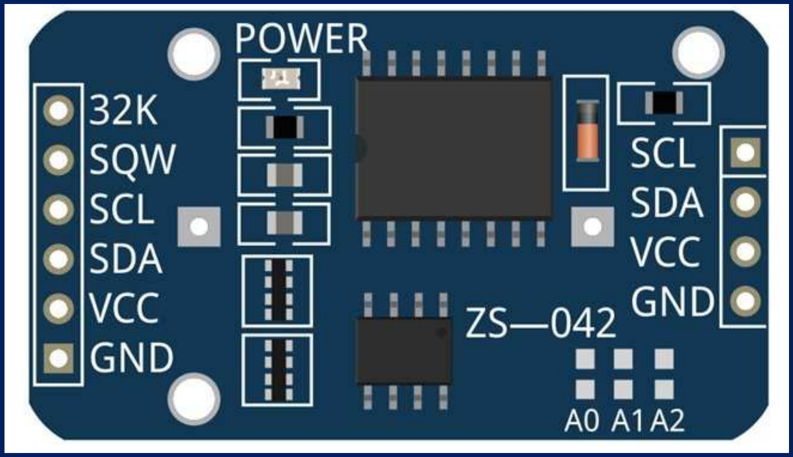
- **Once you have installed the correct library, *now* upload the sketch**
- **Verify in the Serial Monitor that you are able to retrieve output from your BMP180**
- ***Next,* modify your sketch to display the temperature output on your LED matrix**
- **And *then*, modify your sketch so the LED display is in *degrees Fahrenheit*, not Celsius**

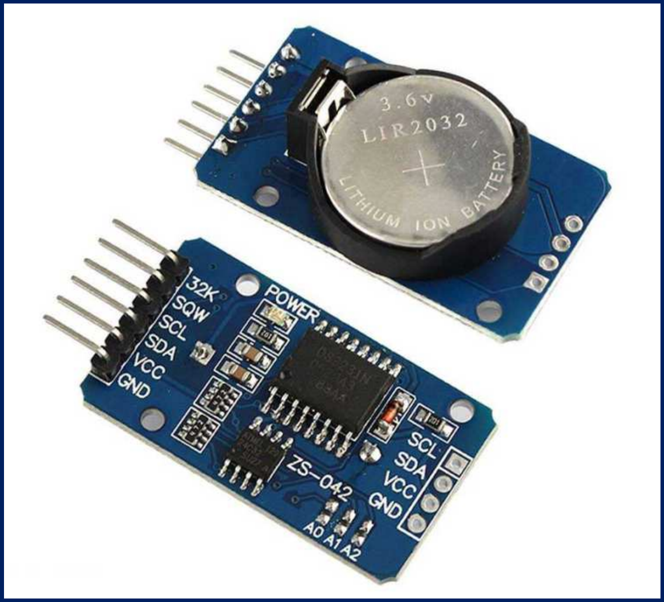
Real Time Clock

A real-time clock (RTC) is an electronic device (most often in the form of an integrated circuit) that measures the passage of time. The term real-time clock is used to avoid confusion with ordinary hardware clocks which are only signals that govern digital electronics, and do not count time in human units. RTCs often have an alternate source of power, so they can continue to keep time while the primary source of power is off or unavailable. Most RTCs use a crystal oscillator,[8][9] but some have the option of using the power line frequency.[10] The crystal frequency is usually 32.768 kHz,[8] the same frequency used in quartz clocks and watches. Being exactly 215 cycles per second, it is a convenient rate to use with simple binary counter circuits. The low frequency saves power, while remaining above human hearing range. The quartz tuning fork of these crystals does not change size much from temperature, so temperature does not change its frequency much.

– Wikipedia

Real Time Clock – DS3231

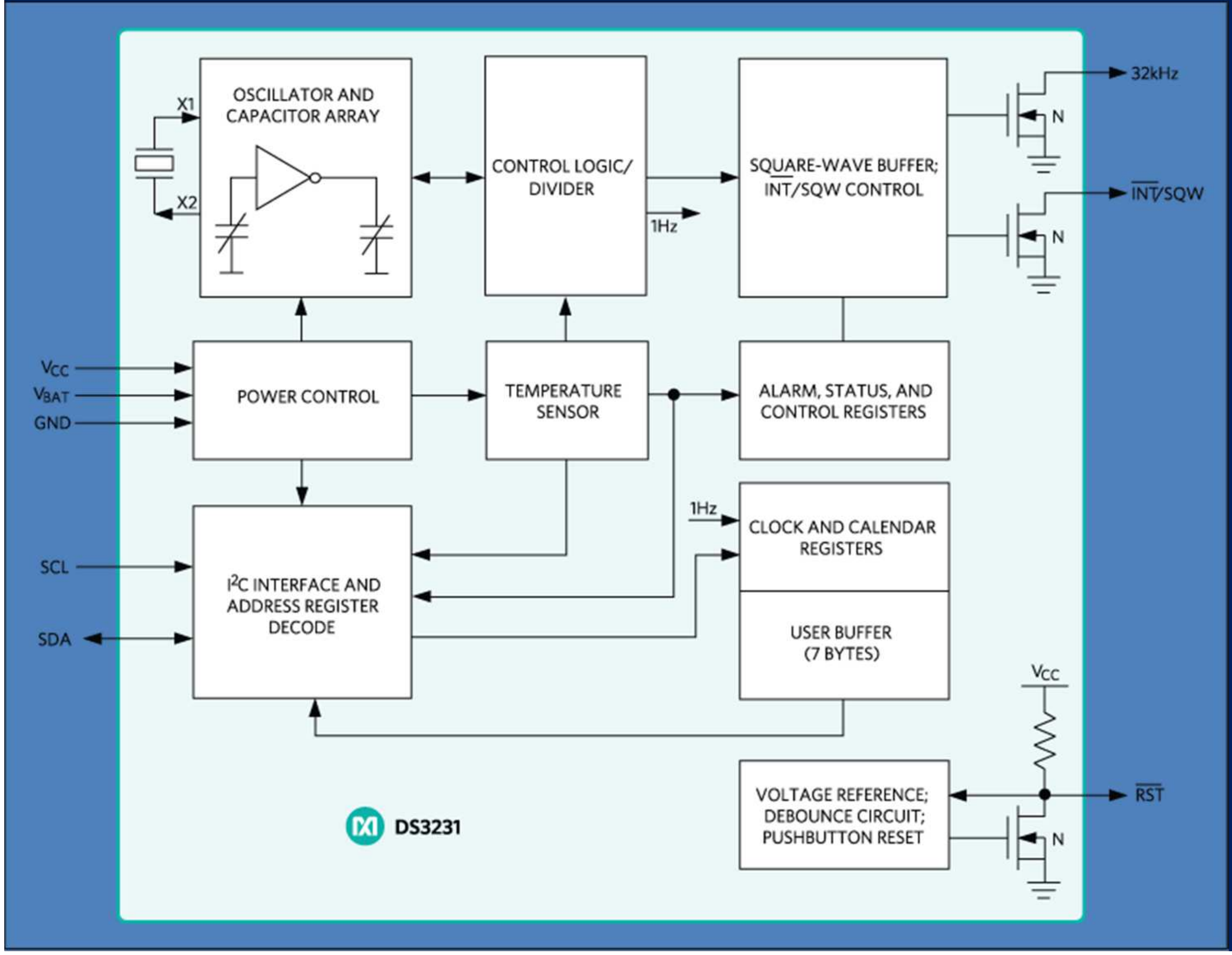




- The DS3231 is a very common RTC
- It is often found in the ZS-042 form factor– a specific circuit board and component layout that provide I2C access to the DS3231

The DS3231 Real Time Clock

The datasheet for the DS3231 explains that this part is an "Extremely Accurate I²C-Integrated RTC/TCXO/Crystal". And, hey, it does exactly what it says on the tin! This Real Time Clock (RTC) is the most precise you can get in a small, low power package. – Adafruit

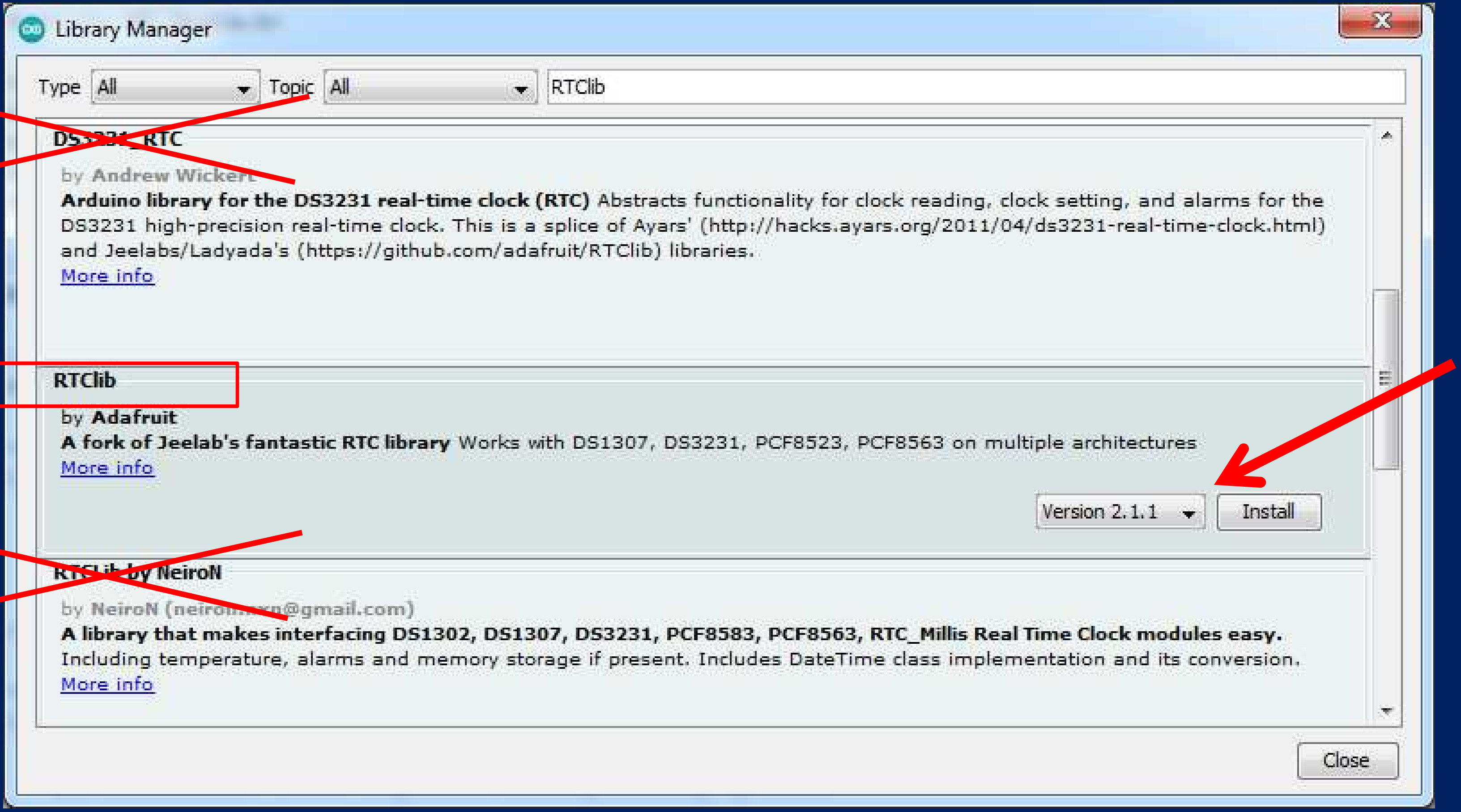

OSCILLATOR AND CAPACITOR ARRAY
X1
X2
CONTROL LOGIC/ DIVIDER
1Hz
SQUARE-WAVE BUFFER; INT/SQW CONTROL
32kHz
N
INT/SQW
N
VCC
VBAT
GND
POWER CONTROL
TEMPERATURE SENSOR
ALARM, STATUS, AND CONTROL REGISTERS
1Hz
CLOCK AND CALENDAR REGISTERS
SCL
SDA
I²C INTERFACE AND ADDRESS REGISTER DECODE
USER BUFFER (7 BYTES)
VCC
RST
VOLTAGE REFERENCE; DEBOUNCE CIRCUIT; PUSHBUTTON RESET
N
DS3231

- **The block diagram of the DS3231 from its manufacturer, Maxim Electronics, is at left**
- **Likewise per the datasheet, the DS3231 default I2C address is 0x68, and the I2C address of the on-board EEPROM is 0x57**

Accessing a DS3231 RTC

- You now have the I2C address of the DS3231 used in your displays
- What else do you need before you can access and use the RTC?
- Go ahead and load SKETCH10C
- What happens when you *validate* the sketch?
- How do you propose we solve *this* error message?

Choosing a DS3231 RTC Library



Library Manager

Type: All | Topic: All | RTClib

~~DS3231_RTC~~

~~by Andrew Wickert~~

Arduino library for the DS3231 real-time clock (RTC) Abstracts functionality for clock reading, clock setting, and alarms for the DS3231 high-precision real-time clock. This is a splice of Ayars' (http://hacks.ayars.org/2011/04/ds3231-real-time-clock.html) and Jeelabs/Ladyada's (https://github.com/adafruit/RTClib) libraries.
More info

RTClib

by **Adafruit**

A fork of Jeelab's fantastic RTC library Works with DS1307, DS3231, PCF8523, PCF8563 on multiple architectures
More info

Version 2.1.1 | Install

~~RTClib by NeiroN~~

~~by NeiroN (neiron.nxn@gmail.com)~~

A library that makes interfacing DS1302, DS1307, DS3231, PCF8583, PCF8563, RTC_Millis Real Time Clock modules easy. Including temperature, alarms and memory storage if present. Includes DateTime class implementation and its conversion.
More info

Close

Select and install the Adafruit RTClib library– and ONLY the Adafruit library!

Read and Display RTC Time Sketch 10C

- Go ahead and upload Sketch 10C into your display (after completing the RTClib installation on the previous slide!)
- What Serial Monitor output do you see?
- What does your display show?
- How does your display relate to what's show in the Serial Monitor
- Could anything be wrong or need to be changed?
- Are there any parts of your sketch that introduce new concepts? If so, what are they?

Formal End of Lesson 10

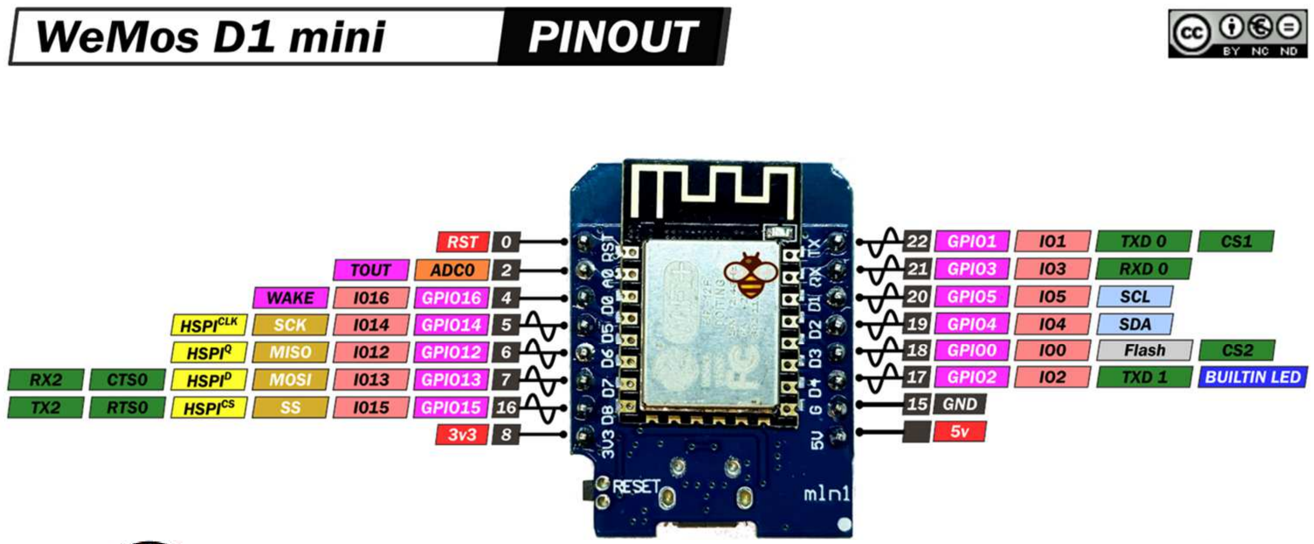
HOMEWORK!

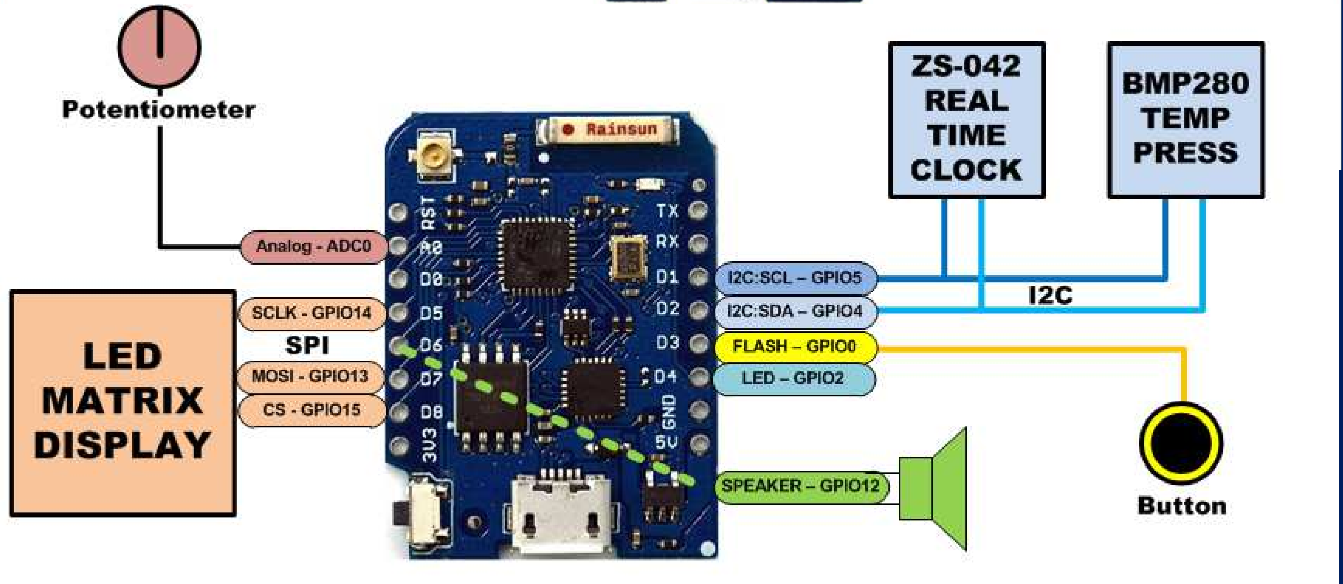
- **Go online and study how DS3231 RTCs are used with Arduinos**
- **Also search online for how time is displayed on MAX7219 LED matrix displays**

In next week's exciting episode

- Time formats and conversions
- Network Time Protocol
- Getting and displaying network time
- Setting an RTC manually
- Setting an RTC from NTP

LESSON REFERENCE



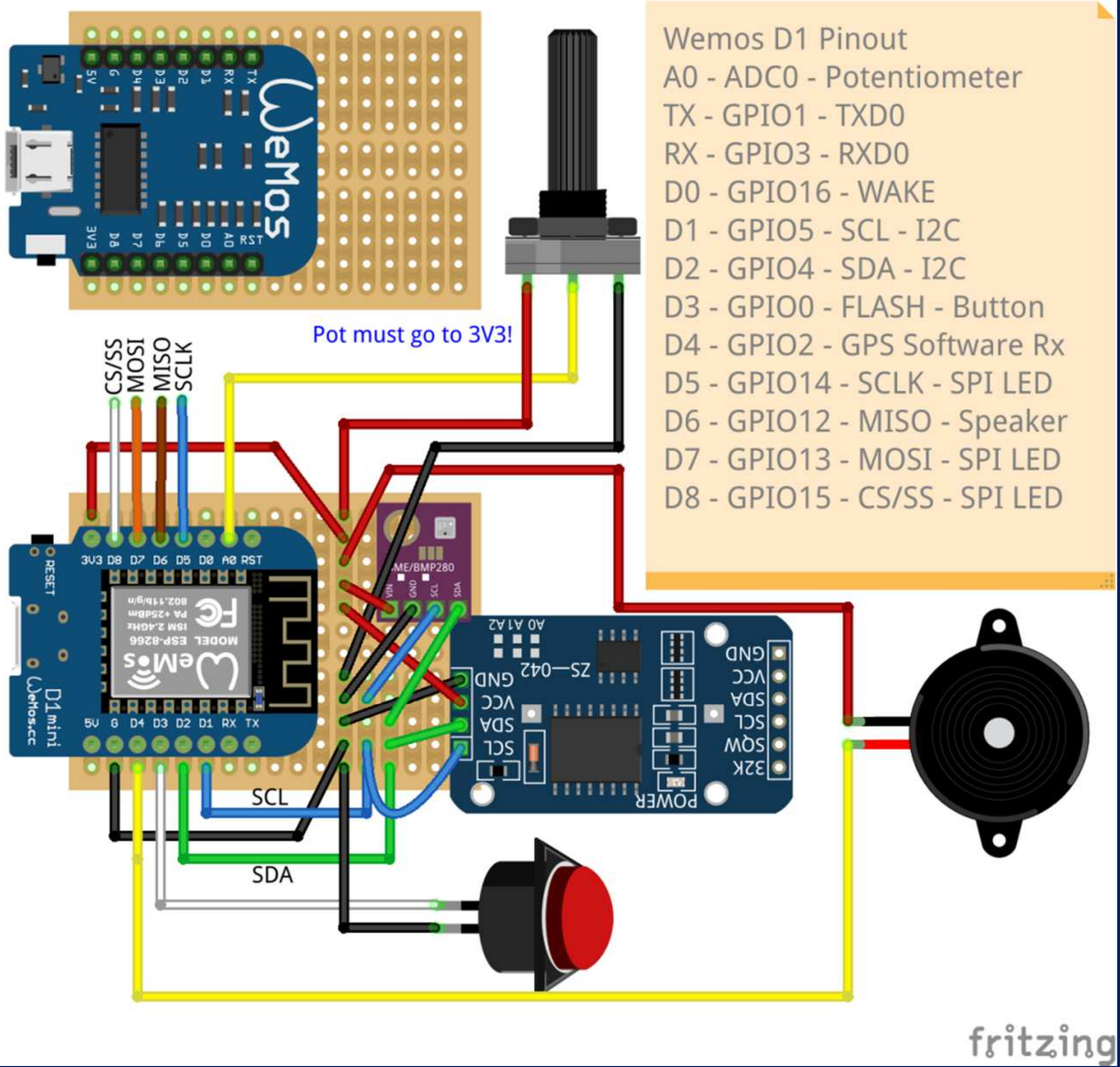


SEICHE LED DISPLAY ARCHITECTURE

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