

LECTURE VI

Communication Protocols II

This work by the Institute of Electrical and Electronics Engineers, UC Irvine Branch, is licensed under CC BY-NC-SA 4.0

SECTION I

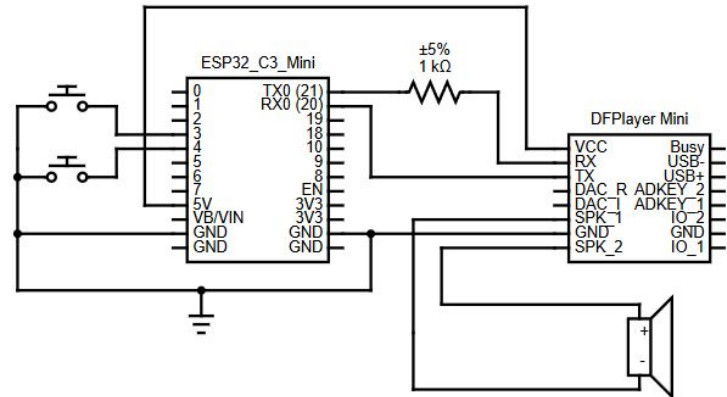
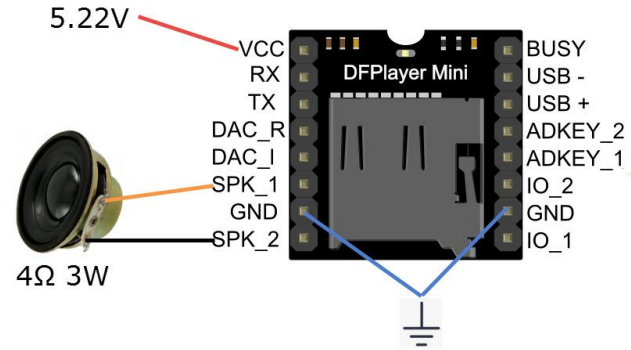
Project V Review

Project 5 Review

- Build and program a **music player** with ESP32, customized to play **your own tunes**.
- Due date: 1/24/2025 at 11:59PM

Learning Concepts:

- Communication Protocols
- UART
- Pull-Up Resistor Circuits
- Serial Library

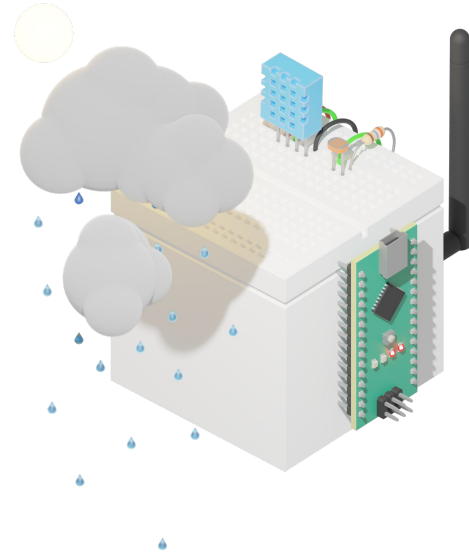


Project 6

- Build a **weather station** that wirelessly transmits **temperature** and **humidity data** to an indoor display.
- Due date: 2/14/2025 at 11:59PM

Learning Concepts:

- Communication Protocols Pt. 2
- I²C Protocol
- SPI Protocol
- Arduino IDE Libraries (Continued)



SECTION II

OPS Project Highlight: “Rockquestor” by Dane Hobrecht

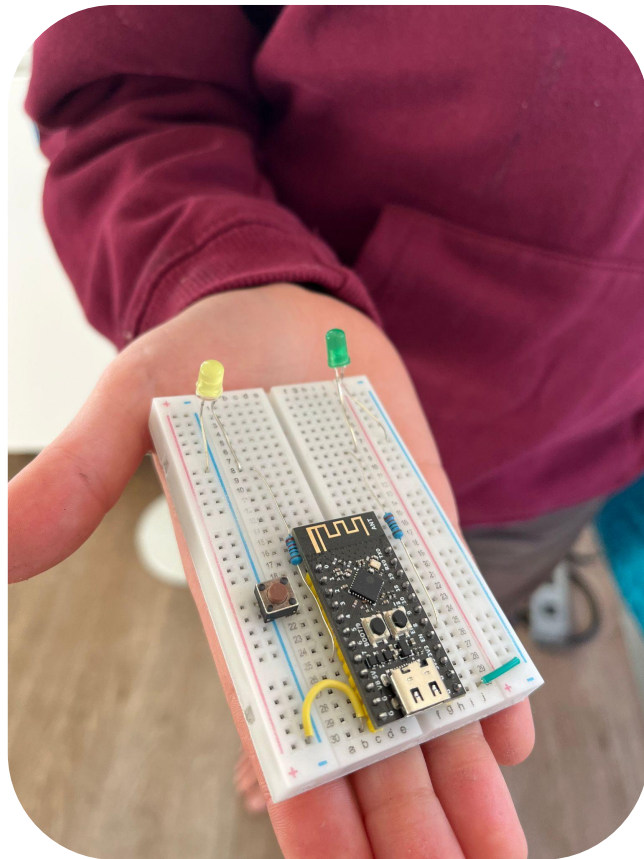
The Problem



- “My little brother, Rocky, is 9 years old, and isn't allowed to digitally communicate on his own yet.”
- “He lives a couple cities away, and in order to hang out, we've had to go through my dad's cell phone as a medium.”

The Solution

- “I was tired of our communications being dependent on my dad's availability, so I made him the **"Rockquestor"** (Rocky + Requestor = 'Rockquestor').”
- “When he wants to hang out, he presses the button, which sends me an email notification and illuminates the Yellow LED, confirming his request was sent.”
- “When I see the email, I go onto a web dashboard and remotely enable the other LED, telling him that I've seen his 'rockquest' and that I'm ready to hang out!”



The Solution

```
void loop() {  
    // put your main code here, to run repeatedly:  
    Blynk.run();  
    if (digitalRead(BTN_BRN) == 0) {  
        digitalWrite(LED_YLW, HIGH);  
        Blynk.logEvent("hang_out"); // Trigger Blynk event to send email notification  
        delay(1000);                // Debounce, prevent spam, and extend LED illumination  
    } else {  
        digitalWrite(LED_YLW, LOW);  
    }  
}
```

- It uses the **Blynk** service for connecting IoT hardware to the cloud.
- It waits for a particular log event (in this case, hang_out) and triggers an

email notification:

Blynk

12/28/2024

Rockquestor: Rocky wants to hang out!

Rocky wants to hang out! Open in the app --

Date: Saturday, December 28, 2024,

The Result



- “He's already used it once so we could go see a movie together. OPS was definitely the reason I was able to get it off the ground so quickly.”

SECTION III

UART Review

UART Protocol

- UART is a **communication protocol with certain characteristics**. It has rules for data transmission:
 - UART can be configured to **___-duplex, ___-duplex, and ___plex modes**
 - There are only **___ lines** (electrical connections)
 - The data transmission speed is determined by a **___ rate**
 - It is quite (**slow/fast?**) **compared to other protocols**

UART Protocol

- UART is a **communication protocol with certain characteristics**. It has rules for data transmission.
 - UART can be configured to **full-duplex, half-duplex, and simplex modes**
 - There are only **two lines** (electrical connections)
 - The data transmission speed is determined by a **baud rate**
 - It is quite **slow compared to other protocols**

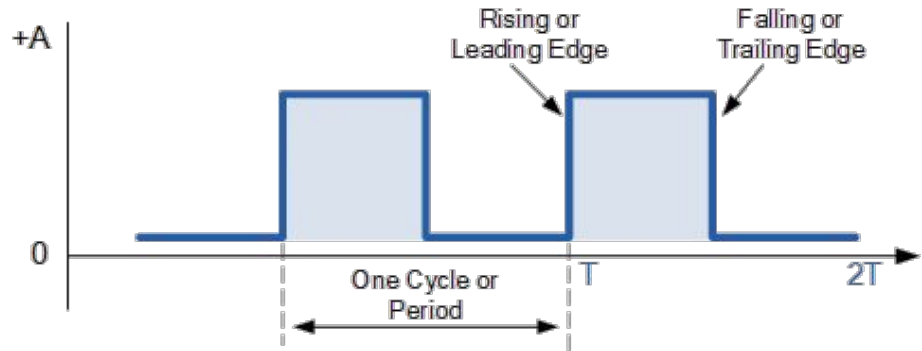
SECTION IV

SPI



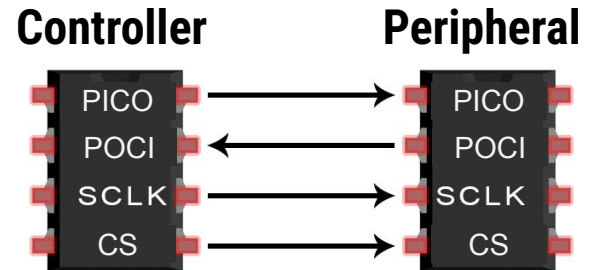
Clock Signals

- **Digital signals** are timed to a **clock signal** - most often, a square wave with a 50% duty cycle
 - For example, in a serial signal, one bit of data is sent per clock cycle
 - The bit might be timed to the rising or falling edge of the clock signal
- **Clock speed** determines the rate at which data is transmitted
 - measured as **frequency (Hz)**
 - inversely proportional to cycle (or period)



SPI Protocol

- **Serial Peripheral Interface** or **SPI** is a serial communication protocol which is **synchronized with a clock**
- Unlike UART, **data is transferred continuously** in SPI
 - There are no packets, start bits, stop bits, parity bits, or anything else
- One device acts as a **controller** to one or more **peripheral** devices
- Supports **full-duplex** communication
 - Bi-directional communication



A Note on Terminology

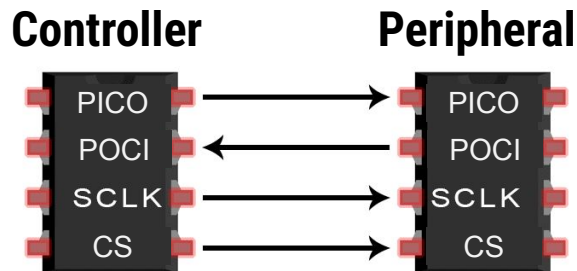
The words **master** and **slave** pervade modern software and electronics documentation. The terms are inappropriate, so we are transitioning to new terminology in OPS documentation.

There will be a discrepancy between what is discussed in lecture/lab and what is written in our parts' datasheets.

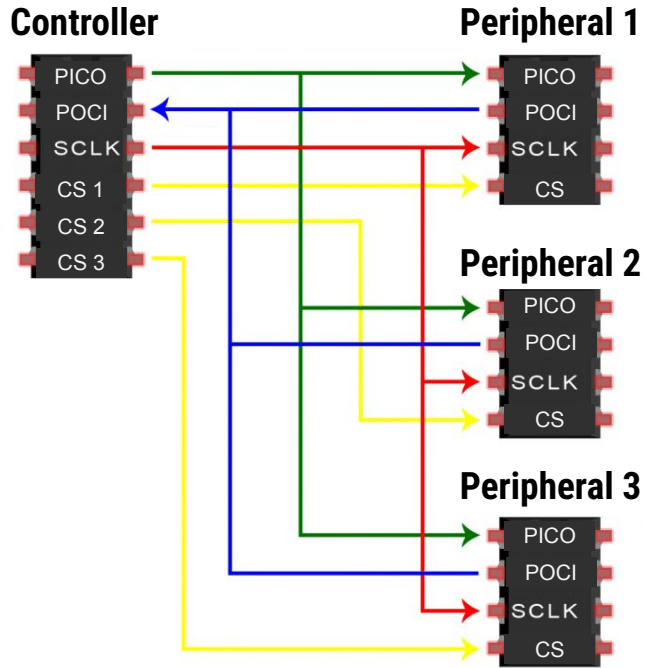
Old Term	New Term
Master	Controller
Slave	Peripheral
MISO	POCI
MOSI	PICO
SS	CS

SPI Layout

- **PICO** (**P**eripheral-**I**n/**C**ontroller-**O**ut) - line for the controller to send data to the peripheral
- **POCI** (**P**eripheral-**O**ut/**C**ontroller-**I**n) - line for the peripheral to send data to the controller
- **SCLK** (**C**lock) - line for the clock signal
- **CS** (**C**hip **S**elect) - line for the controller to select which peripheral to send data

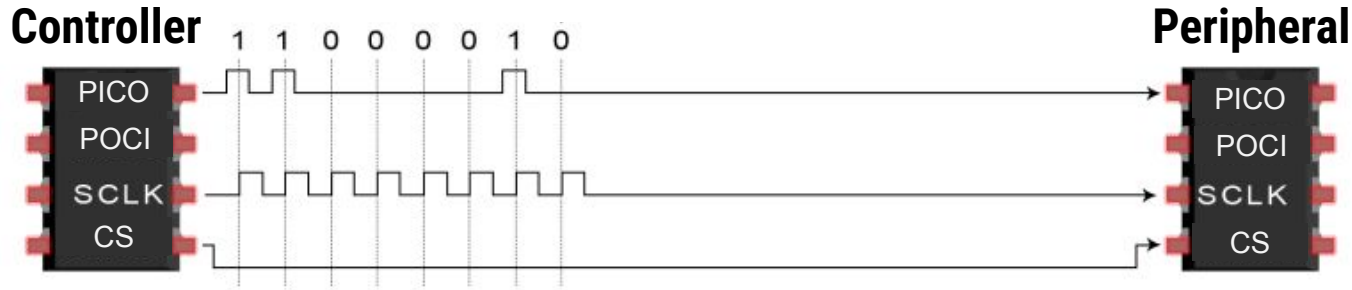


SPI Layout (Cont'd)



- There is **one controller**
 - Often a microcontroller (ESP32)
 - Generates the clock signal
- The controller controls **one or more peripherals**
 - i.e. Radio module, sensor, actuator
 - In the configuration to the left, there is an **additional Chip Select (CS) line for each peripheral**

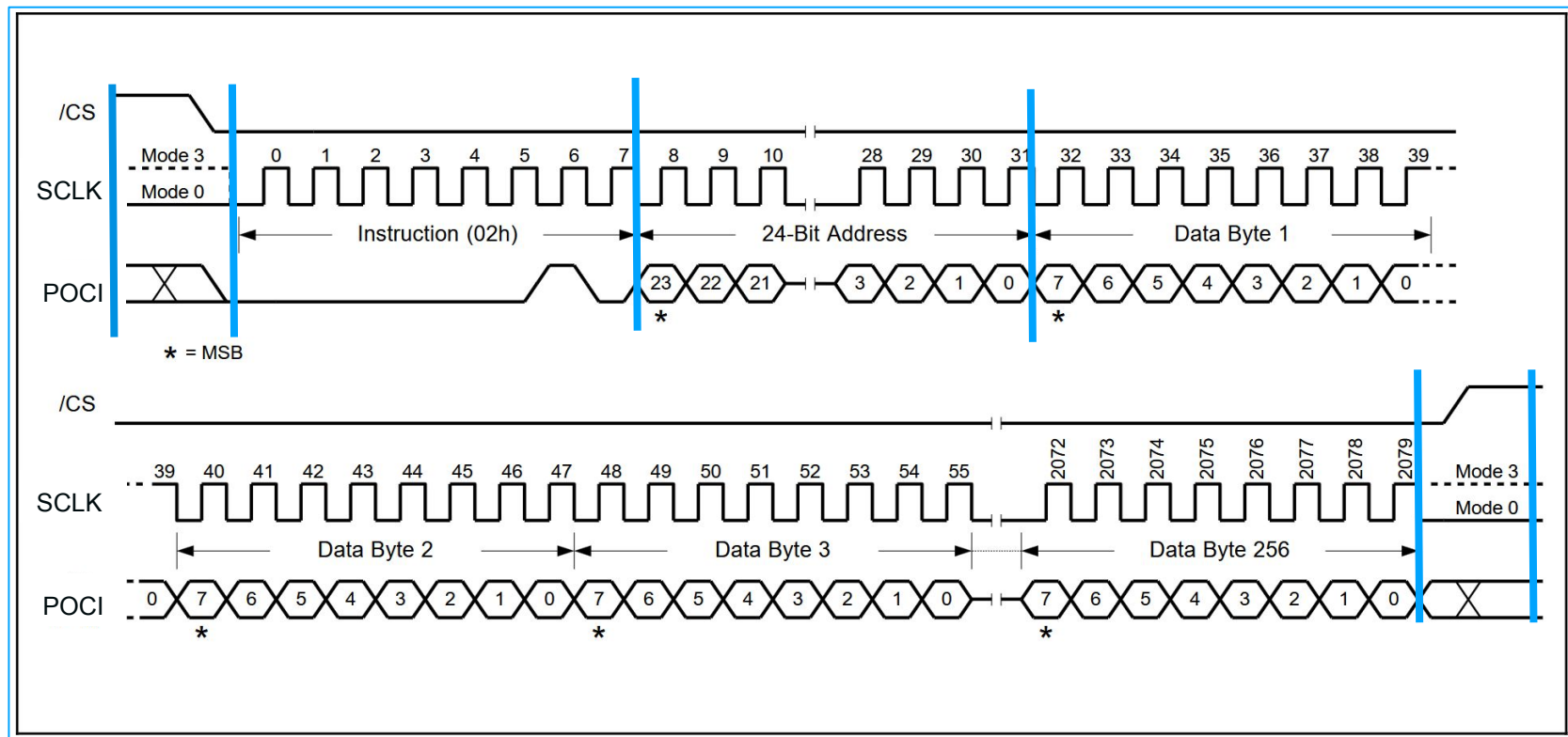
SPI Transmission



1. The controller first **selects a peripheral** by pulling a HIGH signal to LOW along the Chip Select (CS) line
2. The controller **sends data** to the selected peripheral along the Peripheral-In/Controller-Out (PICO) line
 - Bits in the PICO signal are **synchronized with the rising edge** of the oscillating clock signal
 - Alternatively, the peripheral sends data along the Peripheral-Out/Controller-In (POCI) line

SPI Transmission Example

Example is of writing data into a flash memory. (WinBond W25q1128jv)

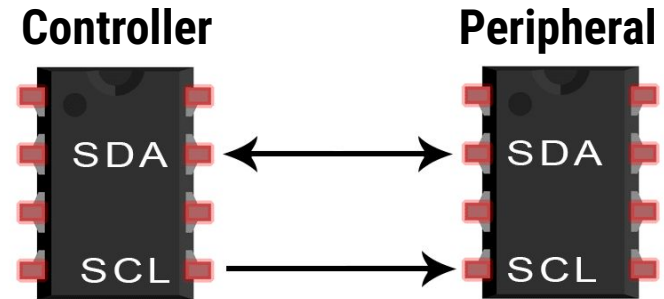


SECTION V

I²C

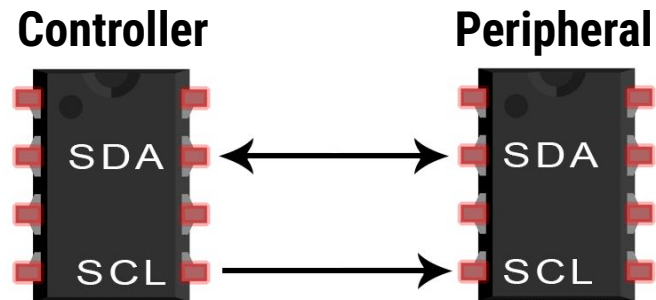
I²C Protocol

- **Inter Integrated Circuit** or I²C is another synchronized, serial protocol
- Unlike UART and SPI, I²C supports **multiple controllers**
- I²C features **acknowledgements** to confirm if messages are received and addressing for peripherals
- Supports **half-duplex** communication
 - One way communication



I²C Layout

- **SDA (Serial Data)** - the line for the controller and peripheral to send and receive data
- **SCL (Serial Clock)** - the line that carries the clock signal
- All peripherals can share the same two SDA and SCL lines



I²C Frame Format

- Data is transferred as **messages**
- A **start condition** from the controller begins the message
 - Pulls **Serial Data** (SDA) line to LOW before **Serial Clock** (SCL) switches HIGH to LOW
- Each message contains an **address frame** with an address to identify the receiving peripheral

I²C Message



I²C Frame Format (Cont'd)

- The **read/write bit** indicates whether the controller is sending or requesting data
 - LOW for controller sending, HIGH for controller receiving
- Then, the peripheral with the matching address sends an **acknowledgement (ACK)** of the message as a single bit
 - Pulled the line from HIGH to LOW

I²C Message



I²C Frame Format (Cont'd)

- Depending on whether the message is read or write, either the controller or peripheral will send **data frames**
- The device receiving data will send an **ACK** before the transmission of the next data frame

I²C Message



I²C Frame Format (Cont'd)

- The devices may transfer as many data frames in one message as desired
- If the controller is reading from the peripheral, it will send a **NACK** to end data transmission
- The controller terminates communication by signalling the **stop condition**
 - Pulls **Serial Data** (SDA) line to HIGH after **Serial Clock** (SCL) switches LOW to HIGH
 - Another controller may send a message upon seeing the stop signal

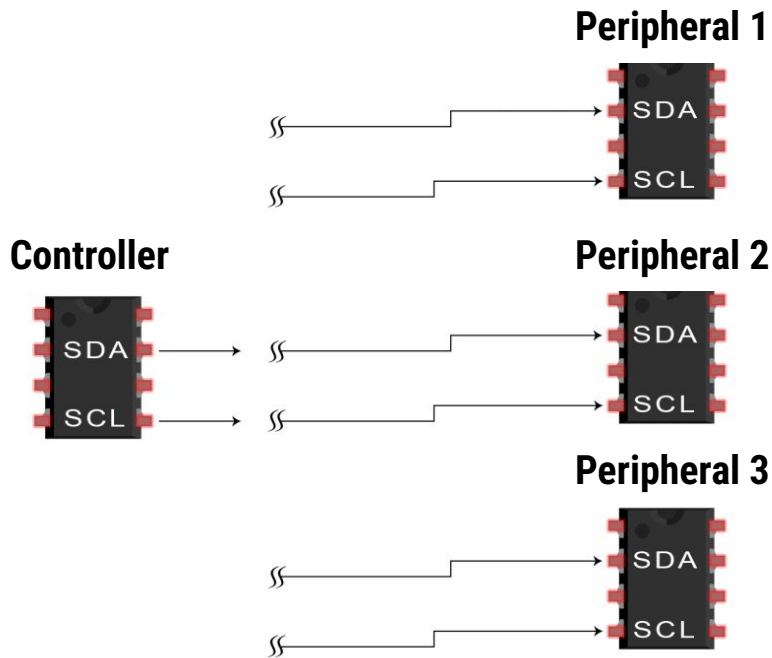
I²C Message



I²C Transmission (Cont'd)

1. The controller outputs the **start condition** to all connected peripherals

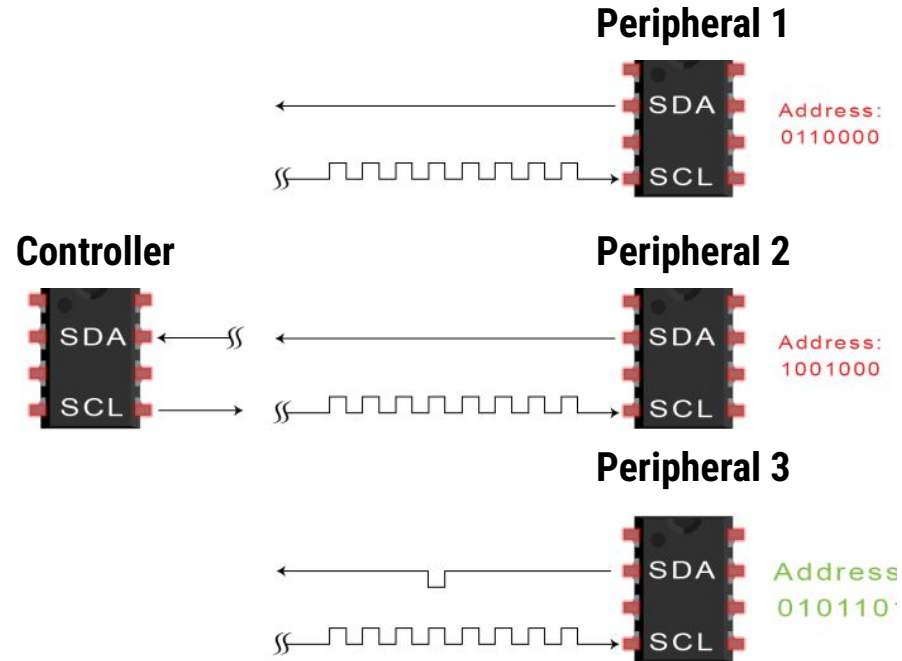
Starting from a **HIGH** voltage in idle, the controller pulls the voltage down **LOW** on the SDA line *then* the SCL line



I²C Transmission (Cont'd)

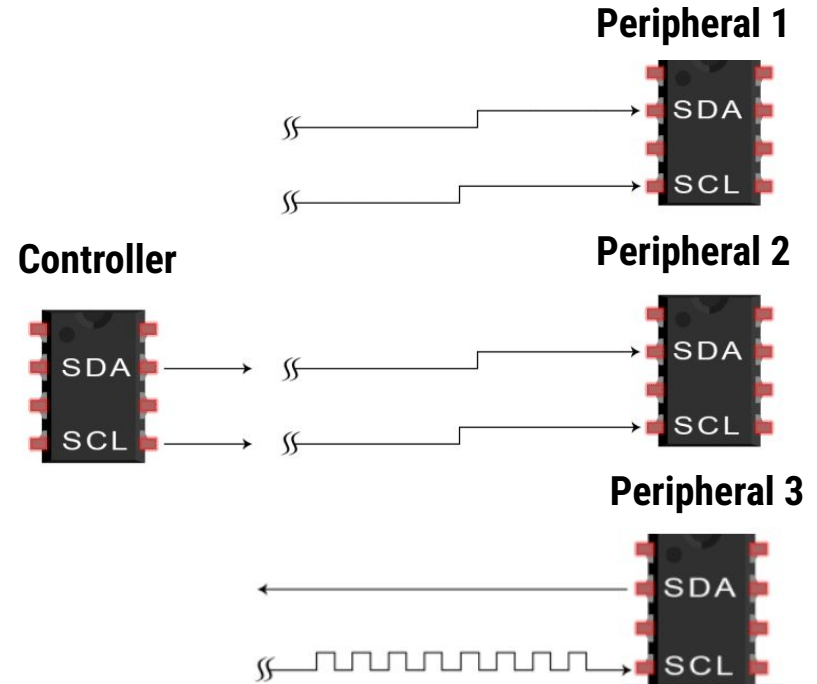
2. The controller then **sends the address** of the peripheral it would like to communicate with

The **last bit of the address is the read/write bit** that indicates whether the controller is sending or receiving data



I²C Transmission (Cont'd)

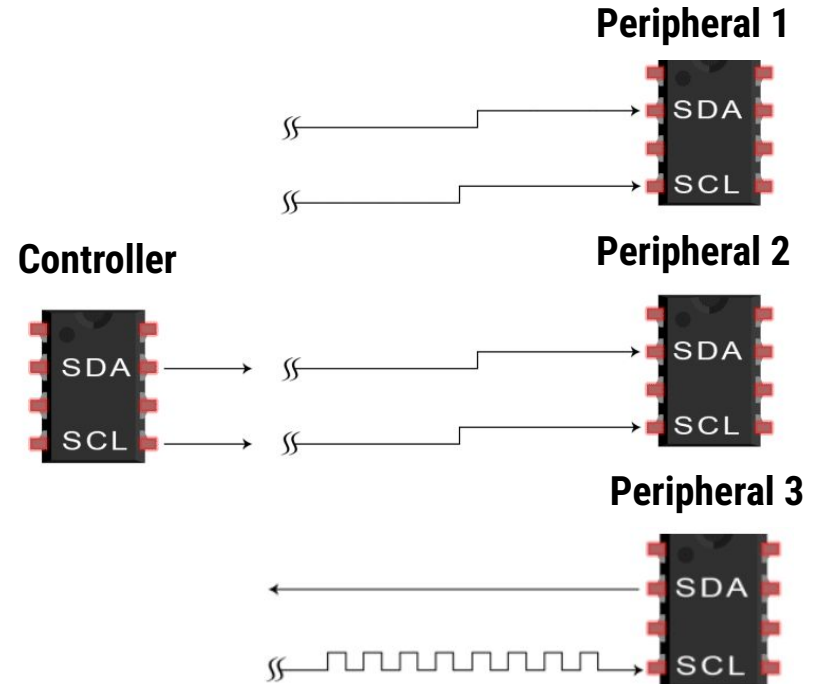
3. The peripheral with the matching address responds with an **ACK**
- **Note:** The ACK signal would go to every peripheral on the I2C line, but communication will now be exclusively from the controller to peripheral 3 in the example.



I²C Transmission (Cont'd)

4. The controller **sends or receives the data frame**, depending on the read/write bit

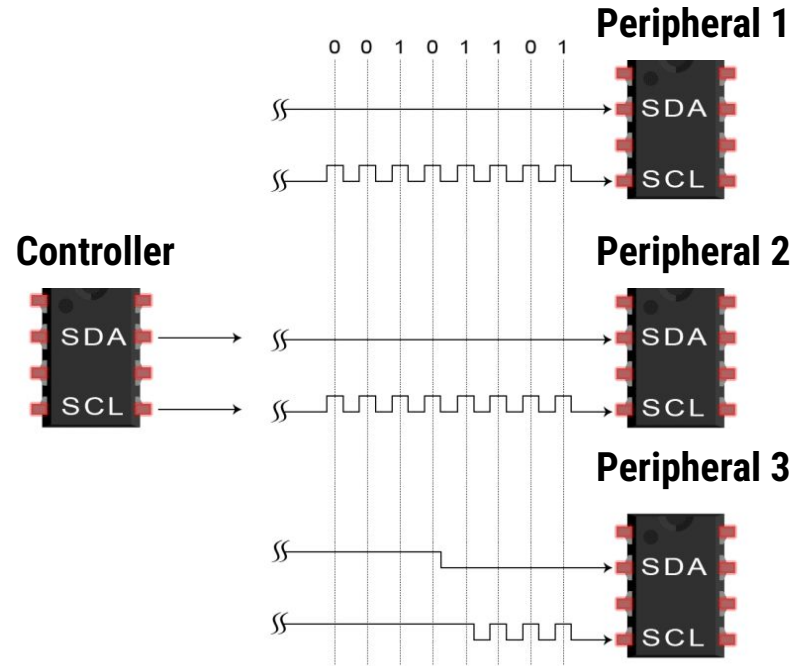
After each transmission, the receiving device sends an ACK for a successful transmission



I²C Transmission (Cont'd)

4. Finally, the controller switches the voltage of the SCL line then the SDA line to **HIGH**, signalling the **stop condition**




The message is complete



SECTION VI

Comparing Protocols

UART vs SPI vs I²C

Protocol	UART	SPI	I ² C
# of Lines	2	4+	2
# of Controllers	1	1	1+
# of Peripherals	1	1+	1+
Transmission Type	Full-Duplex	Full-Duplex	Half-Duplex
Error-Checking			
Speed	Slowest	Fastest	Slower

Which one of the following is a full-duplex communication protocol?

- A. SPI
- B. I²C
- C. Both SPI and I²C
- D. Neither SPI or I²C

Which of the following protocols doesn't require an additional line to support multiple devices on the same bus?

- A. SPI
- B. I²C
- C. Both SPI and I²C
- D. Neither SPI or I²C

Which of the following protocols supports multiple controllers and peripherals?

- A. SPI
- B. I²C
- C. UART
- D. Both UART and I²C

Which of the following protocols doesn't have start and stop bits?

- A. SPI
- B. I²C
- C. UART
- D. Both UART and I²C

Comparing Protocols

The SCLK, PICO, POCI, CS are the four data lines in _____ protocol.

- A. SPI
- B. I²C
- C. UART
- D. Both SPI and I²C

Which of the following communication protocols is a type of synchronous protocol?

- A. SPI
- B. I²C
- C. UART
- D. Both SPI and I²C

Which one of the following protocols needs a clock?

- A. SPI
- B. I²C
- C. UART
- D. Both SPI and I²C

The receiver and transmitter are the two data lines in _____ protocol.

- A. SPI
- B. I²C
- C. UART
- D. Both SPI and I²C

Which of the following protocols needs a chip select line?

- A. SPI
- B. I²C
- C. UART
- D. Both SPI and I²C

Which of the following protocols is a single controller, single peripheral communication protocol?

- A. SPI
- B. I²C
- C. UART
- D. Both SPI and I²C

How many signal lines does I²C Protocol require?

- A. 1
- B. 2
- C. 4

How many signal lines does UART Protocol require?

- A. 1
- B. 2
- C. 4

Which of the following protocols is best for multiple peripherals over the fewest lines?

- A. SPI
- B. I²C
- C. UART

Which of the following protocols is best for multiple peripherals with the highest transmission rate?

- A. SPI
- B. I²C
- C. UART

Which of the following protocols is best for full-duplex communication with multiple peripherals?

- A. SPI
- B. I²C
- C. UART

Which of the following protocols consumes less power?

A. SPI

B. I²C

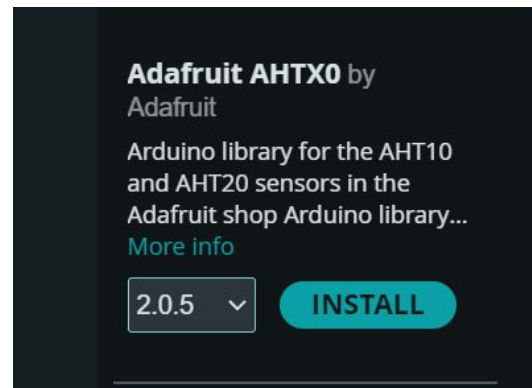
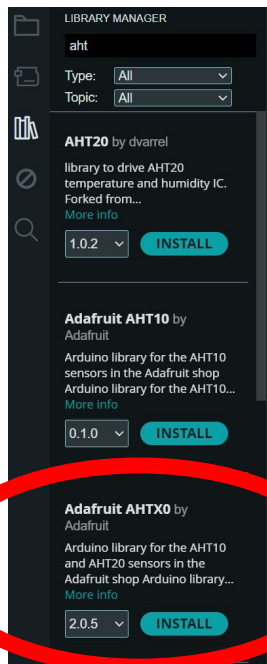
SECTION VII

AHT20 Exercise

AHTX0 Library Installation

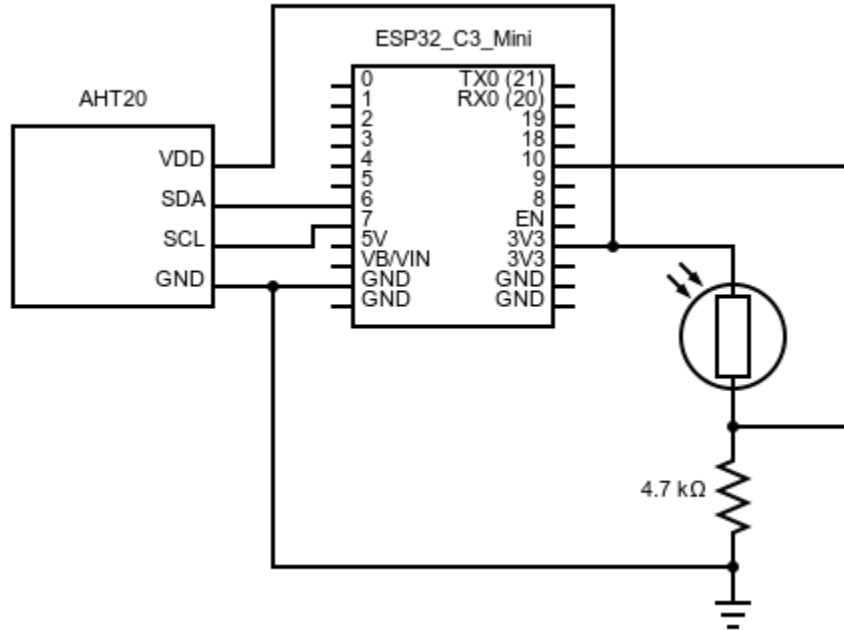
- Find and install the Adafruit AHTX0 library by Adafruit
- You can include this library with

```
#include <Adafruit_AHTX0.h>
```



AHT20 Exercise Schematic

Build this circuit here:



Adafruit AHTX0 by
Adafruit

Arduino library for the AHT10
and AHT20 sensors in the
Adafruit shop Arduino library...
[More Info](#)

2.0.5 ▾

INSTALL

Wire Library

- Built-in library that allows us to set our own SDA and SCL pins for I²C
- Include this library with `#include <Wire.h>`
- `Wire.begin(int I2C_SDA, int I2C_SCL);`
 - Initializes the Wire library and allows the ESP32 to participate in I²C as a controller or peripheral.
 - Takes two pin numbers and configures them to be SDA and SCL pins, respectively.
 - Need to connect AHT20's SDA and SCL to the ESP32 pins configured with this function

```
// Define I2C pins
#define I2C_SDA 6
#define I2C_SCL 7

void setup() {
    // configure I2C for pins 6 and 7
    Wire.begin(I2C_SDA, I2C_SCL);
}
```

AHT20 Functions

- `aht.begin()` finds the AHT sensor via the ESP32's SDA and SCL pins
- `aht.getEvent(sensors_event_t *humidity, sensors_event_t *temperature)`
 - Stores the AHT's humidity and temperature readings into two `sensors_event_t` objects (named humidity and temperature in this example)

```
void setup() {  
    Serial.begin(115200);  
  
    // configure I2C for pins 6 and 7  
    Wire.begin(I2C_SDA, I2C_SCL);  
  
    aht.begin();  
}
```

```
void loop() {  
    sensors_event_t humidity, temp;  
    aht.getEvent(&humidity, &temp);  
  
    delay(500);  
}
```

FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.

CC BY-NC-SA 4.0

This work by the Institute of Electrical and Electronics Engineers, UC Irvine Branch, is licensed under CC BY-NC-SA 4.0