LECTURE V

Communication Protocols I

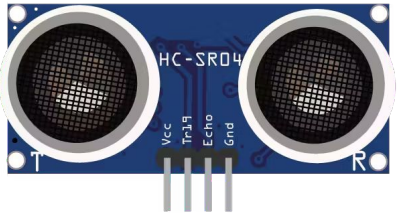
This work by the Institute of Electrical and Electronics Engineers, UC Irvine Branch, is licensed under CC BY-NC-SA 4.0

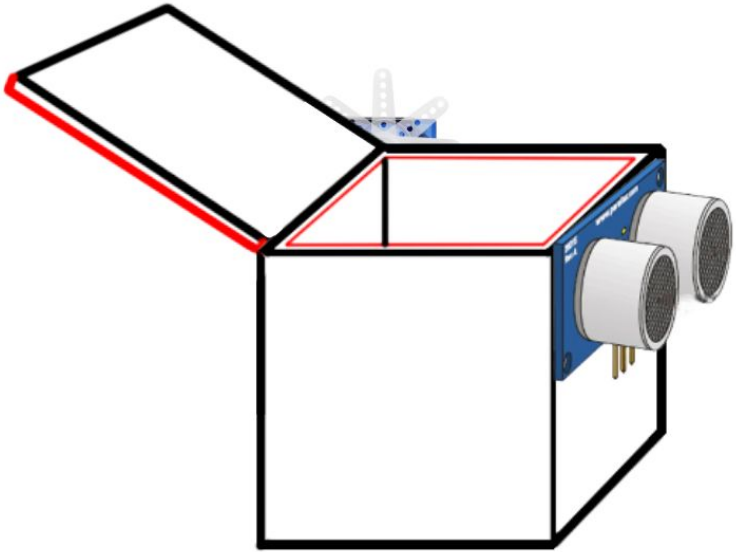
SECTION I

Project IV Review

Project 4 Review

- Build a mini trash can that uses an **ultrasonic sensor, micro servo,** and **ESP32**





You learned:

- ESP32 (Continued)
- Servos
- Ultrasonic Sensor
- Arduino Libraries
- Tips for Programming in Arduino IDE

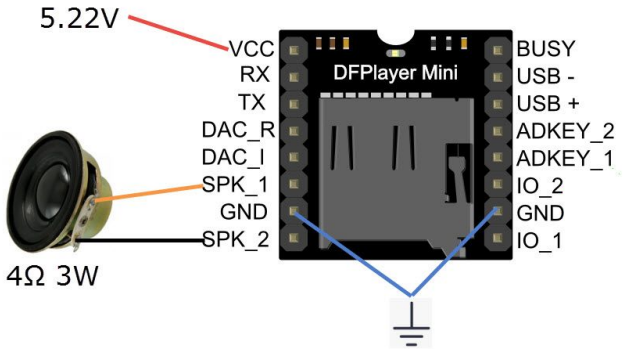
Fall Quarter Feedback Form + refunds

- If you haven’t already please fill out the fall quarter feedback form, We appreciate your feedback!

- Refunds will be processed by during the first few weeks of the quarter

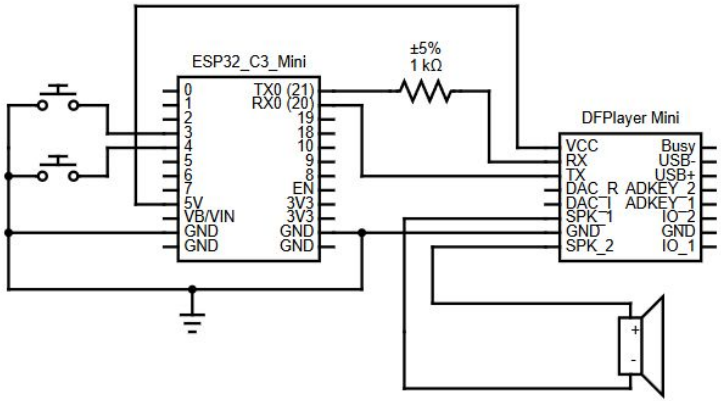
Project 5

- Build and program a **music player** with ESP32, customized to play **your own tunes**.
- Due date: 1/26/2026 at 11:59PM



Learning Concepts:

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



Real World Uses of Microcontrollers

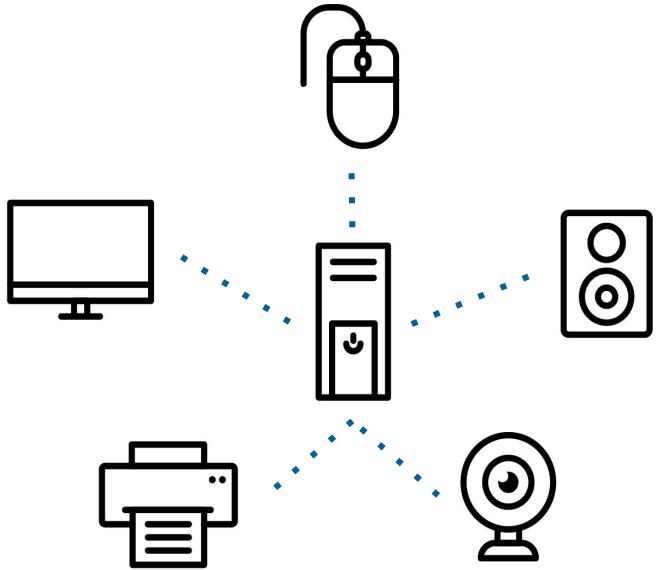
This is how i build a Flight Computer! - YouTube

SECTION I

What are Communication Protocols?

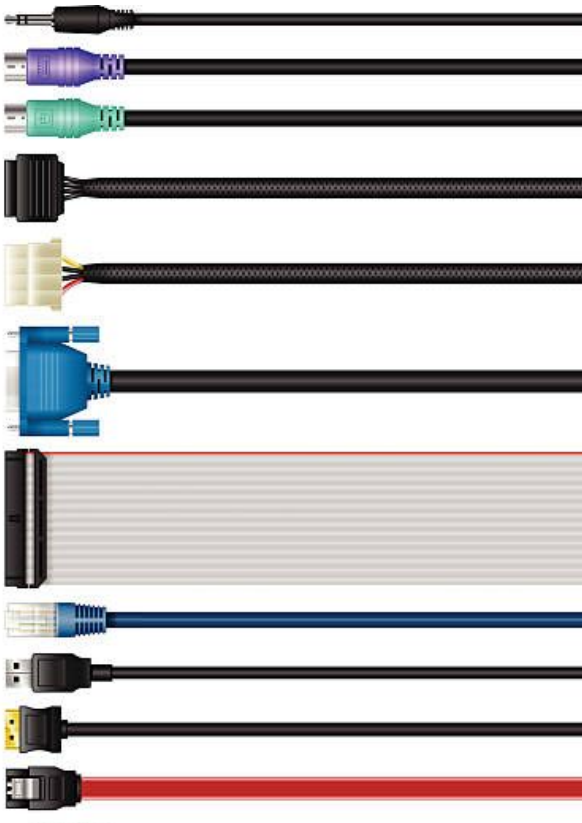
Communication

- Devices, like your home computer, **communicate** with other devices
    - i.e. mouse, monitor, printer, webcam, speakers, ESP32
- Industry focusing on this is called **Internet of Things (IoT)**
    - Involves connecting to a network of physical devices and exchanging information.
- We will see this in project 6

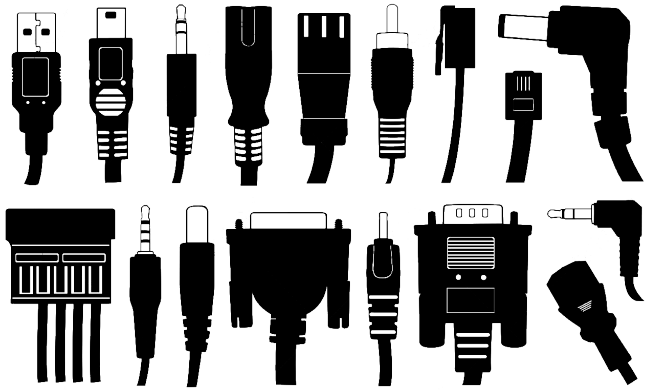


Communication Media

- Devices communicate over different **media**
  - i.e. copper wire, radio, fiber optic wire, etc.
  - Media can be **wired** or **wireless**
- Devices communicate by **transmitting** and **receiving** electrical signals over a shared medium (physical or wireless)
  - Signals may be **digital** (1s and 0s) or **analog** (i.e. AM/FM radio)



Communication Protocols



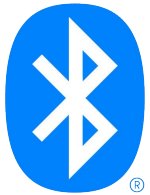
- With so many possibilities for communication between devices, it makes sense that in industry, it has been standardized to make it simpler
- Like humans, communicating devices **must share a common "language"** to understand messages transmitted between them
- The rules for how we send, receive, and interpret these signals are defined by **protocols**
- You already know of some protocols…

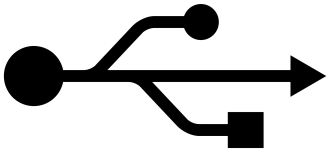
Communication Protocols (Cont’d)

Can you name an example of a communication protocol?

Communication Protocols (Cont'd)

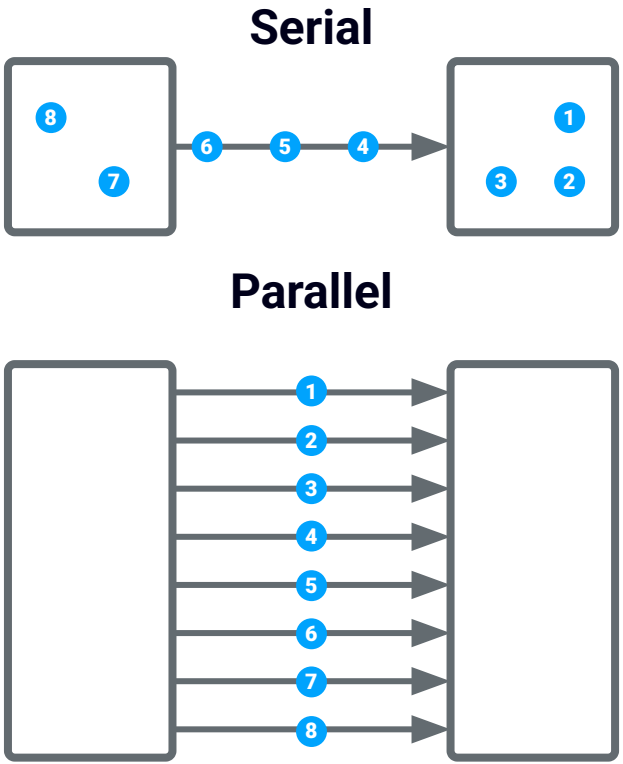
- Your smartphone communicates with your wireless earbuds over a **Bluetooth** protocol
- All the laptops in the lecture hall connect to a wireless router using a **WiFi** protocol (a.k.a **IEEE 802.11x**)
- Your wired mouse connects to your computer using a **Universal Serial Bus** (**USB**) protocol





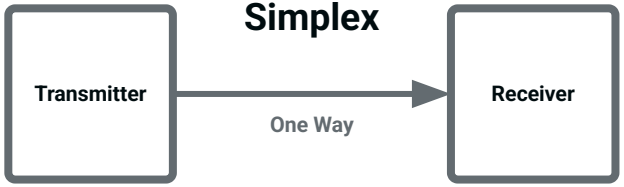
Serial vs Parallel Protocols

- **Serial** - bits are sent over a connection one by one to a device
  - Bits are sent in a specific order
  - The most common protocols are serial
- **Parallel** - multiple bits (often one byte) are sent simultaneously over a connection
  - This connection requires more wires, which takes up more space
  - Higher transmission rate/throughput
  - Can be prone to crosstalk and timing mismatches over longer distances!

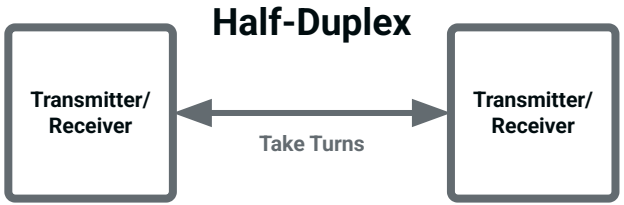


Transmission Modes

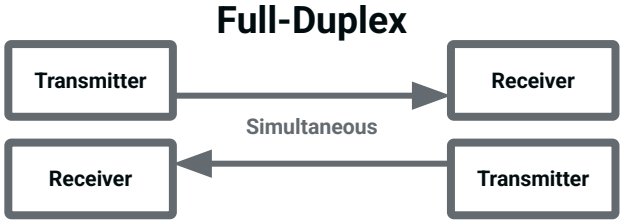
- **Simplex** protocols allow communication in only one direction



- **Half-Duplex** protocols allow communication in both directions but only in one direction at a time

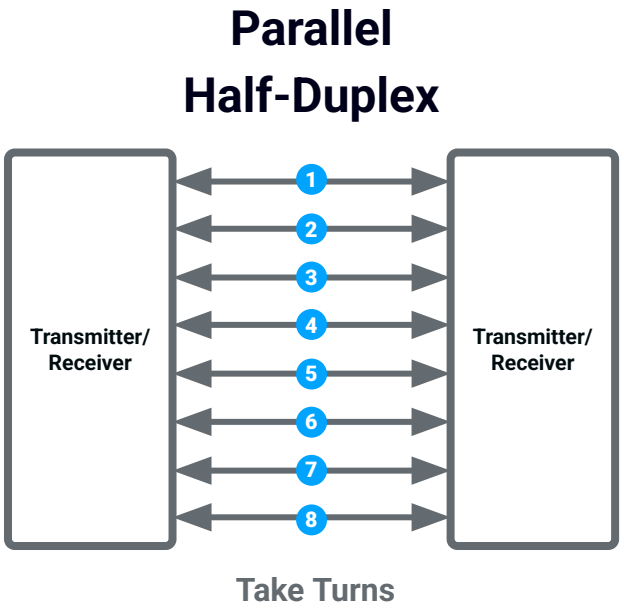


- **Full-Duplex** protocols allow communication in both directions simultaneously



Protocols vs. Transmission Modes

- It's important to note that **transmission modes** and **protocols** are **different**
- When two devices are exchanging data, it can be simplex **and** parallel, or **full duplex** and serial; there are many possible combinations
- Each **transmission mode** and **protocol** have their own benefits and drawbacks that we won't get into this lecture



SECTION II

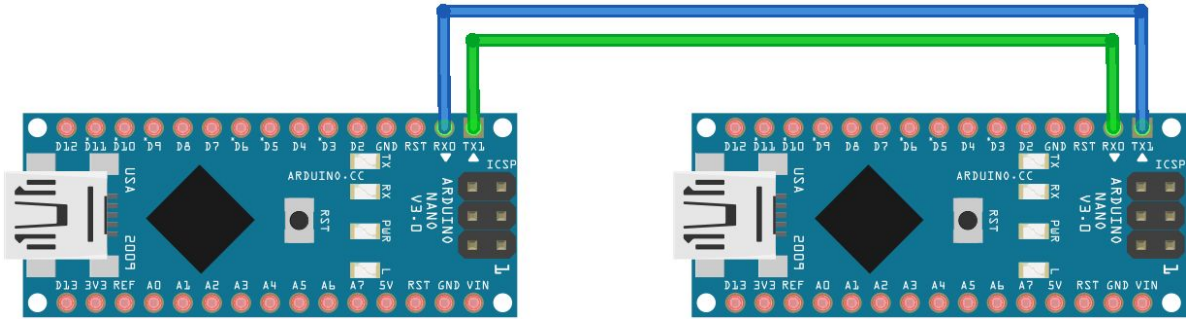
UART

UART Protocol

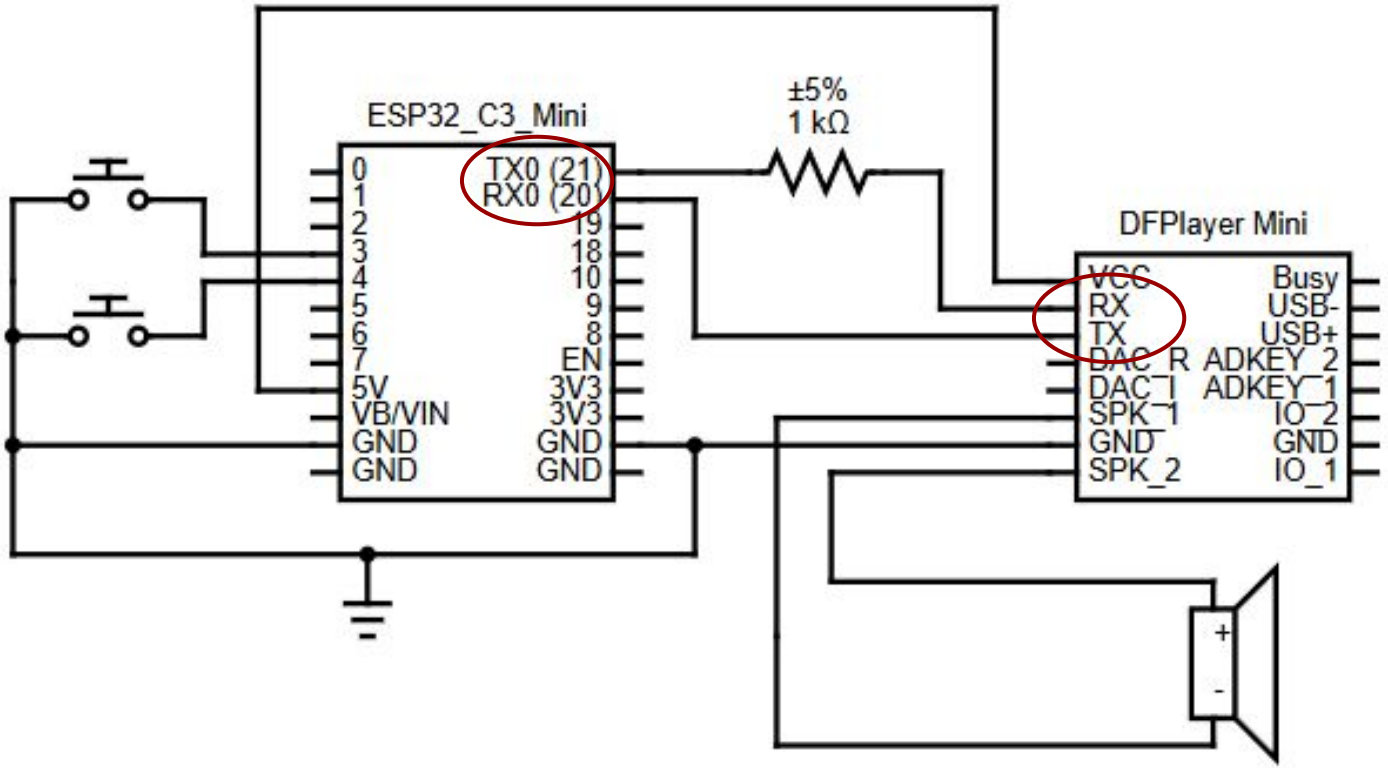
- Most Commonly used communication protocol
- Universal Asynchronous Receiver-Transmitter (UART) is a **communication protocol with certain characteristics**. It has rules for data transmission.
  - UART can be configured to **full-duplex, half-duplex,** and **simplex modes**
  - There are only **two lines** (electrical connections)
  - There is **no clock signal**/line hence the “asynchronous” part of the title
  - The data transmission speed is determined by a **baud rate**
    - It is quite **slow compared to other protocols**

UART Layout

- Two UARTs may be connected as pictured
  - The **transmitter** (**TX**) pin of one UART is connected to the **receiver** (**RX**) pin of the other
  - The are two data lines, which **enables full-duplex communication**
- Data is sent from the transmitter of one UART to the receiver of the other UART



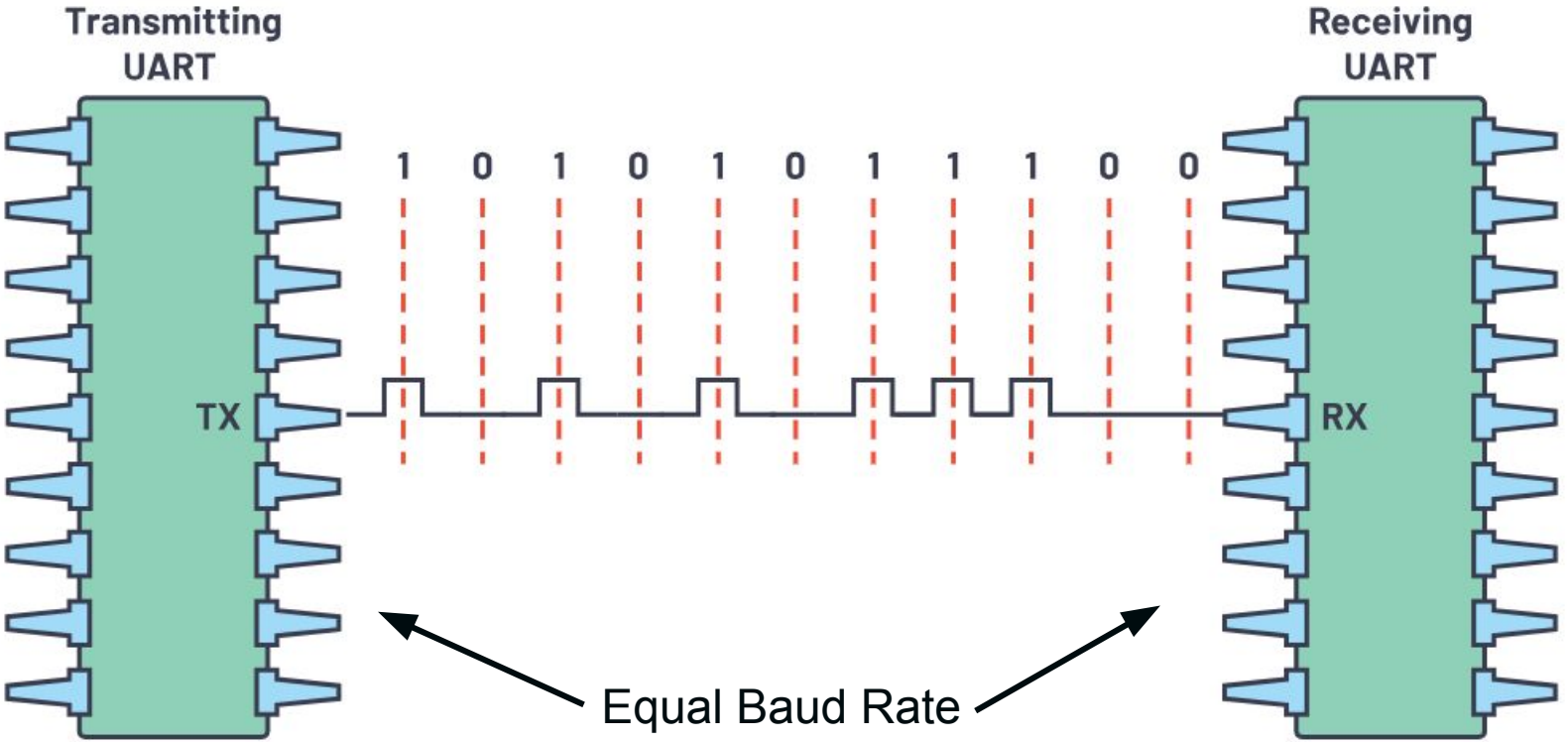
UART in Project 5



UART Baud Rate

- A UART sends and receives transmissions at the speed of the **baud rate**
  - It is configured by the programmer on each UART
  - Measured in **bits per second**
    - When you set `Serial.begin(115200,` that's 115200 bits per second!
- The baud rate **must be equal** on the two communicating UARTs to successfully send and receive the packets
  - Otherwise, messages are read at the wrong speed and become garbled

UART Baud Rate (cont'd)



UART Frame Format

- Data is transmitted as segments of bits called **packets**
  - This data is sent one bit at a time (a.k.a **serially**) between UARTs
- Without modification, the UART packet follows this format:

UART Packet

UART Frame Format (Cont'd)

- The **start bit** indicates when the packet begins
  - The data transmission line goes from default HIGH voltage to a LOW
- The **data frame**, containing the data, follows the start bit
  - If the parity bit is used, it can be 5 to 8 bits. If the parity bit is not used, it can be 9 bits.

UART Packet

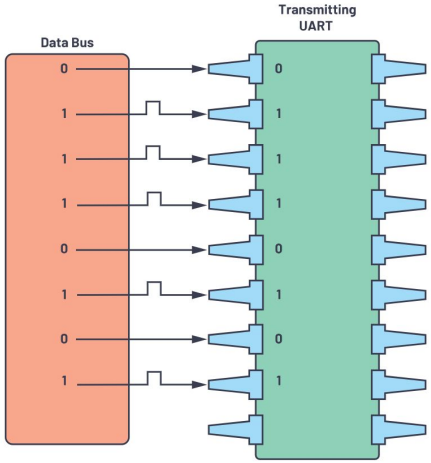
UART Frame Format (Cont'd)

- The **parity bit** is an **optional** bit comes after the data frame
  - used by the receiving UART to check if the data is free of errors
- The **stop bit** indicates the end of a packet
  - The data transmission line goes from LOW voltage back to default HIGH

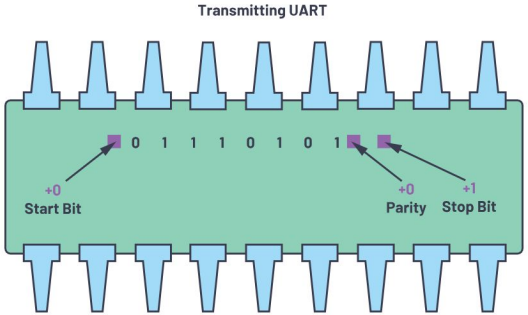
UART Packet

UART Communication steps

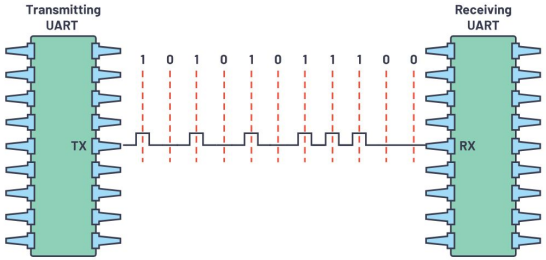
1.



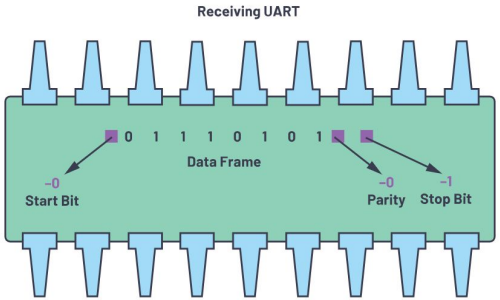
2.



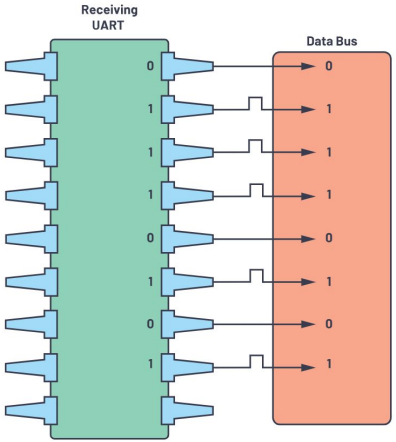
3.



4.



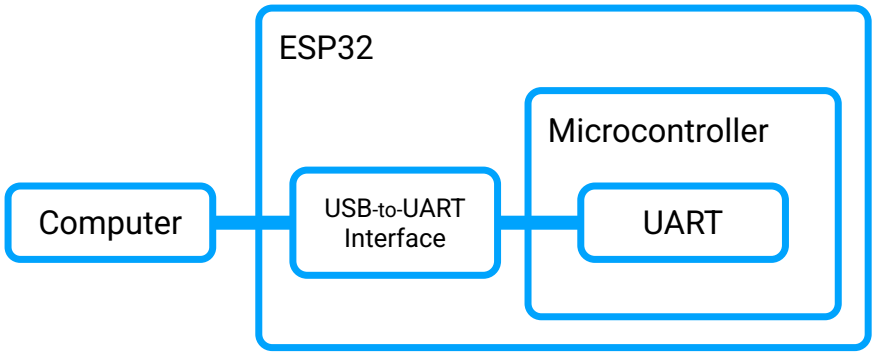
5.



USB on the ESP32

- Data is transmitted between the computer and ESP32 through a chip that converts USB signals to UART signals
    - Those converted signals are transmitted to and from the UART

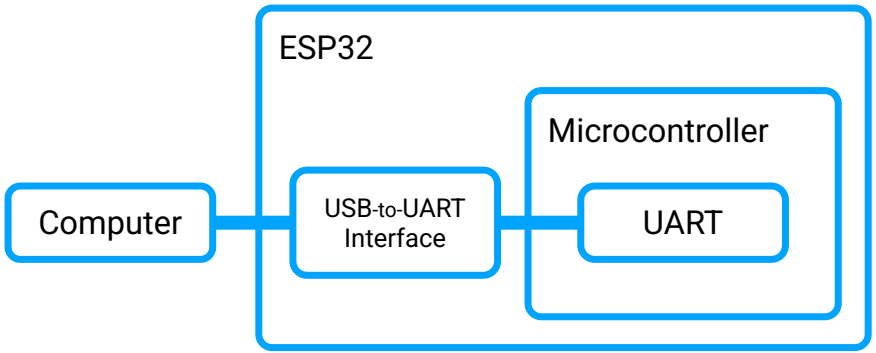
- The ESP32's USB port is hardwired to the TX and RX pins of the board
- **Do not use the UART while simultaneously using USB**
    - This will cause data corruption due to the ESP32 device receiving data from both the computer and its UART paired device



USB on the ESP32

- Data is transmitted between the computer and ESP32 through a chip that converts USB signals to UART signals
    - Those converted signals are transmitted to and from the UART

- The ESP32's USB port is hardwired to the TX and RX pins of the board
- **Do not use the UART while simultaneously using USB**
    - This will cause data corruption due to the ESP32 device receiving data from both the computer and its UART paired device



UART Trivia Questions

- What is UART?

A: Circuit Design

B: Communication Protocol

C: Network

D: None of the above

UART Trivia Questions (Cont.)

- What is UART?

A: Circuit Design

B: Communication Protocol

C: Network

D: None of the above

UART Trivia Questions (Cont.)

- UART is _____.

A: Simplex

B: Half-Duplex

C: Full-Duplex

D: All of the above

UART Trivia Questions (Cont.)

- UART is _____.

A: Simplex

B: Half-Duplex

C: Full-Duplex

D: All of the above

UART Trivia Questions (Cont.)

- True or false: The (TX) pin of one UART is connected to the (TX) pin of the other

A: True

B: False

UART Trivia Questions (Cont.)

- True or false: The (TX) pin of one UART is connected to the (TX) pin of the other

A: True

B: False

SECTION III

Arduino Serial Library

Arduino Serial Library

- We can use the UART in ESP32 to transmit data over UART protocol
  - To do so, we will enlist the help of the Serial library, which is already installed in the Arduino IDE
- We will go over various functions of the Serial library in Workshop V
  - `Serial.begin(int baud_rate)`
  - `Serial.print(val/str)`
  - `Serial.read()`
  - `Serial.write(val/str)`
  - `Serial.available()`

SECTION IV

Pull-Up Resistor Circuits

Project 5 Requirements

iPoduino

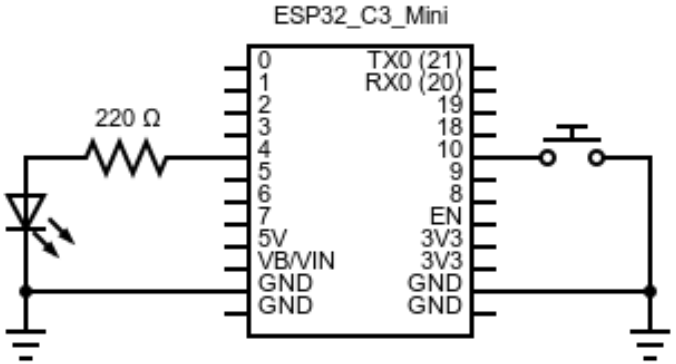
You must build a MP3 Player with speaker sound using the DFPlayerMini module.
The player must have two buttons.
One of the player's buttons must toggle the music between "Play" mode and "Pause" mode. ◦ In "Play" mode, the player must playback a song. It should resume the song that was previously paused if the "Skip/Next Song" button was not pressed. ◦ In "Pause" mode, no song should be played.
The second button should be the "Skip/Next Song" button, which interrupts the current song and plays a new song. ◦ Each press of this button must cycle through all songs loaded on the MicroSD card. ◦ If the player was in "Pause" mode when the "Skip/Next Song" button is pressed, the player must immediately enter "Play" mode.
The speaker must be silent when no song is played. No static or white noise.
The speaker must clearly output music when a song is played.
The circuit must be built on a breadboard.

Project 5 Requirements (Continued)

- We want to use a pushbutton to trigger an event / for a specific purpose

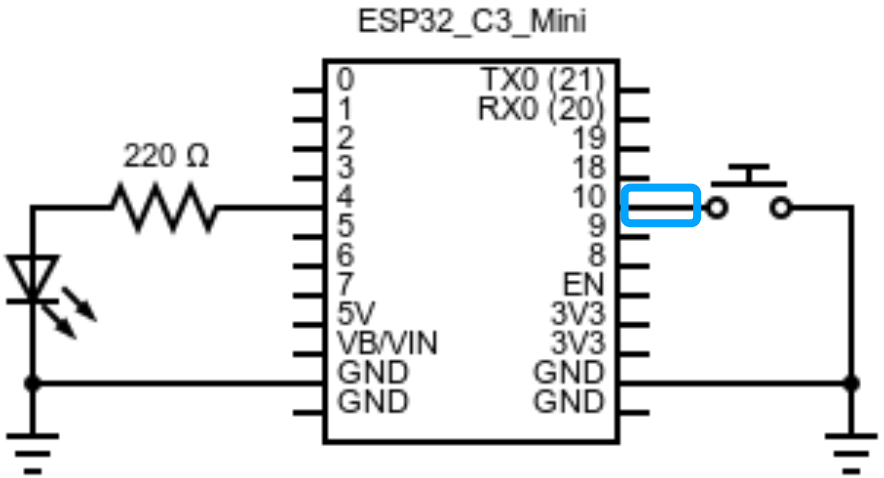
Reading Button Inputs w/ ESP32

- We will use `digitalRead(int pin)` to read the button's inputs.
  - **Reads the voltage** at the input pin, returning `HIGH` (3.3V) or `LOW` (0V) as an integer (1 or 0)
  - When the button is pressed, pin 10 is connected to GND
  - But how about the when the button is not pressed?



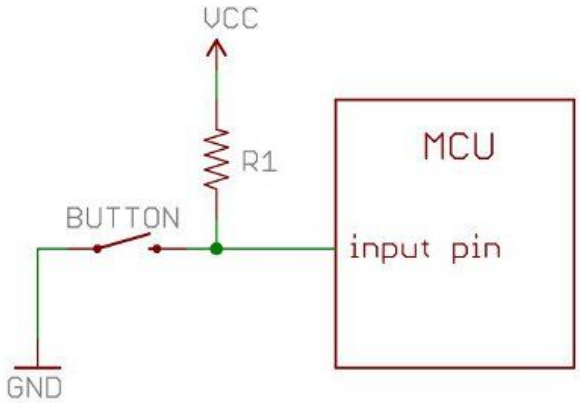
Floating Signal

- When the button is unpressed, we call it a **floating signal**
- Voltage at a signal can either return `HIGH` or `LOW` as an integer (1 or 0)
  - This is due to Electromagnetic interference (EMI) or unwanted noise



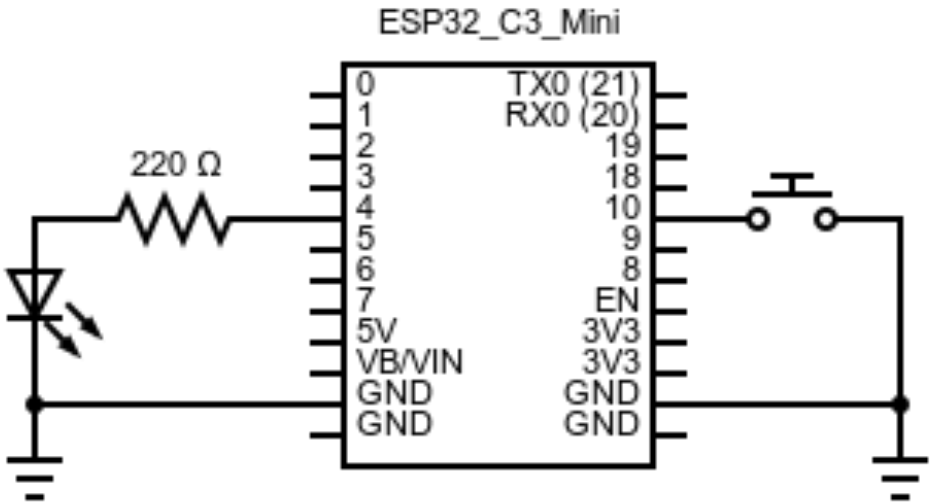
Pull-Up Resistors

- Pull-up resistors are one way we can solve the problem of floating signals!
  - R1 is typically a higher resistor value (e.g. 10k Ohms)
- When the button is NOT being pressed, a tiny amount of current flows through the resistor and pulls the pin voltage up to VCC
- When the button is pressed, it provides a low resistance path to GND and pulls the pin voltage down to GND



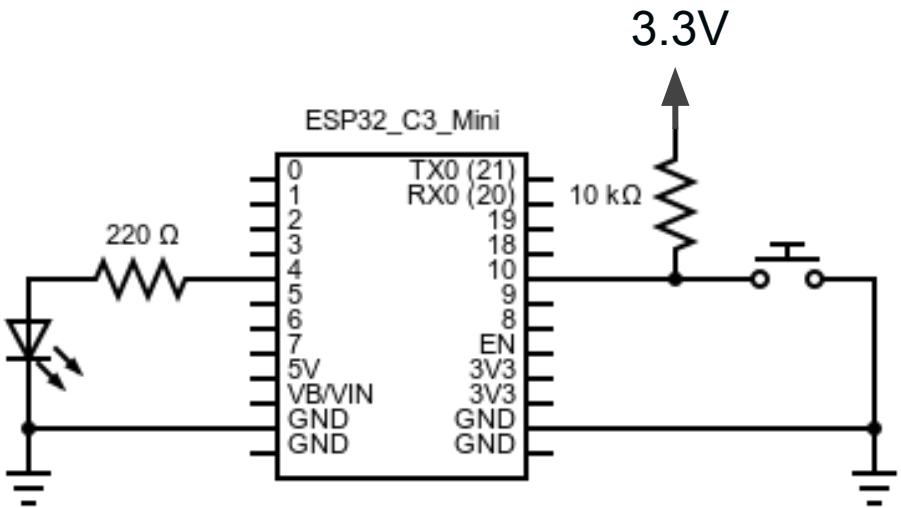
Pull-Up Resistor Circuit Exercise

Let’s build this circuit here:



Pull-Up Resistor Circuit Exercise (Cont.)

- We use a **Pull-Up Resistor** to no longer make it floating signal
- A **floating signal** is caused by noise/interference from neighboring electronic devices, strong current,etc.
- This causes the pin's state to fluctuate frequently and unpredictably
- By shunting the signal to **3.3V**, through a **pull-up resistor**, we ensure that the button's state is **3.3V** by default



IEEE
Student Branch at UC Irvine

iFamilEEE Info Session: Fam Head Speed Dating

LOCATION: TBA!_

TIME: TBA!_

DATE: THURSDAY, JAN 15TH_

JOIN US FOR A QUICK INFO SESSION ON THE PROGRAM'S EVENTS FOR WINTER QUARTER, MEET OUR FAM HEADS IN A SPEED DATING SOCIAL, GET THE CHANCE TO WIN IN A QUICK JEOPARDY GAME, & FINALLY PICK YOUR PREFERRED FAM AND FAM HEAD!!_

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