UCI University of California, Irvine

Ch. 5 - Serial Communication Protocols
UART, SPI and I2C

COMPSCI 147

Internet-of-Things; Software and Systems

Communication protocols

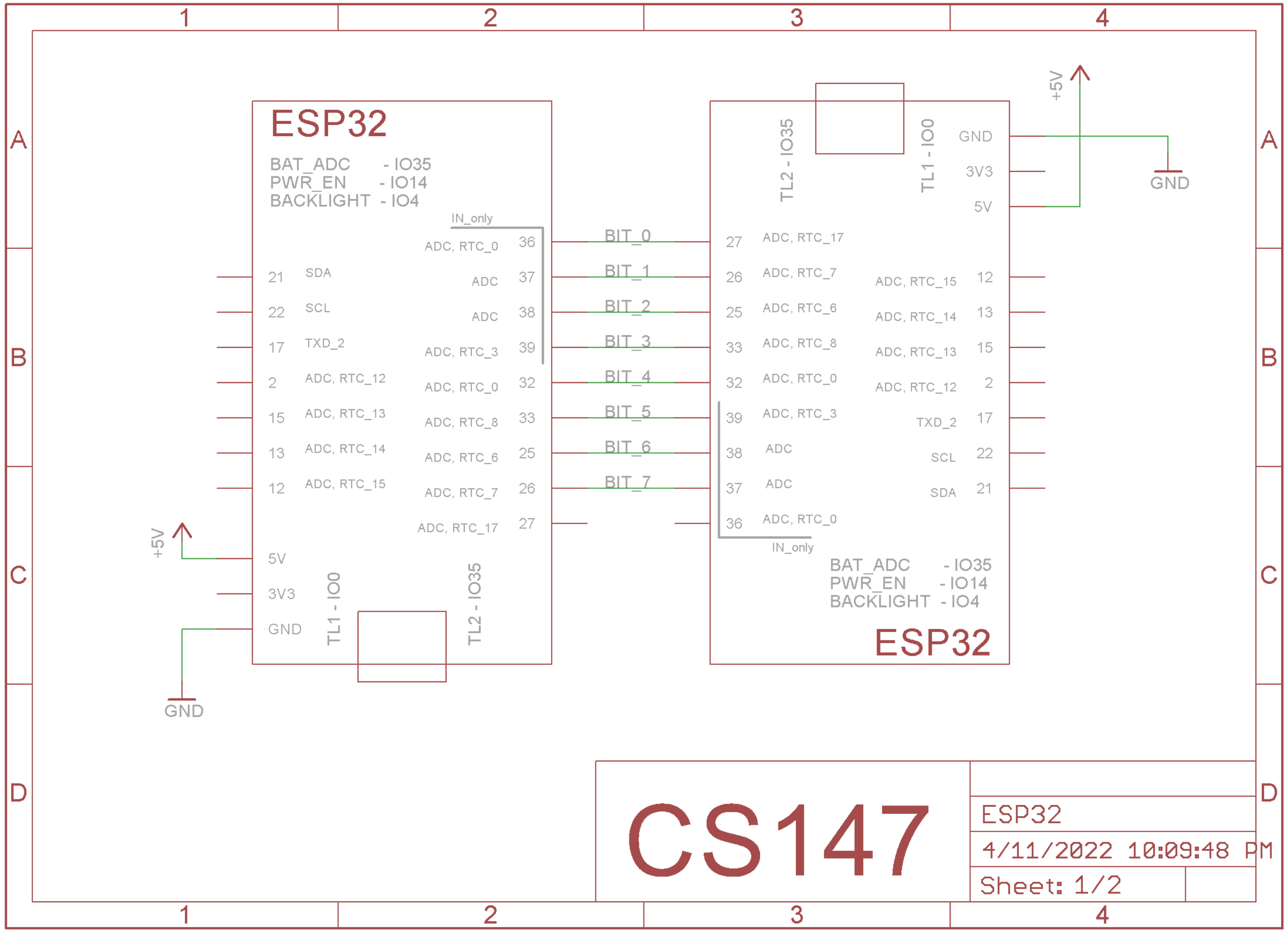
- A set of rules that allow two or more entities of a communication system to transmit information via physical medium.

- Syntax, semantics, and synchronization of communication and possible error recovery methods between communication systems are all defined by the term “protocol”.

- Today’s focus is on wired communication protocols

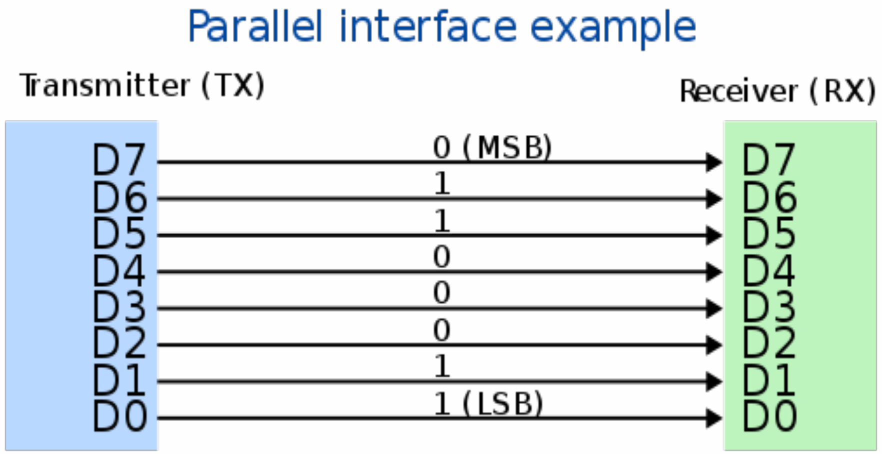
Goal

- Transfer 1 byte of information from 1 ESP32 to another ESP32



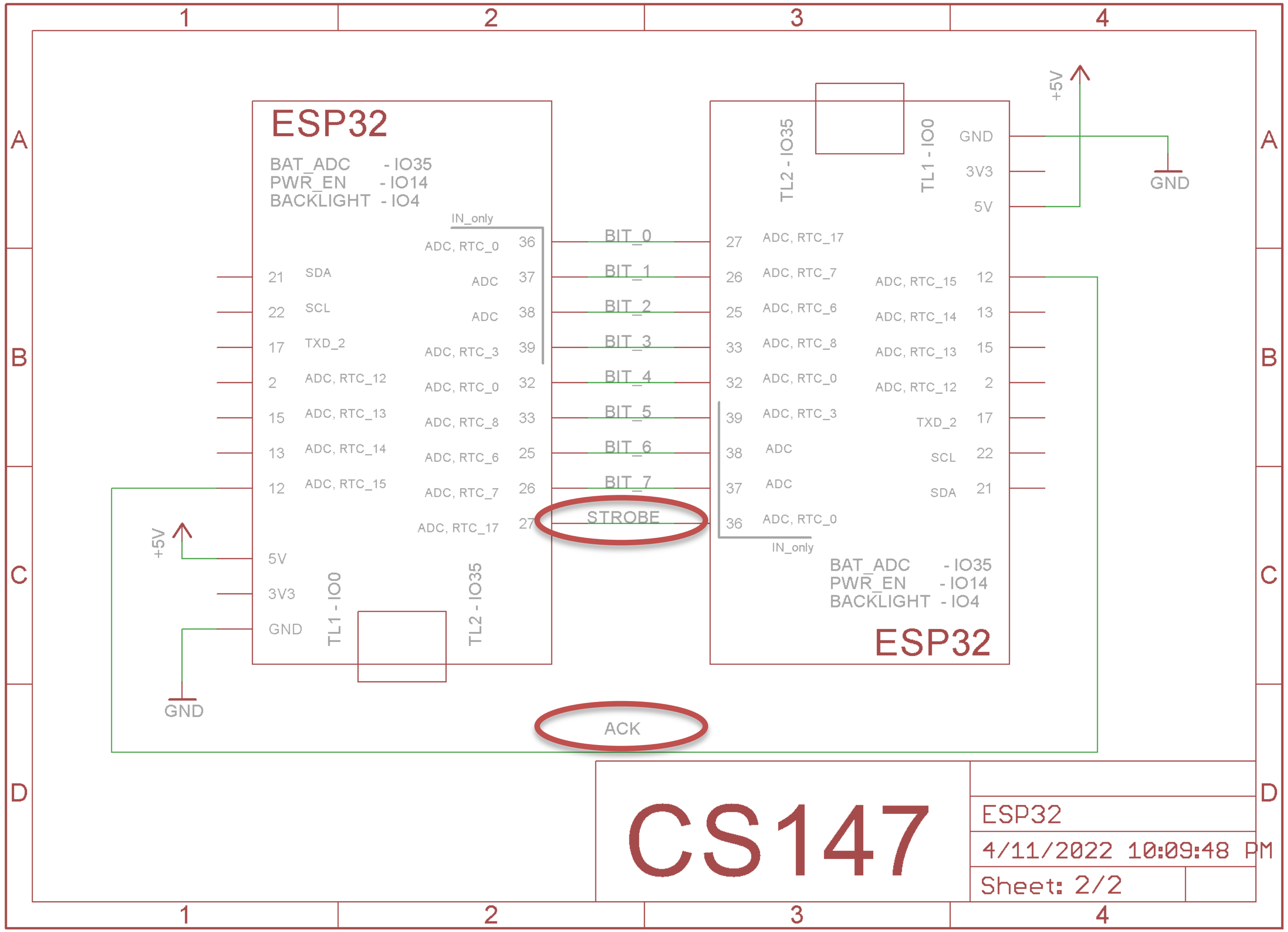
PARALLEL COMMUNICATION

- Method of conveying multiple binary digits (bits) simultaneously.
- Example: memory bus, system bus
- Peripheral Component Interconnect (PCI) bus
- Legacy Printer Port



Goal

- Transfer multiple bytes of information from 1 ESP32 to another ESP32



- Impossible to keep using more pins (H/W limitation)
- Idea is re-use existing pins, but adding control flow mechanisms

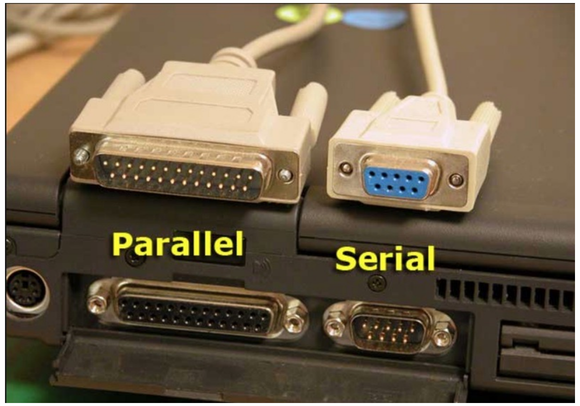
PARALLEL VS SERIAL COMMUNICATION

- Transfer multiple bytes of information from 1 ESP32 to another ESP32
- If we can re-use pins, why not just use 1 pin for data!
  -> Serial communication

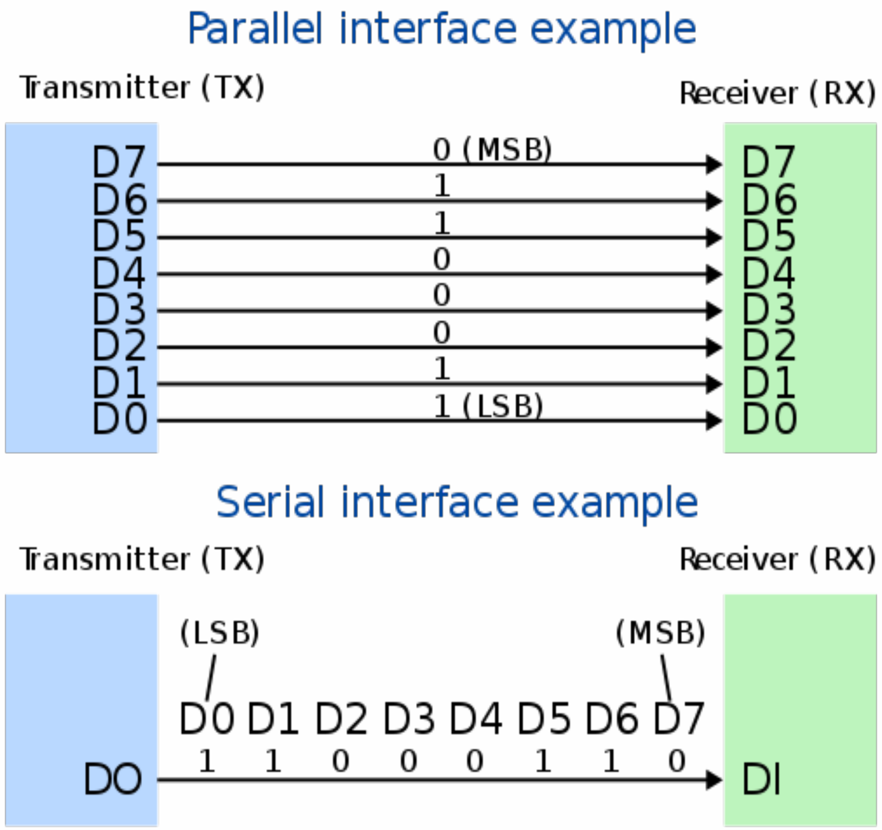
- Impossible to keep using more pins (H/W limitation)
- Idea is re-use existing pins, but adding control flow mechanisms

PARALLEL VS SERIAL COMMUNICATION

- Long story short: [A queue of bits (channel)] VS [several queues].



Example: Serial VS 8-bit parallel channel

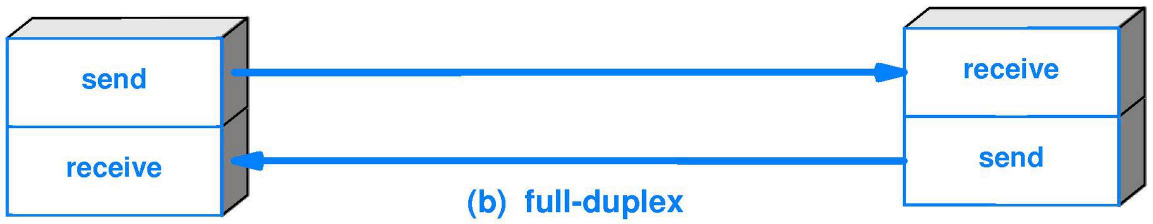


- Common Serial Protocols in MCUs:
  - **UART**: Universal Asynchronous Receiver/Transmitter
  - **SPI**: Serial Peripheral Interface
  - **I2C**: Inter-Integrated Circuit

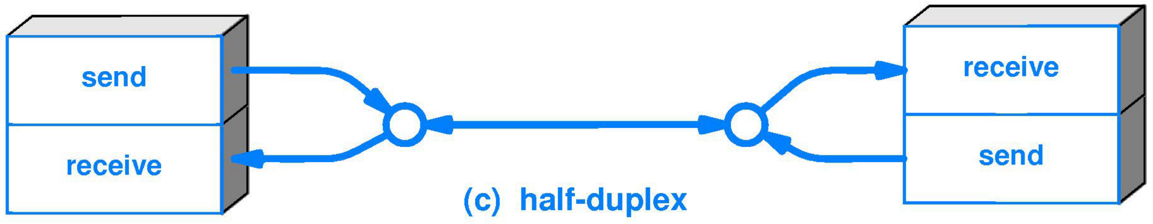
**MSB/LSB: Most/Least Significant bit.*

Communication can be simplex, half-duplex or full-duplex

One side sends
Other side receives



Both sides can send/receive



Both sides can send/receive
but only 1 at a time

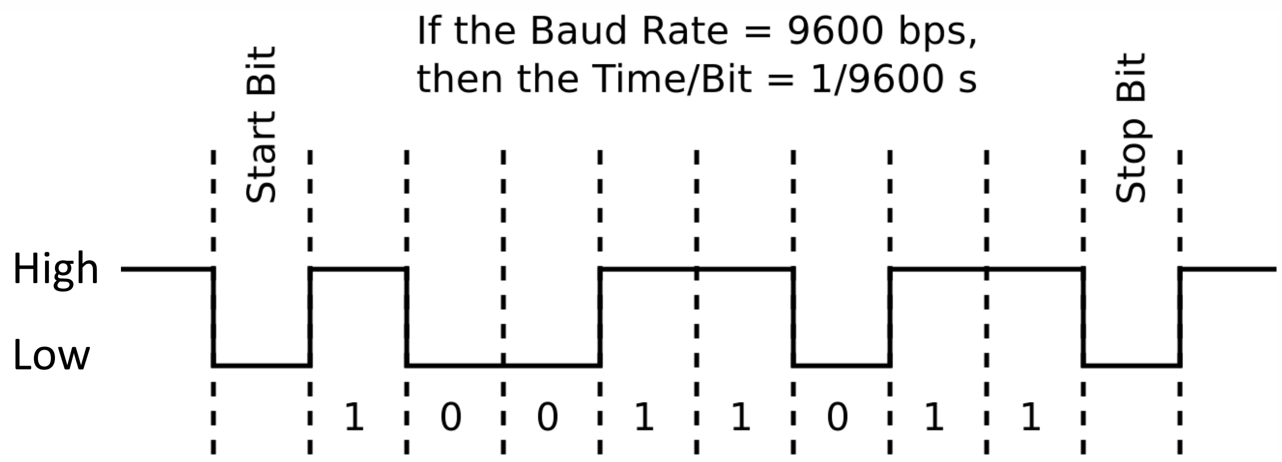
Universal Asynchronous Receiver/Transmitter (UART)

- UART is a character-oriented data link
- Achieve communication between two devices
- Not necessary to add clocking information to the data being sent

- A typical UART frame begins with a START bit, followed by a “character” and an optional parity bit for error detection, and it ends with a STOP condition.

UART – DATA FRAME FORMAT

- Universal **Asynchronous** Receiver/Transmitter (UART)
  - A UART must **create the data packet** and send that packet out the **TX** line **with precise timing** (according to the set baud rate).
  - On the **receive end**, the UART must **sample** the **RX** line at rates according to the **expected baud rate**, pick out the sync bits, and spit out the data.
  - An idle state is high (5 V / 3.3V)



SERIAL COMMUNICATION UART – DATA TRANSMISSION

- It's not a communication protocol, but a physical circuit in a microcontroller.

- Adjustable parameters:
  - Baud rate - Communication speed
  - Data length - Number of actual data bits in one byt
  - Stop bit - Length of the stop after each sent byte
  - Parity bit - Used for error checking

- You can define protocols by fixing the parameters: (e.g., DMX)

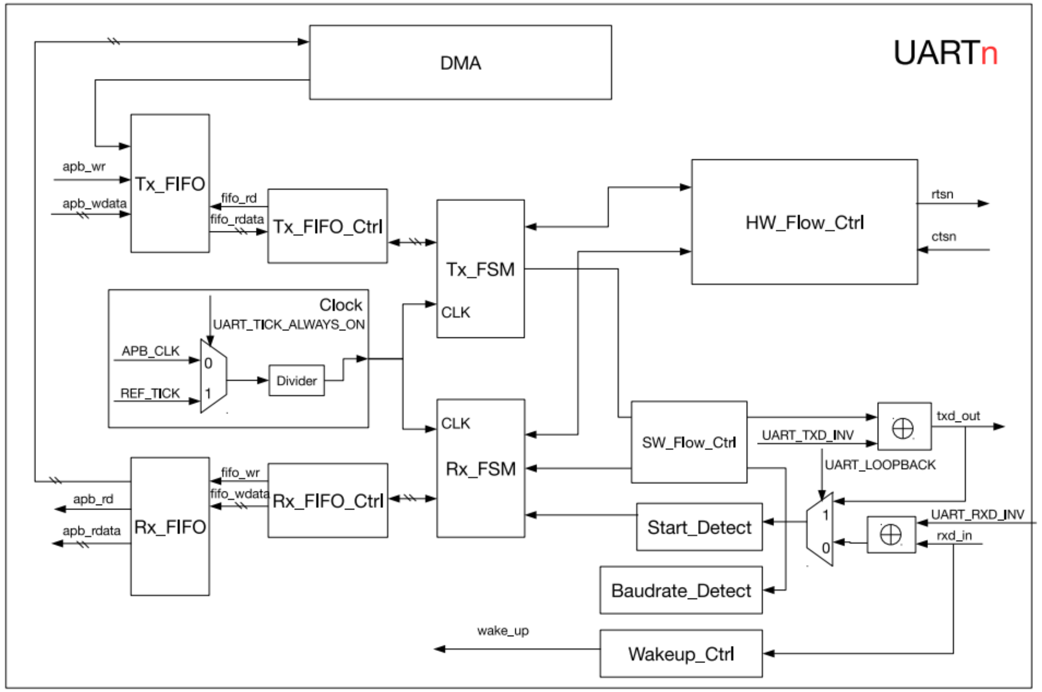
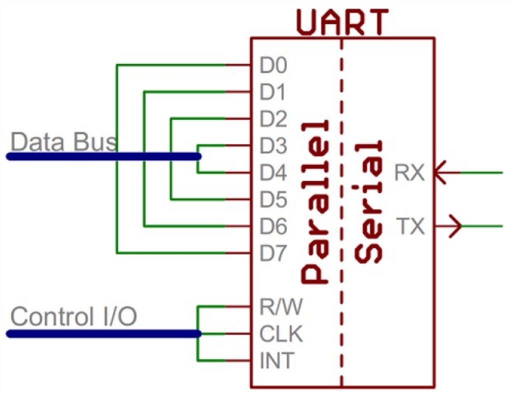


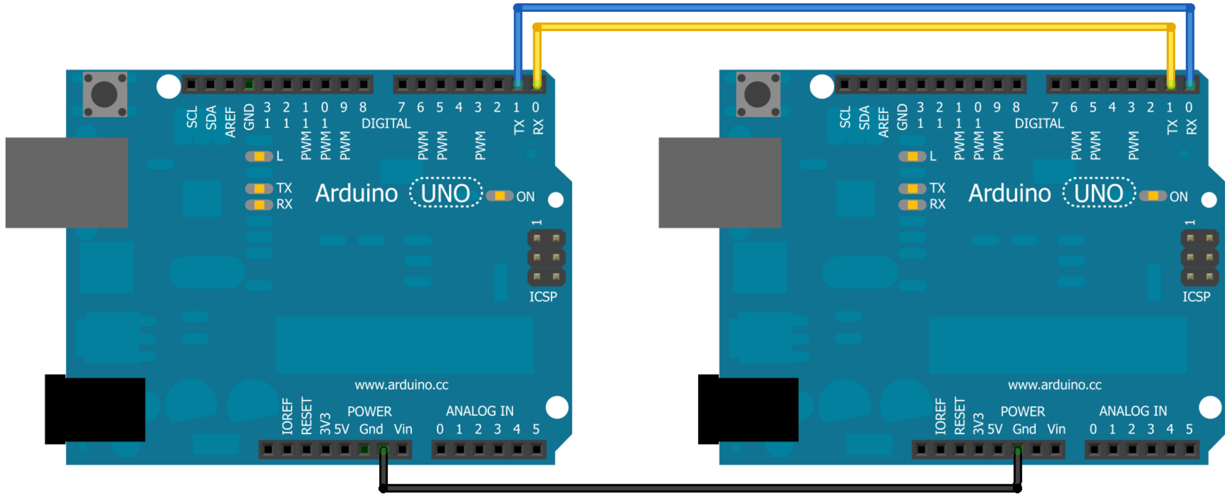
Figure 82: UART Basic Structure

- Additional explanation/examples by Rohde Schwarz
  – https://youtu.be/sTHckUyxwp8

SERIAL COMMUNICATION UART WITH ARDUINO

- Point-to-point communication between only two devices
- Requires two wires minimum (RX and TX)
  - Also recommended to connect the ground wires of the devices together
- Arduino provides two hardware serial communication methods
  - Serial communication over USB
  - Serial over RX and TX pins (0 & 1)





SERIAL COMMUNICATION UART WITH ARDUINO FRAMEWORK

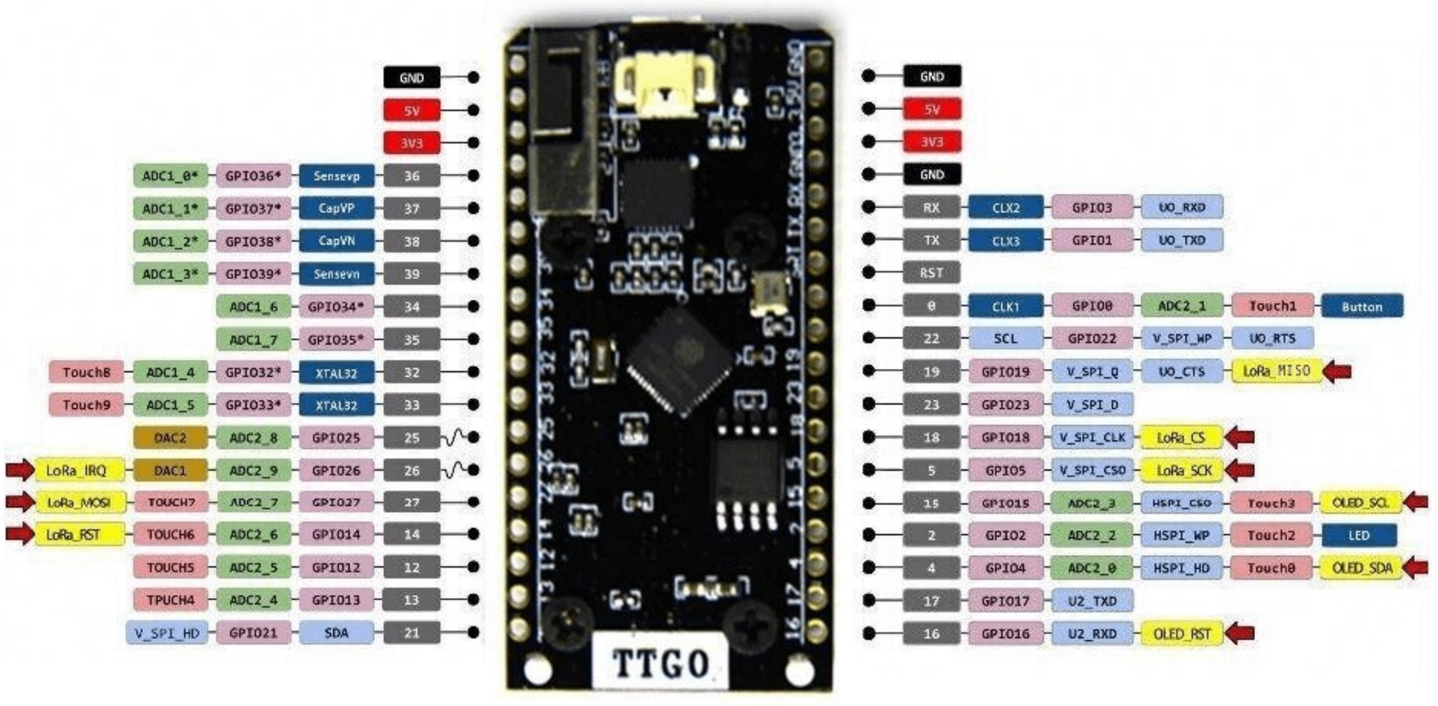
`Serial.begin();`	Setting the baud rate.
`Serial.available()`	Returns the number of bytes available in the serial input buffer.
`Serial.write()`	Write a single byte into the serial port
`Serial.read()`	Read a single byte from the input buffer

MORE INFO:

- http://arduino.cc/en/reference/serial

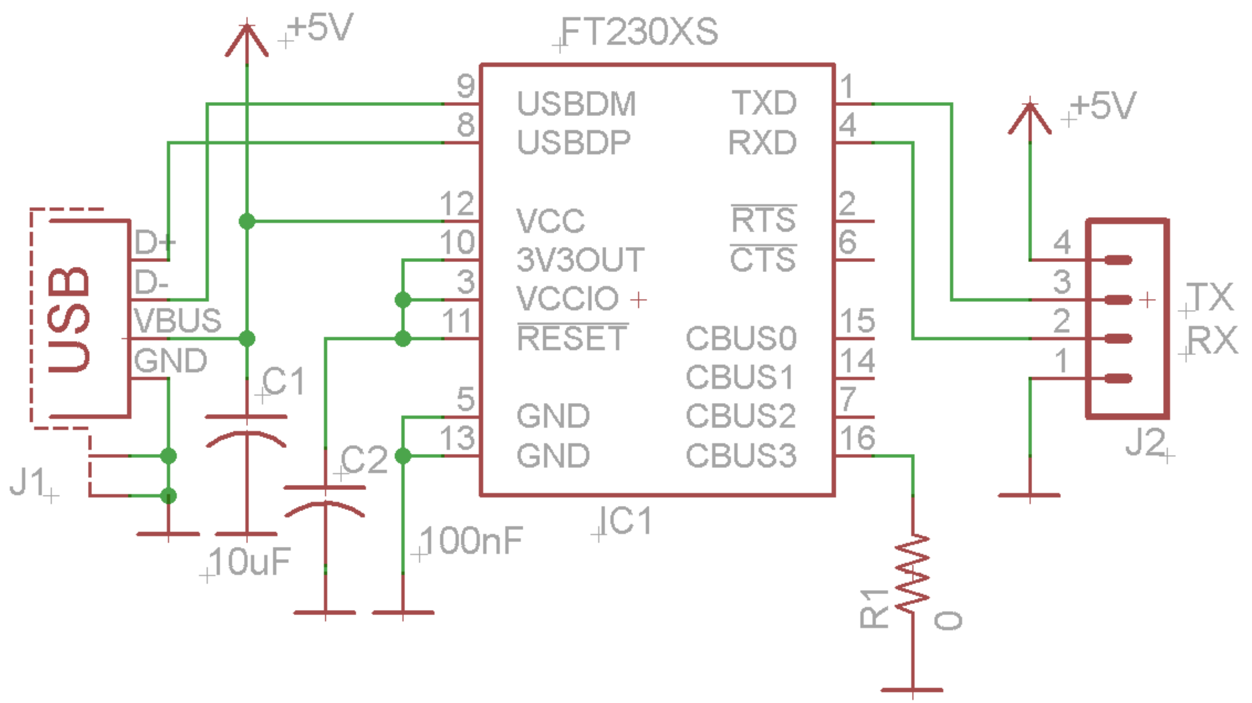
UART on ESP32

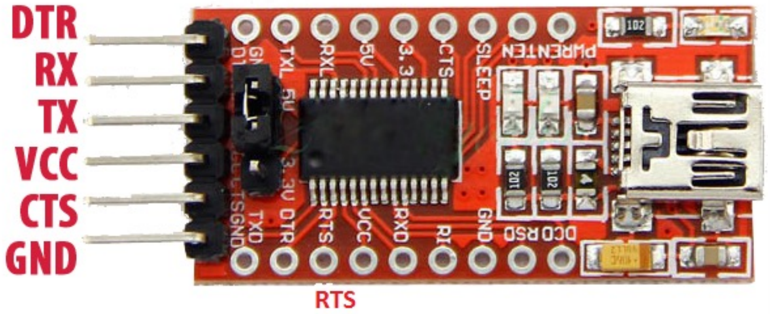
- There are three H/W UART controllers available on the ESP32 chip
  Serial0, Serial1, and Serial2.
- TTGO uses Serial0 for programming: do not remap..
- Pins can be reconfigured .
- ```
  void begin(unsigned long baud,
             uint32_t config=SERIAL_8N1,
             int8_t rxPin=-1,
             int8_t txPin=-1,
             bool invert=false,
             unsigned long timeout_ms = 20000UL)
  ```
- Advanced features like
  - flow-control
  - Interrupts
  - DMA, for seamless high-speed data transfer.


  ```

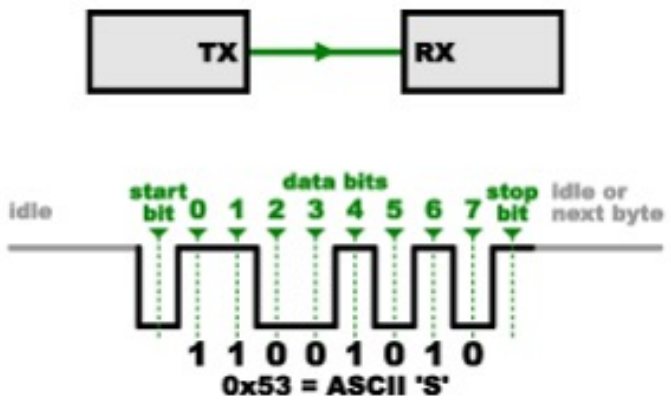
USB-Serial

- Small electronic circuit that can convert USB signal to UART signals
- Continue to use serial communications with modern computers
- Standard ICs (e.g., FTDI)





WHAT'S WRONG WITH UART?



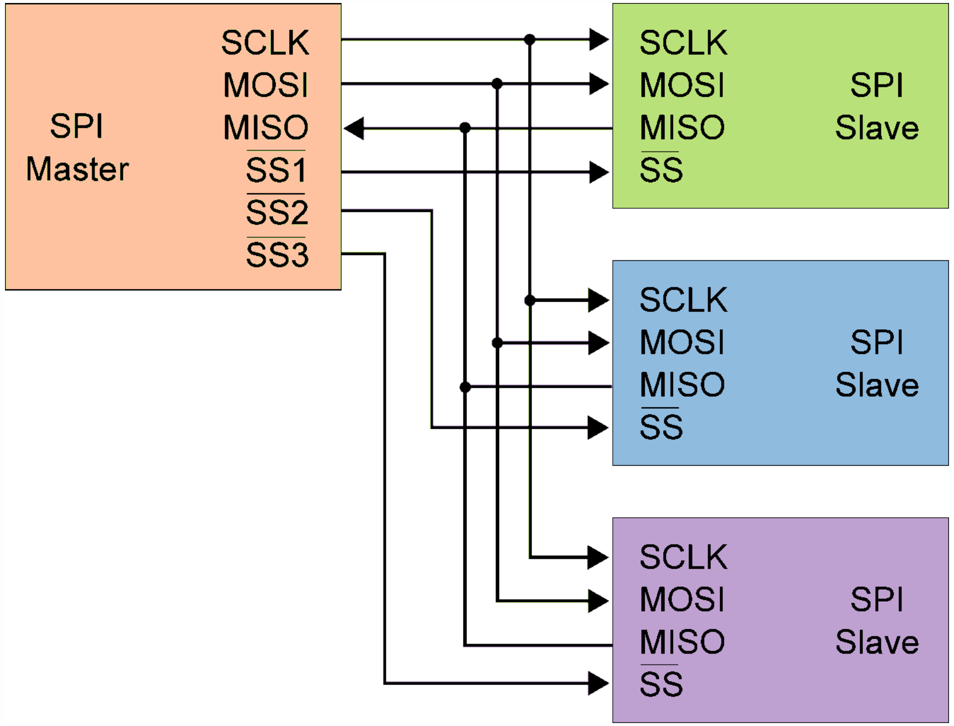
- It's (primarily) one-to-one communication.
- UART hardware required to send and receive data
- It is "asynchronous"
  - No guarantee that both sides are running at **precisely** the same rate.
  - Both sides need to agree on the transmission speed in advance.
  - Receiver needs to re-sync at the start of each byte.
  - Extra **start** and **stop bits** to each byte help the receiver sync up.
- UART mostly replaced by SPI, I2C in MCU, and Ethernet, USB in computers.
  - But it is still good for low-speed applications given simple implementation.

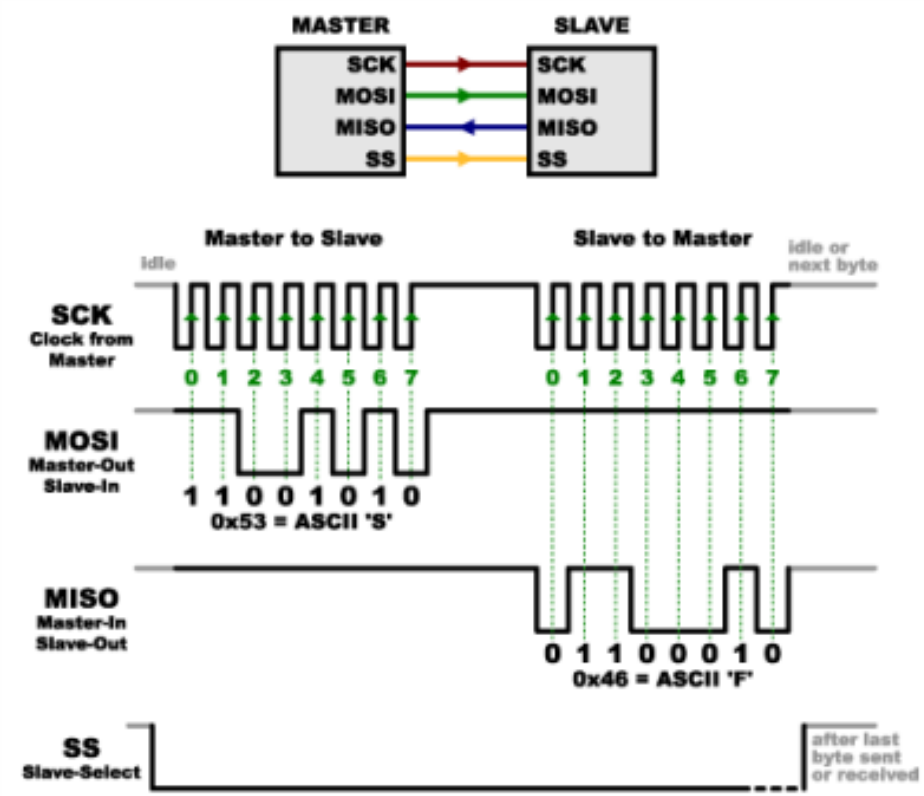
SPI COMMUNICATION

- **Serial Peripheral Interface** Bus
  - **Bus** type communication (one transmits, multiple receives).
  - **Slaves** send data to master at the **same time** when **master** is sending data to them.
- A "synchronous" data bus:
  - Only one side **generates** the clock signal (i.e., master).
  - If the slave needs to send a response back to the master, the master will continue **to generate a prearranged number of clock cycles** and **the slave will put the data on MISO**.
- Hardware can be a simple shift register.

SPI WIRING

- The desired **recipient** is selected individually with a **dedicated wire (Slave/Chip Select)**
- Requires **2 or 3 wires for the communication** +1 wire for **each device** in the bus
  - If no outputs available, there are **binary decoder chips** that can multiply the board SS outputs.



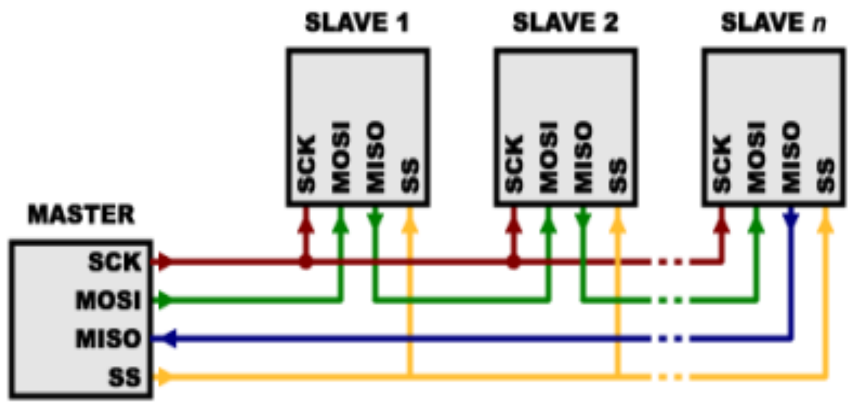


DAISY-CHAINED CONNECTION

- As the number of slaves increase, so do the number of slave-select lines.
- In this situation, the board layout of the system can become quite a challenge.
- *One layout alternative is daisy-chaining.*

- Data overflows from one slave to the next.
  - To send data to any one slave, you'll need to transmit enough data to reach all of them.
  - The first piece of data you transmit will end up in the last slave.

- Useful for output-only situations.
  - E.g., driving LEDs where you don't need to receive any data back



SPI on ESP32

- The ESP32 has four SPI peripheral devices: SPI0, SPI1, HSPI, VSPI
- **SPI0** is entirely dedicated to the flash cache the ESP32 uses to map the SPI flash device it is connected to into memory.
- **SPI1** is connected to the same hardware lines as SPI0 and is used to write to the flash chip.
- HSPI and VSPI are free to use (Hardware SPI, and Virtual SPI).

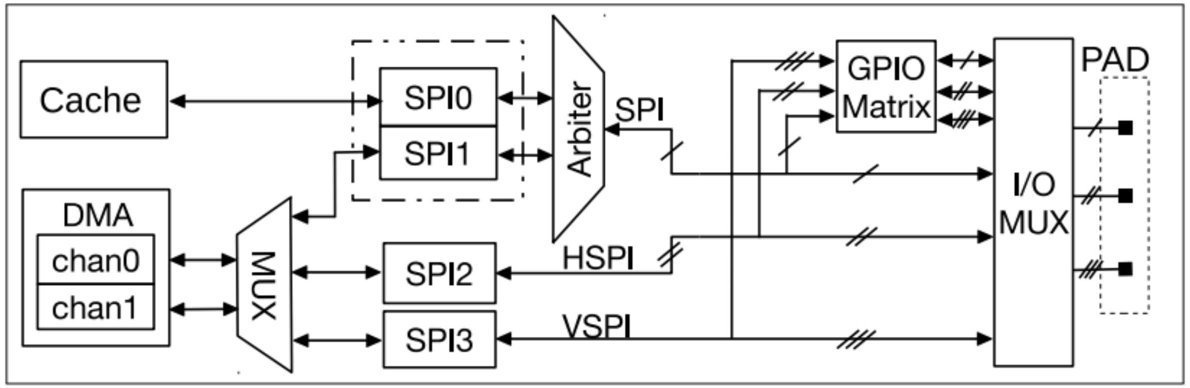


Figure 16: SPI Architecture

PROGRAMMING FOR SPI

Three ways you can communicate with SPI devices using an ESP32:

- Use the **SPI HW with default pin mapping**
  - It uses the SPI **hardware** built into the microcontroller.
  - Dedicated pins (fastest: 80MHz).

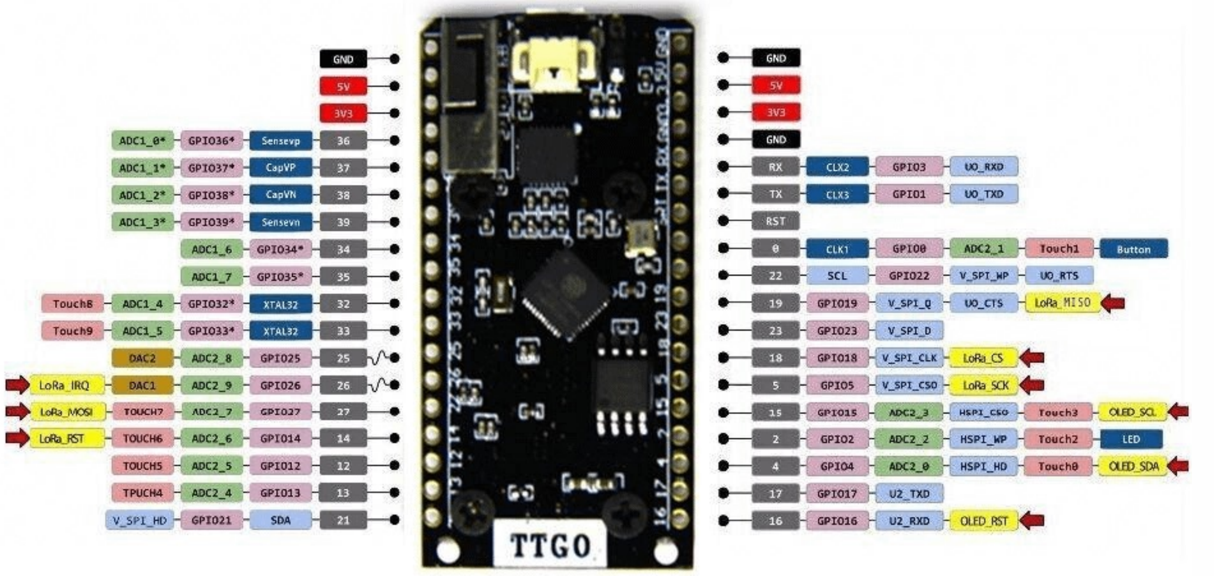
SPI

By default, the pin mapping for SPI is:

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

- Use the **SPI HW with GPIO pin re-mapping**
  - Signals **have** to be routed through the *GPIO Matrix*.
  - Slow compared to default pins (max: 40MHz).
  - Most SPI peripherals have speed < 40MHz

- You can use the **S/W SPI** libraries.
  - **Software**-based commands that will work on **any pins** but will be **slow.**
  - Useful if more than two SPI master is required.



PROGRAMMING FOR SPI

To write code for a new SPI device you need to note a few things:

- `void SPIClass::setFrequency(uint32_t freq)`
  - What is the maximum SPI speed your device can use?
  - If the default SPI is **too fast** for some devices you can adjust the data rate
  - E.g., 27MHz for onboard display

- `void SPIClass::setBitOrder(uint8_t bitOrder)`
  - Send data with the most-significant bit (**MSB**) first, or least-significant bit (**LSB**) first
  - Most SPI chips use MSB first data order.

- `void SPIClass::setDataMode(uint8_t dataMode)`
  - 4 modes
  - The **slave** will **read** the data on either the **rising edge** or the **falling edge** of the clock pulse
  - The clock can be considered "**idle**" when it is **high** or **low**

```
#include <SPI.h>

SPIClass SPI1(HSPI);

SPI1.begin();
or
SPI1.begin(SCLK, MISO, MOSI, CS);

SPI1.beginTransaction(SPISettings(3000000, MSBFIRST, SPI_MODE2));

// Make Slave Select Low

SPI1.transfer()
.
.
SPI1.transfer()

// Make Slave Select High

SPI1.endTransaction()
```

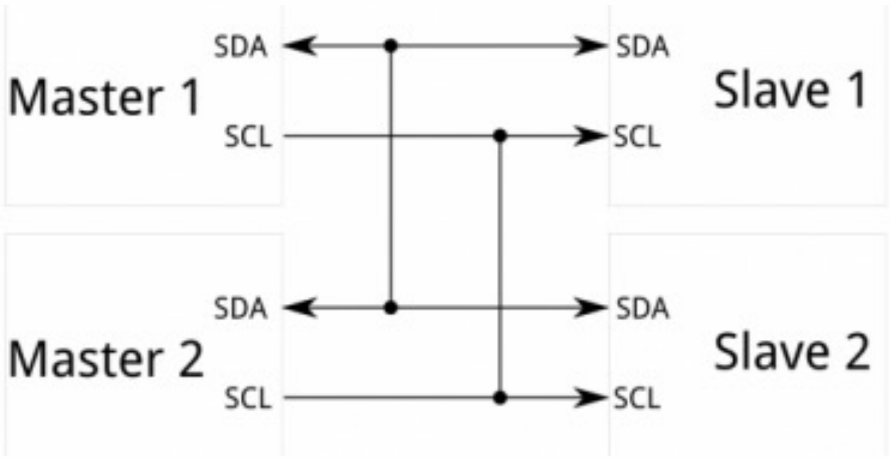
PROS AND CONS OF SPI

- Advantages of SPI:
  - It's faster than asynchronous serial
    - UART's highest transmission rate is around 512000 bits per second
    - SPI supports clock rates upwards of 10MHz (10 million bits per second)
  - The receive hardware can be a simple shift register
  - It supports multiple slaves

- Disadvantages of SPI:
  - It requires more signal lines (wires) than other communications methods
  - The **communications** must be **well-defined** in advance
    - you can't send random amounts of data whenever you want
  - The master must control all communications (slaves can't talk directly to each other)
  - It usually requires separate **SS** lines to each slave
    - which can be problematic if numerous slaves are needed.

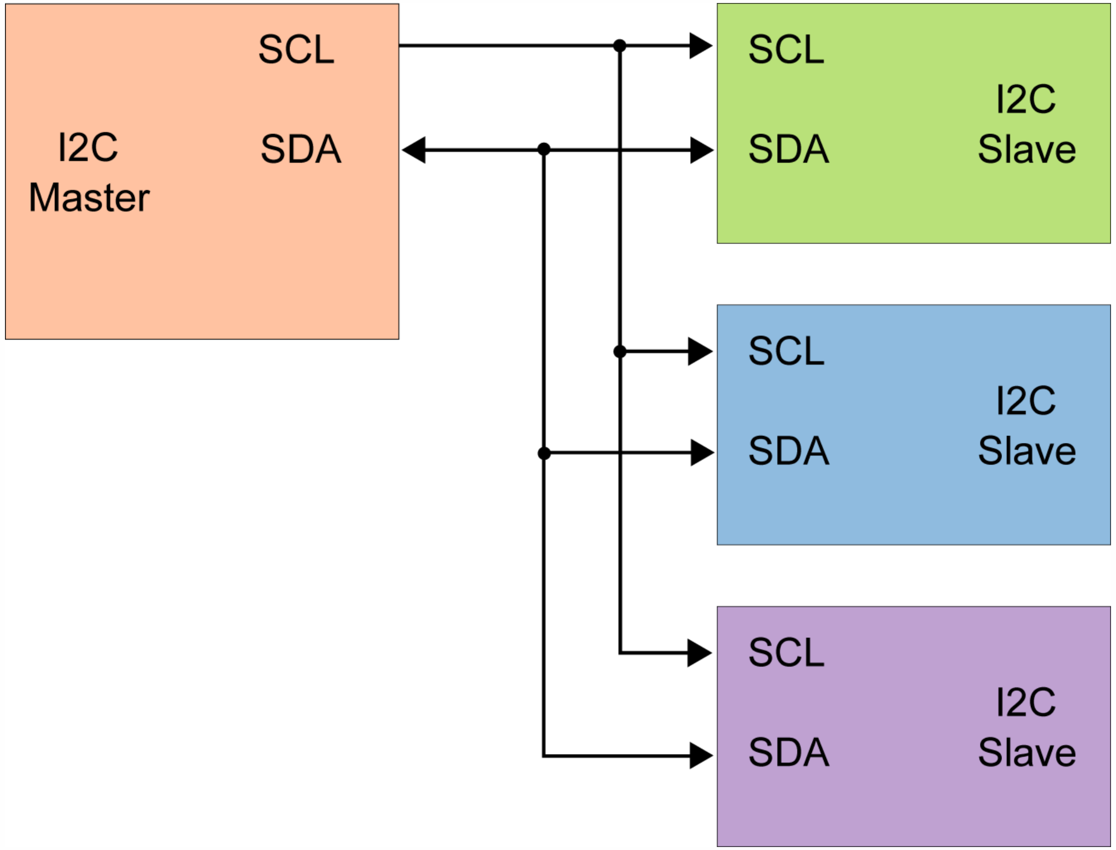
I2C COMMUNICATION

- Inter-Integrated Circuit (I2C) Protocol
- **Bus** type communication (one transmits, multiple receives)
- All the devices in the bus have **a 7-bit address**
- Unlike SPI, it supports a **multi-master** system
  - Master devices **can't** talk to each other
  - Data rates fall between 100kHz or 400kHz
- ESP32 also supports 10-bit addresses (Dual addressing mode)



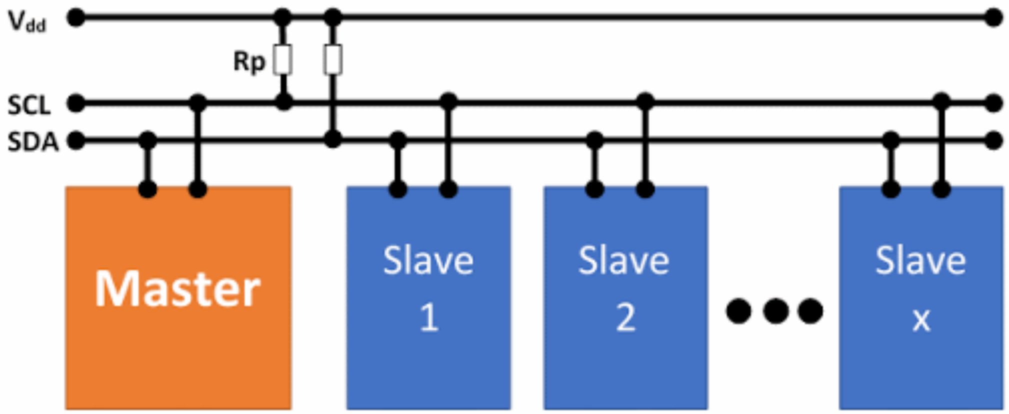
I2C WIRING

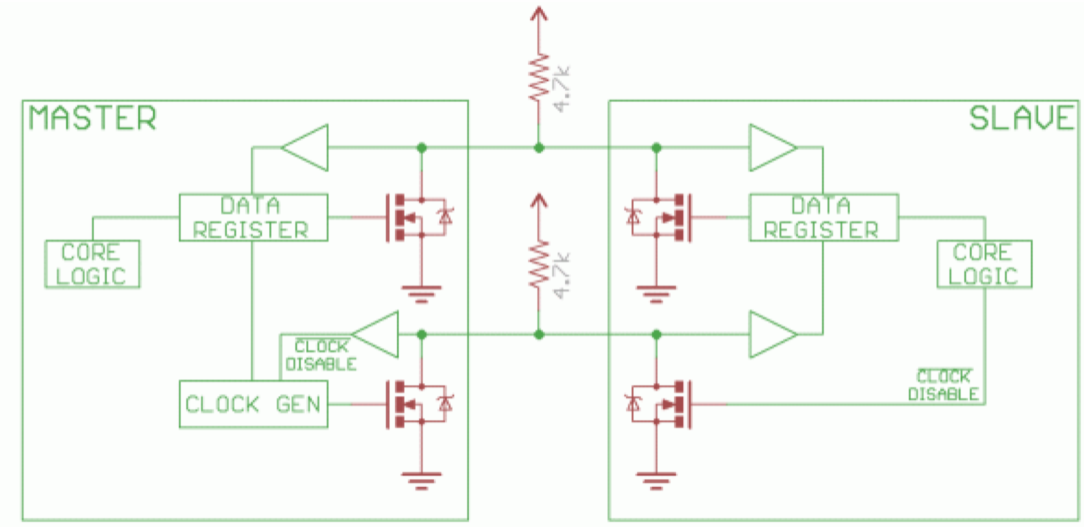
- I2C only requires **two signal wires** (SDA & SCL)
  - SDA = Serial **Data** Line
  - SCL = Serial **Clock** Line



I2C AT THE HARDWARE LEVEL

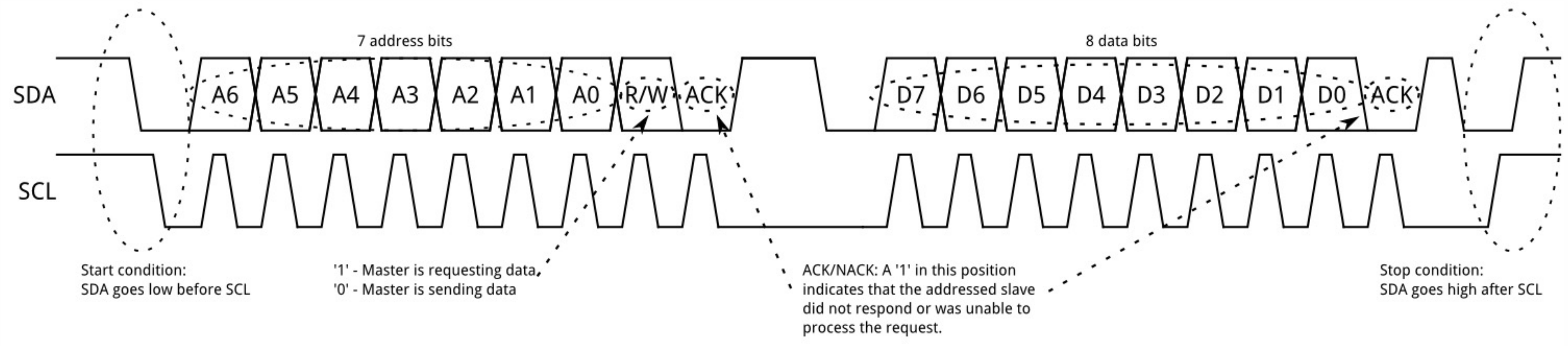
- The clock signal is **always** generated by the current **bus master**
- I2C bus drivers are "**open drain**", meaning that they **can pull** the corresponding signal line **low**, but **cannot drive** it **high**
- A good rule of thumb is to use a **4.7k/10k** resistor





I2C PROTOCOL (DETAILS NOT CRITICAL)

- Two types of frame:
  - An address frame
  - Several data frames
- Communication starts when a master sends out a **start condition**: it will pull the SDA line low, and will then pull the SCL line high.
- The first eight pulses are used to shift out a byte consisting of a 7-bit **address** and a **read/write bit**.
- If a slave with this address is active on the bus, the slave can answer by pulling the SDA low (**ACK**) on the ninth clock pulse.
- The master can then send out more 9-bit clock pulse clusters and, depending on the read/write bit sent, the device or the master will **shift out data on the SDA line**,
  with the other side acknowledging the transfer by pulling the SDA low on the ninth clock pulse.



I2C USING ARDUINO (CODE FOR MASTER READER)

```
#include <Wire.h>

void setup() {
    // join i2c bus (address optional for master)
    Wire.begin();

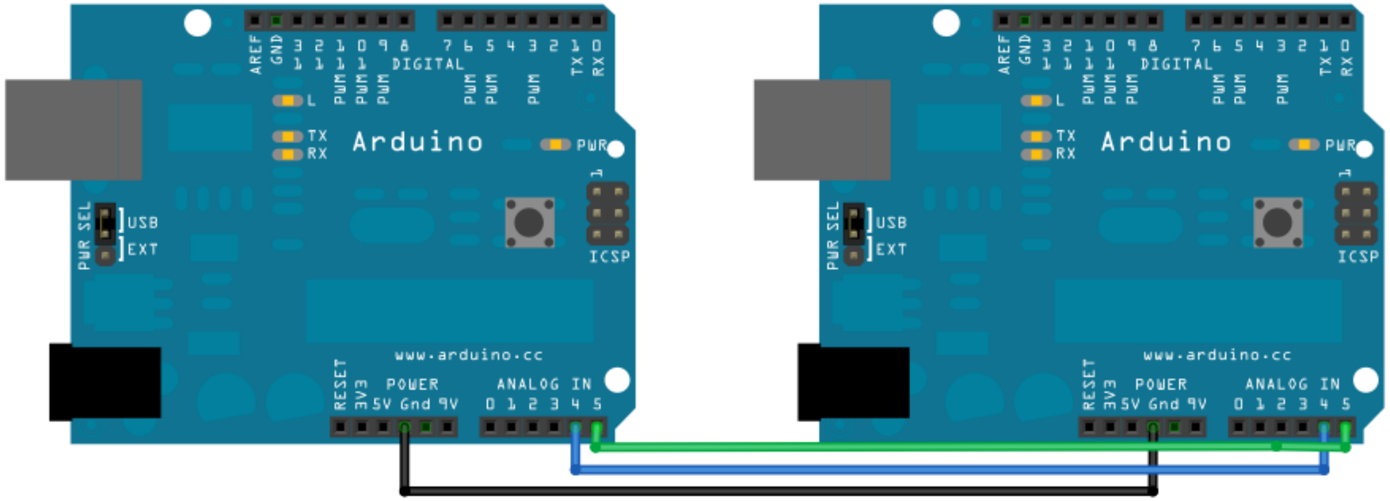
    // start serial for output
    Serial.begin(9600);
}

void loop() {
    // request 6 bytes from slave device #8
    Wire.requestFrom(8, 6);

    // slave may send less than requested
    while (Wire.available()) {
        // receive a byte as character
        char c = Wire.read();

        // print the character
        Serial.print(c);
    }

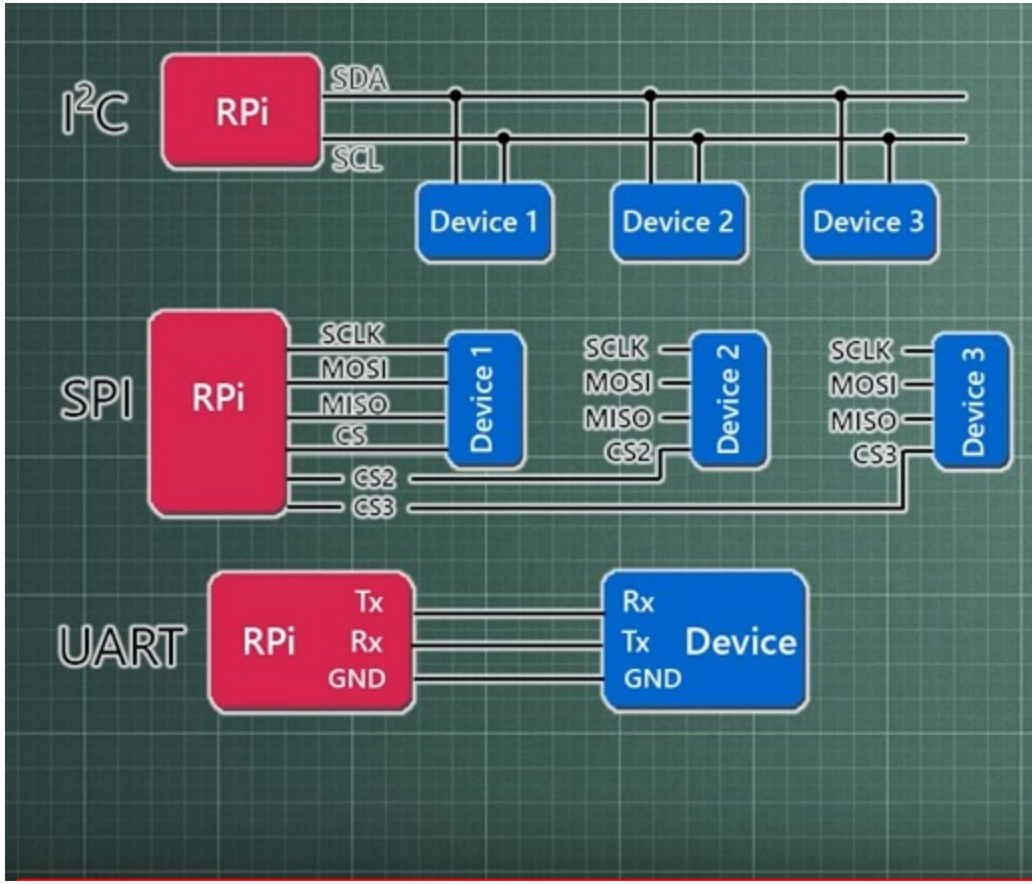
    delay(500);
}
```



Pros and cons of I2C

- Advantages of I2C
  - More than one master devices
  - Less wires than SPI
  - Software based addressing (easy to scale)
- Disadvantages of I2C
  - It is half duplex mode of communication.

SUMMARY



UART	SPI	I2C
1 to 1	Duplex	Simplex
Asynchronous	Synchronous	Synchronous
2 wires	4+ wires	2 wires
20Kbps	25Mbps	1Mbps
Acknowledge pins	NO acknowledge	acknowledge
Medium power	Lower power	Medium power
15m distance	20cm distance	1m distance