When you connect a microcontroller to a sensor, display, or other module, do you ever think about how the two devices talk to each other? What exactly are they saying? How are they able to understand each other?

Communication between electronic devices is like communication between humans. Both sides need to speak the same language. In electronics, these languages are called communication protocols. Luckily for us, there are only a few communication protocols we need to know when building most DIY electronics projects. These are

1. Serial Peripheral Interface (SPI)
2. Inter-Integrated Circuit (I2C)
3. Universal Asynchronous Receiver/Transmitter driven communication (UART)

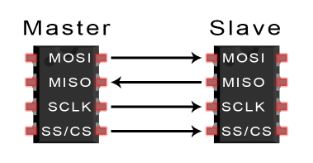
Also, for the record, there are two ways in which communication can occur – serial and parallel.

First, we’ll explain in detail how SPI works. In the next article, we’ll discuss UART driven communication, and in the third article, we’ll dive into I2C.

**SPI, I2C, and UART are quite a bit slower than protocols like USB, ethernet, Bluetooth, and Wi-Fi** but they’re a lot more simple and use less hardware and system resources. SPI, I2C, and UART are ideal for communication between microcontrollers and between microcontrollers and sensors where large amounts of high speed data don’t need to be transferred.

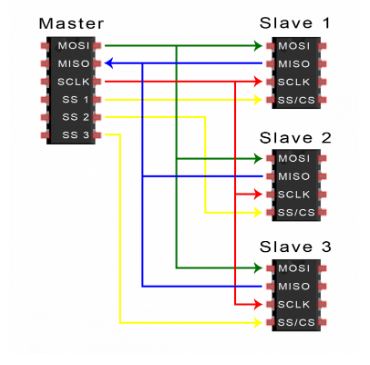
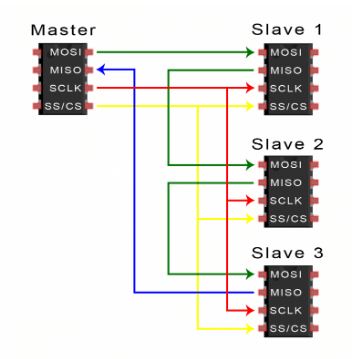
# Serial Peripheral Interface (SPI)

## Introduction

* One unique benefit of SPI is the fact that **data can be transferred without interruption.** Any number of bits can be sent or received in a continuous stream. **With I2C and UART, data is sent in packets**, limited to a specific number of bits. Start and stop conditions define the beginning and end of each packet, so the data is interrupted during transmission.
* Devices communicating via SPI are in a **master-slave relationship.** The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master. The simplest configuration of SPI is a **single master, single slave system**, but one master can control more than one slave (more on this below).

1. **MOSI (Master Output/Slave Input)** – Line for the master to send data to the slave.
2. **MISO (Master Input/Slave Output)** – Line for the slave to send data to the master.
3. **SCLK (Clock)** – Line for the clock signal.
4. **SS/CS (Slave Select/Chip Select)** – Line for the master to select which slave to send data to. Theoretically, the number of slaves is limited by the load capacitance of the system, which reduces the ability of the master to accurately switch between the voltage levels.

## Working

* **The Clock**
  + The clock signal synchronizes the output of data bits from the master to the sampling of bits by the slave. One bit of data is transferred in each clock cycle, so the speed of data transfer is determined by the frequency of the clock signal. **SPI communication is always initiated by the master** since the master configures and generates the clock signal.
  + Any communication protocol where devices share a clock signal is known as **synchronous.** There are also asynchronous methods that don’t use a clock signal. For example, in UART communication, both sides are set to a pre-configured baud rate that dictates the speed and timing of data transmission but we will cover that later.
  + The clock signal in SPI can be modified using the properties of **clock polarity and clock phase.** These two properties work together to define when the bits are output and when they are sampled.
    - **Clock polarity** can be set by the master to allow for bits to be output and sampled on either the rising or falling edge of the clock cycle.
    - **Clock phase** can be set for output and sampling to occur on either the first edge or second edge of the clock cycle, regardless of whether it is rising or falling.
* **SLAVE Select**
  + The master can choose which slave it wants to talk to by setting the slave’s CS/SS line to a low voltage level. In the idle, non-transmitting state, the slave select line is kept at a high voltage level.
  +  Multiple CS/SS pins may be available on the master, which allows for multiple slaves to be wired in parallel.
  + If only one CS/SS pin is present, multiple slaves can be wired to the master by daisy-chaining.
* **MOSI & MISO**
  + The master sends data to the slave bit by bit, in serial **through the MOSI line**. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with **the most significant bit first.**
  + The slave can also send data back to the master through **the MISO line in serial**. The data sent from the slave back to the master is usually sent with **the least significant bit first**.

## Steps Involved

1. The master outputs the clock signal
2. The master switches the SS/CS pin to a low voltage state which activates the slave.
3. The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received:
4. If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received:

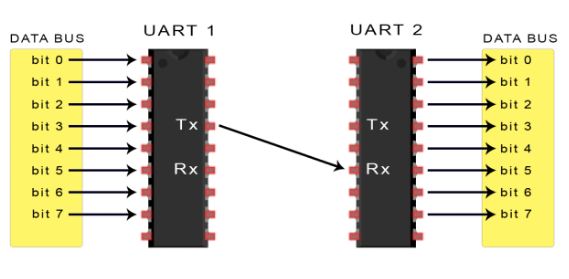
## Advantages

1. No start and stop bits, so the data can be streamed continuously without interruption
2. No complicated slave addressing system like I2C (covered later).
3. Higher data transfer rate than I2C (almost twice as fast)
4. Separate MISO and MOSI lines, so data can be sent and received at the same time.

## Disadvantages

1. Uses four wires (I2C and UARTs use two).
2. No acknowledgement that the data has been successfully received (I2C has this).
3. No form of error checking like the parity bit in UART.
4. Only allows for a single master.

# UART Communication

UART stands for Universal Asynchronous Receiver/Transmitter. It’s not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART’s main purpose is to transmit and receive serial data.

One of the best things about UART is that it only uses two wires to transmit data between devices. The principles behind UART are easy to understand.

In UART communication, two UARTs communicate directly with each other. The transmitting **UART converts parallel data from a controlling device like a CPU into serial form**, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. **Only two wires are needed to transmit data between two UARTs.** Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:

**UARTs transmit data asynchronously**, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting **UART adds start and stop bits to the data packet being transferred.** These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

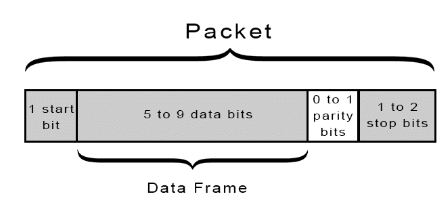
When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as **the baud rate.** Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). **Both UARTs must operate at about the same baud rate.** The baud rate between the transmitting and receiving UARTs can only differ by **about 10%** before the timing of bits gets too far off.

Both UARTs must also must be configured to transmit and receive the same data packet structure.

## Working

1. The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form.
2. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet.
3. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end:

## More Info

UART transmitted data is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits **(depending on the UART)**, an optional parity bit, and 1 or 2 stop bits:

1. **Start Bit**
   1. The UART data transmission line is normally held at a high voltage level when it’s not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.
2. **Data Frame**
   1. The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits **(depending on the physical UART IC)** long if a parity bit is used.
   2. If no parity bit is used, the data frame can be 9 bits long. In most cases, the data is sent with the least significant bit first.
3. **Parity**
   1. Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long distance data transfers.
   2. After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number.
   3. When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd; or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.
4. **Stop Bits**
   1. To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

## Advantages

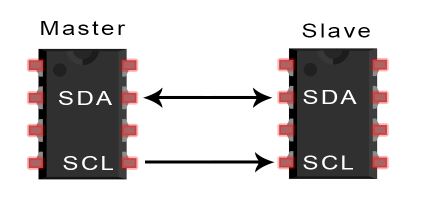
1. Only uses two wires
2. No clock signal is necessary
3. Has a parity bit to allow for error checking
4. The structure of the data packet can be changed as long as both sides are set up for it
5. Well documented and widely used method

## Disadvantages

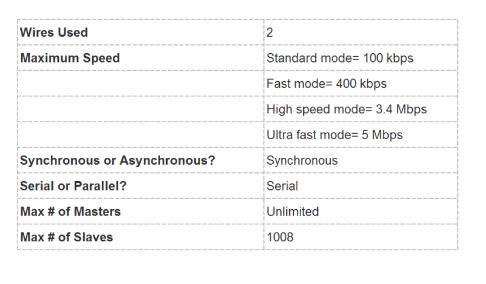
1. The size of the data frame is limited to a maximum of 9 bits
2. Doesn’t support multiple slave or multiple master systems
3. The baud rates of each UART must be within 10% of each other

# I2C Communication

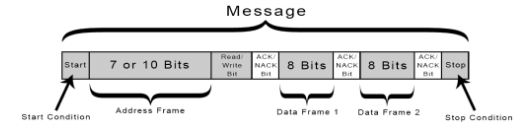
I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and **you can have multiple masters controlling single, or multiple slaves.** This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

Like UART communication, **I2C only uses two wires to transmit data between devices:**

1. **SDA (Serial Data)** – The line for the master and slave to send and receive data.
2. **SCL (Serial Clock)** – The line that carries the clock signal.

Like SPI, **I2C is synchronous**, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master

## Working

With I2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:

1. **Start Condition:** The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.
2. **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.\
3. **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
4. **Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
5. **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

## Addressing

I2C doesn’t have slave select lines like SPI, so it needs another way to let the slave know that data is being sent to it, and not another slave. It does this by addressing. The address frame is always the first frame after the start bit in a new message.

The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address. If the address matches, it sends a low voltage ACK bit back to the master. If the address doesn’t match, the slave does nothing and the SDA line remains high.

## Read/Write Bit

The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level.

## The Data Frame

After the master detects the ACK bit from the slave, the first data frame is ready to be sent.

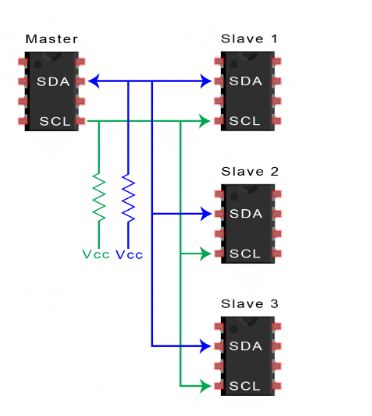
The data frame is always 8 bits long, and sent with the most significant bit first. Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully. The ACK bit must be received by either the master or the slave (depending on who is sending the data) before the next data frame can be sent.

After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission. The stop condition is a voltage transition from low to high on the SDA line after a low to high transition on the SCL line, with the SCL line remaining high.

## Steps

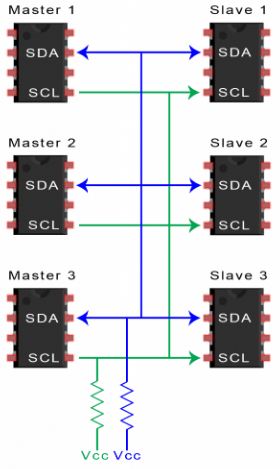
1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low:
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:
3. Each slave compares the address sent from the master to its own address. If the address matches, **the slave returns an ACK bit by pulling the SDA line low for one bit**. If the address from the master does not match the slave’s own address, the slave leaves the SDA line high.
4. The master sends or receives the data frame:
5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:
6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:

## Connecting Multiple Slaves

Because I2C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (27) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (210) unique addresses. To connect multiple slaves to a single master, wire them like this, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:

## Multiple Masters

Multiple masters can be connected to a single slave or multiple slaves.

**The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line.** To solve this problem, **each master needs to detect if the SDA line is low or high before transmitting a message.** If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message. If the SDA line is high, then it’s safe to transmit the message. To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:

## ADVANTAGES

1. Only uses two wires
2. Supports multiple masters and multiple slaves
3. ACK/NACK bit gives confirmation that each frame is transferred successfully
4. Hardware is less complicated than with UARTs
5. Well known and widely used protocol

## DISADVANTAGES

1. Slower data transfer rate than SPI
2. The size of the data frame is limited to 8 bits
3. More complicated hardware needed to implement than SPI