UART communication protocol

UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device to the receiving end.

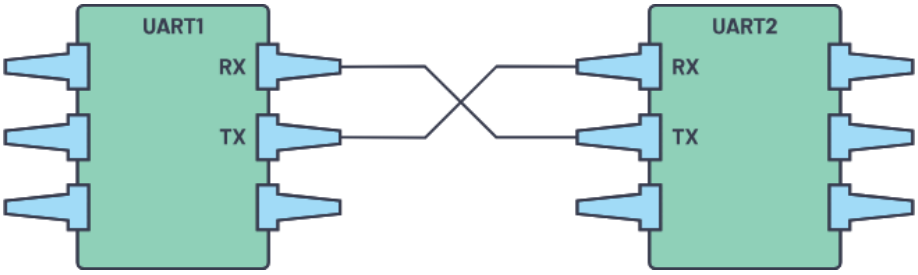

Fig 1. Two UARTs directly communicate with each other.

The two signals of each UART device are named:

- Transmitter (Tx)
- Receiver (Rx)

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication.

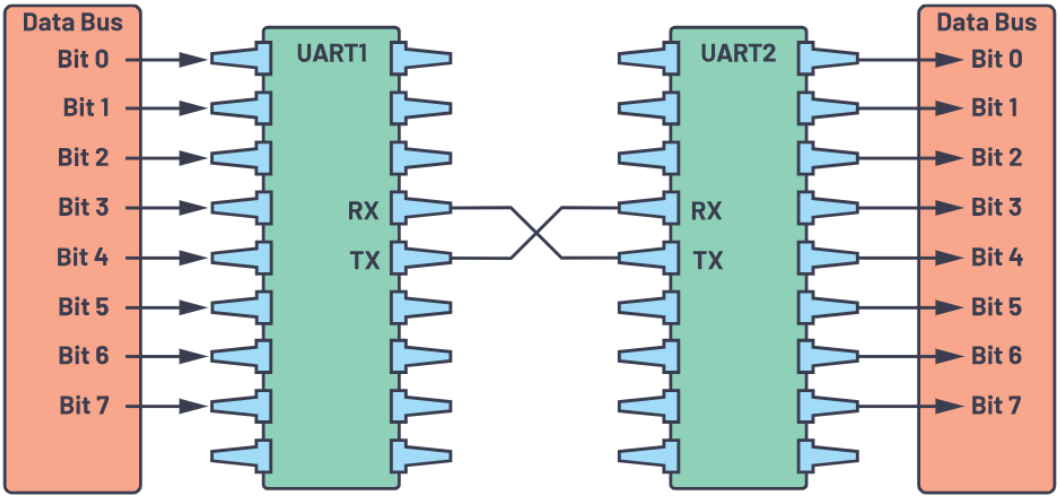

Fig 2. UART with data bus.

The transmitting UART is connected to a controlling data bus that sends data in parallel. The data will now be transmitted on the transmission line (wire) serially, bit by bit, to the receiving UART. This will convert the serial data into parallel for the receiving device.

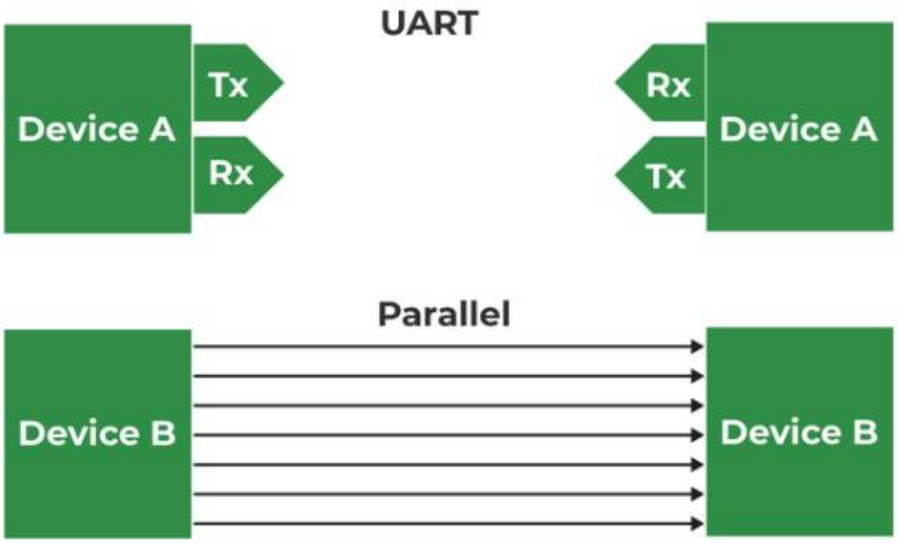

Fig 3. UART

The UART lines serve as the communication medium to transmit and receive one data to another. UART device has a transmit and receive pin dedicated to transmitting or receiving.

Wires	2
Speed	9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1000000, 1500000
Methods of Transmission	Asynchronous
Maximum Number of Masters	1
Maximum Number of Slaves	1

The UART interface does not use a clock signal to synchronize the transmitter and receiver devices; it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bitstream based on its clock signal while the receiver uses its internal clock signal to sample the incoming data. The point of synchronization is managed by having the same baud rate on both devices.

Data Transmission

In UART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, data frame, a parity bit, and stop bits.

Fig 4. UART packet.

Start Bit

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 (1) 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.

Data Frame

The data frame contains the actual data being transferred. It can be five (5) bits up to eight (8) bits long if a parity bit is used. If no parity bit is used, the data frame can be nine (9) bits long. In most cases, the data is sent with the least significant bit first.

Parity

Parity describes the number for being even or odd. 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.

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. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.

- In **even parity**, this bit is set such that the total number of 1s in the frame will be even.
- In **odd parity**, this bit is set such that the total number of 1s in the frame will be odd.

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.

Stop Bits

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 one (1) to two (2) bit(s) duration.

Steps of UART Transmission

First: The transmitting UART receives data in parallel from the data bus.

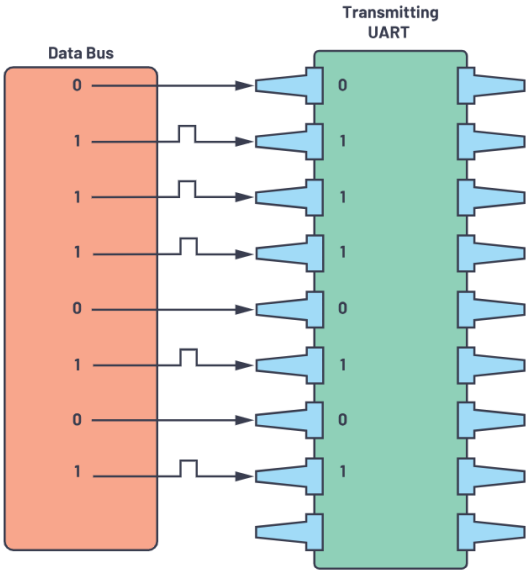

Fig 5. Data bus to the transmitting UART

Second: The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame.

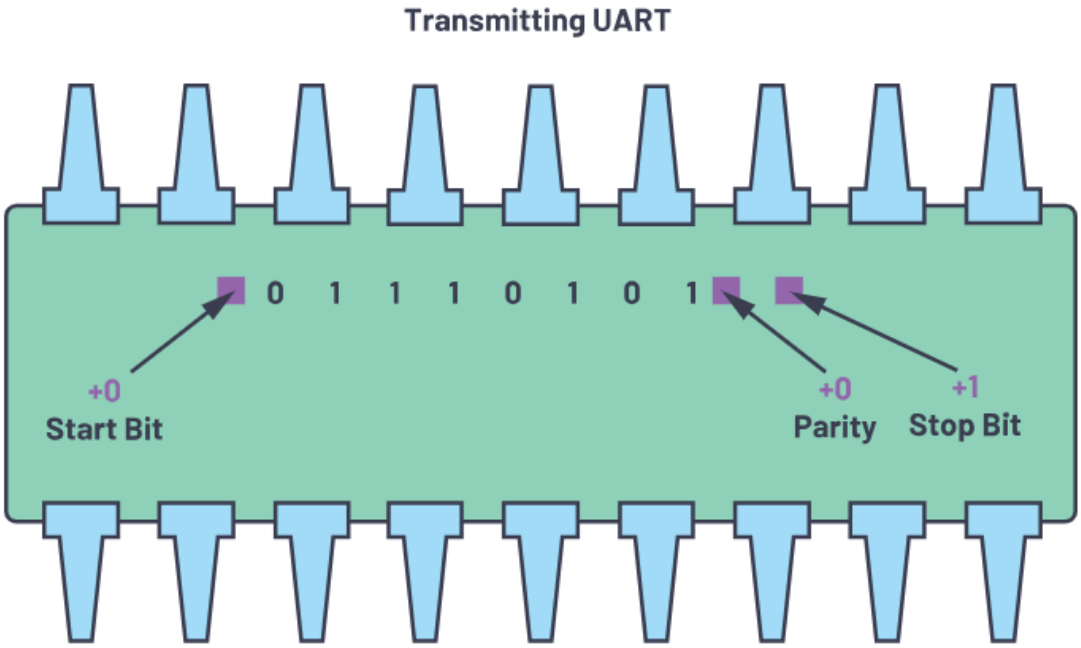

Fig 6. UART data frame at the TX side.

Third: The entire packet is sent serially starting from the start bit to the stop bit from the transmitting UART to the receiving UART. The receiving UART samples the data line at the preconfigured baud rate.

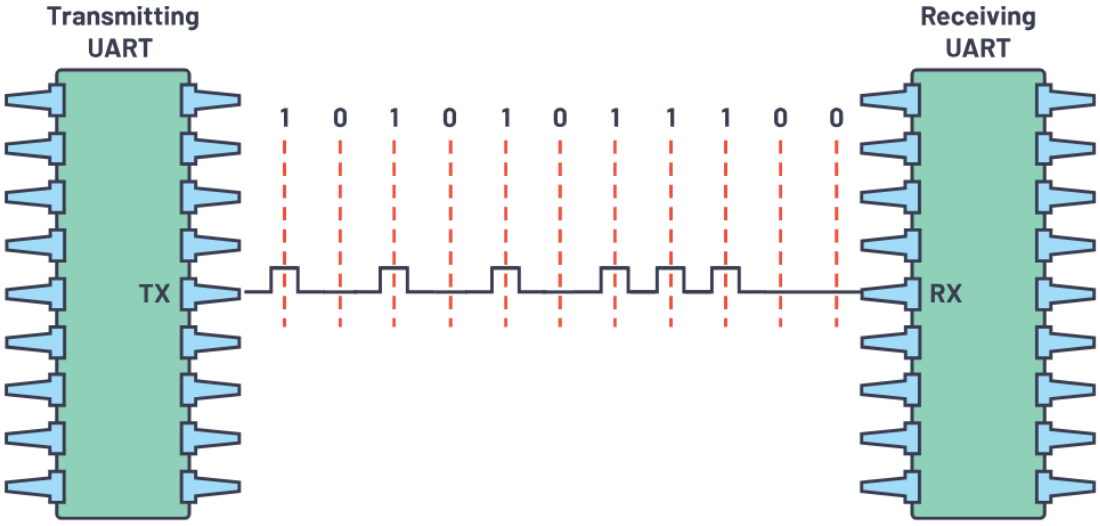

Fig 7.UART transmission.

Fourth: The receiving UART discards the start bit, parity bit, and stop bit from the data frame.

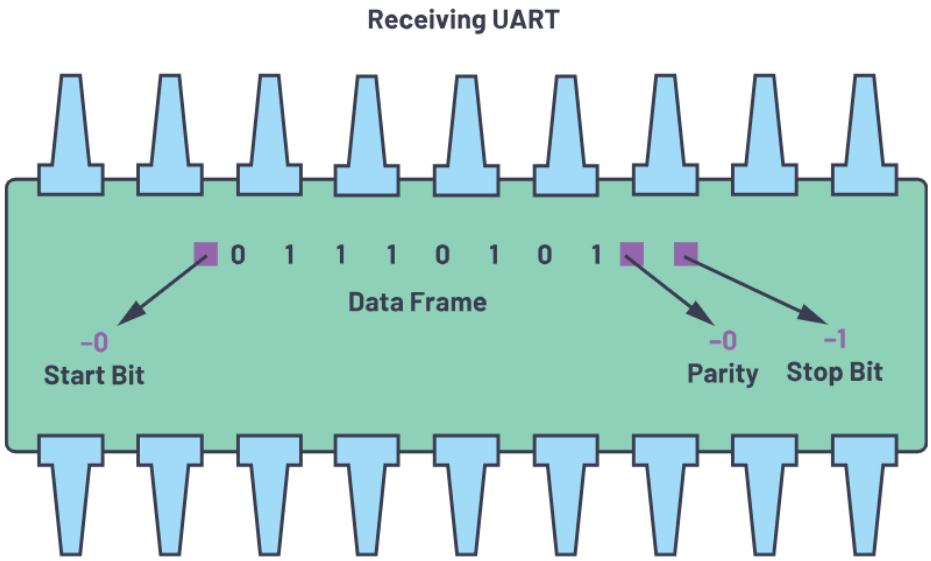

Fig 8.The UART data frame at the Rx side.

Fifth: The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end.

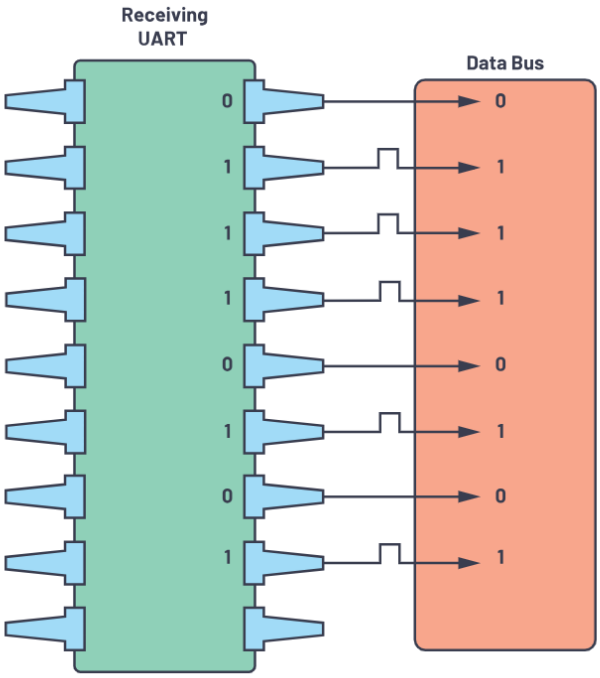

Fig 9. Receiving UART to data bus.

Frame Protocol

One key feature that is available in UART yet not fully used is the implementation of a frame protocol. The main use and importance of this is an added value for security and protection on each device.

For instance, when two devices use the same UART frame protocol, there are tendencies that, when connecting to the same UART without checking the configuration, the device will be connected to different pins that may cause malfunctions in the system.

On the other hand, implementing this ensures security because of the need to parse the information received in alignment with the design frame protocol. Each frame protocol is specifically designed to be unique and secure.

In designing a frame protocol, designers can set the desired headers and trailers, including CRC, to different devices.

Fig 10. Sample UART frame protocol.

Header 1 (H1 is 0xAB) and Header 2 (H2 is 0xCD)

Header is the unique identifier that determines if you are communicating with the correct device.

Command (CMD) Selection

Command will depend on the list of command designed to create the communication between two devices.

Data Length (DL) per Command

Data length will be based on the command chosen. You can maximize the length of data depending on the command chosen, so it can vary based on the selection. In that case, the data length can be adjusted.

Data n (Varying Data)

Data is the payload to be transferred from devices.

Trailer 1 (T1 is 0xE1) and Trailer 2 (T2 is 0xE2)

Trailers are data that are added after the transmission is ended. Just like the Header, they can be uniquely identified.

Cyclic Redundancy Checking (CRC Formula)

The cycling redundancy checking formula is an added error detecting mode to detect accidental changes to raw data. The CRC value of the transmitting device must always be equal to the CRC computations on the receiver's end.

Advantages of UART

1. **Limited Connectivity**:
    - Restricted to one master and one slave per UART line.
2. **Distance Constraints**:
    - Susceptible to errors over long distances or in noisy environments.
3. **Baud Rate Sensitivity**:
    - Requires matching baud rates for successful communication.

Limitations of UART

1. **Limited Connectivity**:
    - Restricted to one master and one slave per UART line.
2. **Distance Constraints**:
    - Susceptible to errors over long distances or in noisy environments.
3. **Baud Rate Sensitivity**:
    - Requires matching baud rates for successful communication.

I2C communication protocol

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:

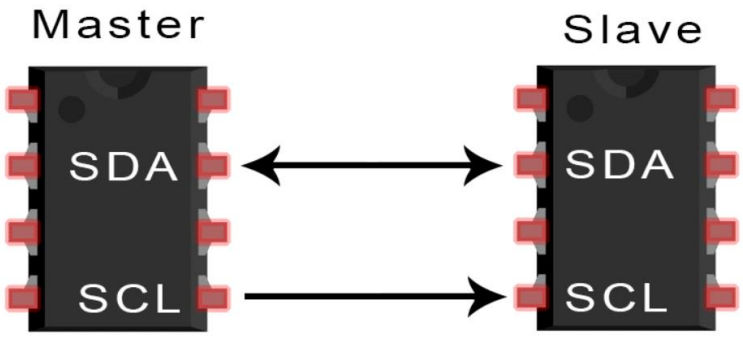

Fig 1.

SDA (Serial Data) – The line for the master and slave to send and receive data.

SCL (Serial Clock) – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

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.

Wires	2
Maximum Speed	Standard mode= 100 kbps Fast mode= 400 kbps High speed mode= 3.4 Mbps Ultra-fast mode= 5 Mbps
Methods of Transmission	Synchronous
Serial or parallel	Serial
Maximum Number of Masters	Unlimited
Maximum Number of Slaves	1008

Common uses of I2C where speed is mostly out of the equation, while simplicity and low cost are prioritized, include:

- Turning the power supply on and off.
- Reading sensors' data.
- Changing the volume in speakers,
- Controlling smaller OLEDs or LCDs.
- Changing the hue, color balance, contrast, and brightness settings in monitors.
- Accessing the RTCs.
- Accessing low-speed DACs (digital-to-analog converters) and ADCs (analog-to-digital converters).

HOW I2C WORKS

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:

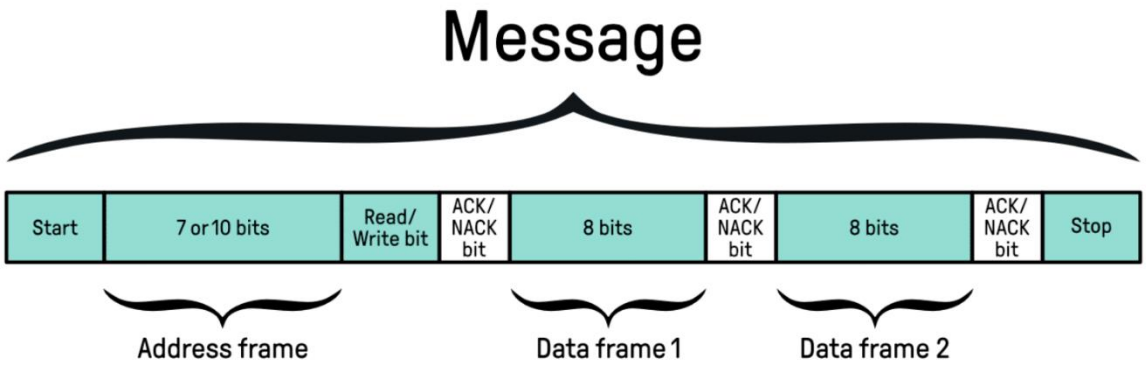

Fig 2

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.

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.

Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

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).

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.

Start and Stop condition

To send the data among the devices, the **controller will first generate a clock signal** through the SCL line. This will synchronize the data transfer between the devices on the I2C bus. The controller will generate a clock pulse on the clock line for all data bits, even the acknowledge bit. A clock pulse means that the SCL line will go from high to low and back in sequence. The data will be carried through a single SDA line.

These two lines are open-drain, meaning they need to use pull-up resistors. Commonly used values of resistors range from 2K for higher speeds at about 400 kbps, up to 10K for lower speeds at about 100 kbps.

Both SCL and SDA lines can be either low or high. The start condition (marked S) will occur when the SDA line drops low while the SCL line is still high. At the end of the message, the stop condition (marked P) will occur when the SCL line goes or is high, and the SDA line goes high after that. Start and stop bits are unique signals that can only be generated by a controller, and never by a peripheral.

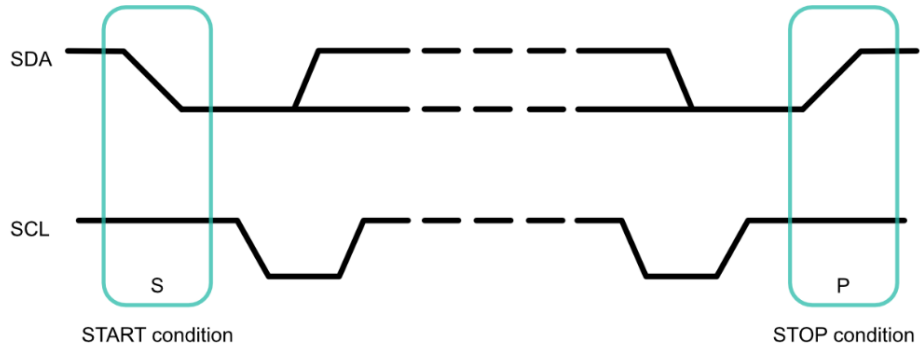

Fig 3

Data transfer

Data is sent between the start and stop conditions. It is transferred in bytes, the 8-bit packets. There is no limit on the number of bytes that can be sent. All of them must be followed by an ACK/NACK bit. This bit signals whether the device is ready to work with another data frame or not.

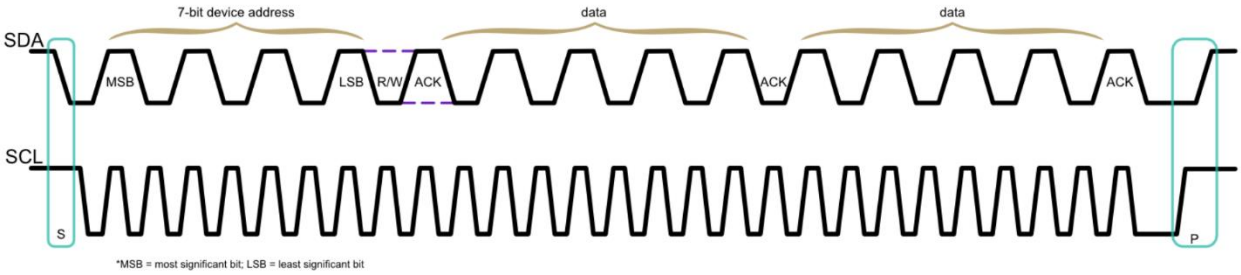

Fig 4

After the start condition, the controller will start sending pulses on the SCL line. For each clock pulse a controller sends, one bit of data is transferred. **The SDA line can only change when the SCL line is low**. When it is high, the SDA line will remain stable, either high or low.

If the SDA line is low on the Read/Write bit, the controller will write to the peripheral. If it's high, it will request to read data from the peripheral.

When it comes to the acknowledgment bit, if the SDA line is low, the peripheral successfully understood the frame. If the SDA line is high, the peripheral didn't understand and acknowledge the frame.

Repeated start

A controller will sometimes need to exchange more than one message at once. Since there can be multiple controllers on one bus, it must be ensured that the exchange doesn't get interrupted. Otherwise, another controller could interfere and take control of the bus, and the initial message would be lost. This is why a **repeated start** condition was defined.

As the name suggests, repeated start does the start condition again. To perform it, the SDA line is allowed to go high while the SCL line is low. Then the SCL line goes high as well. Once both lines are high, a new start condition begins the same way as the first one. The structure of this new message is the same as any other. An address frame is specified with data frames

following. Since there was no stop condition up to that point, the previous message hadn't truly ended. Thus, the current controller is still in control of the bus.

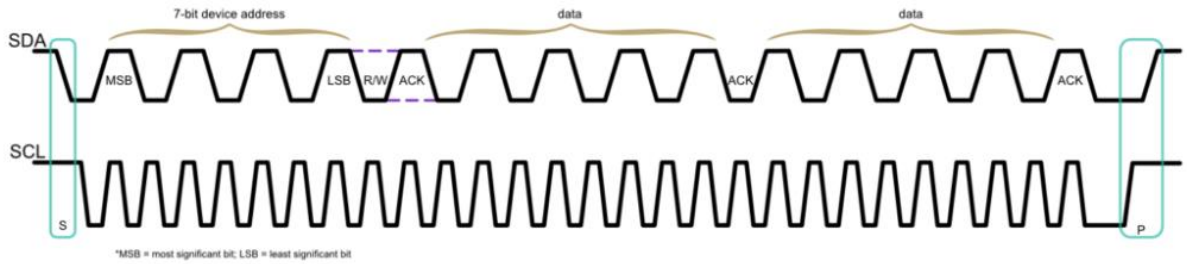

Fig 5. Message with a repeated start condition.

A repeated start condition is essentially the same as a start condition, with the only difference being it can be initiated indefinitely. The controller will maintain control of the bus until it issues a stop condition, no matter the number of repeated starts.

A message can have only one stop condition. After it, another controller can take control of the bus and start its message.

STEPS OF I2C DATA TRANSMISSION

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:

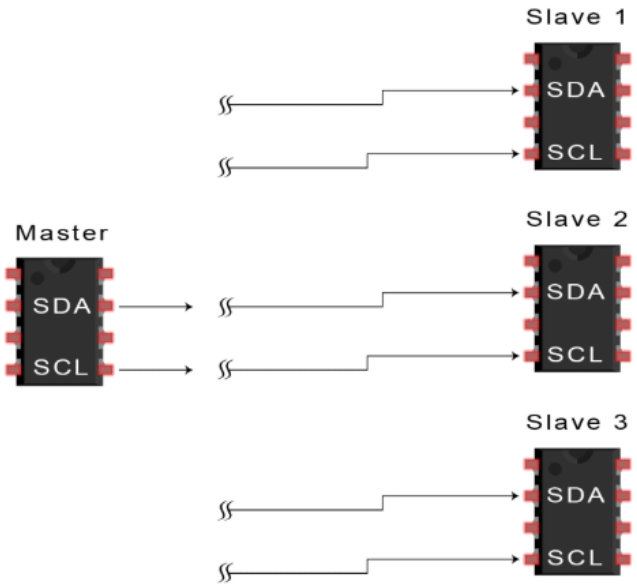

Fig 6

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:

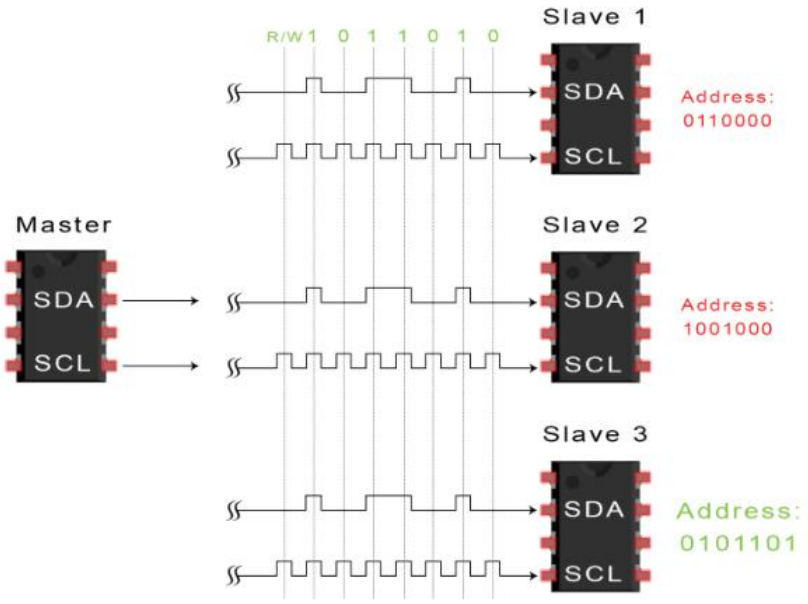

Fig 7

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.

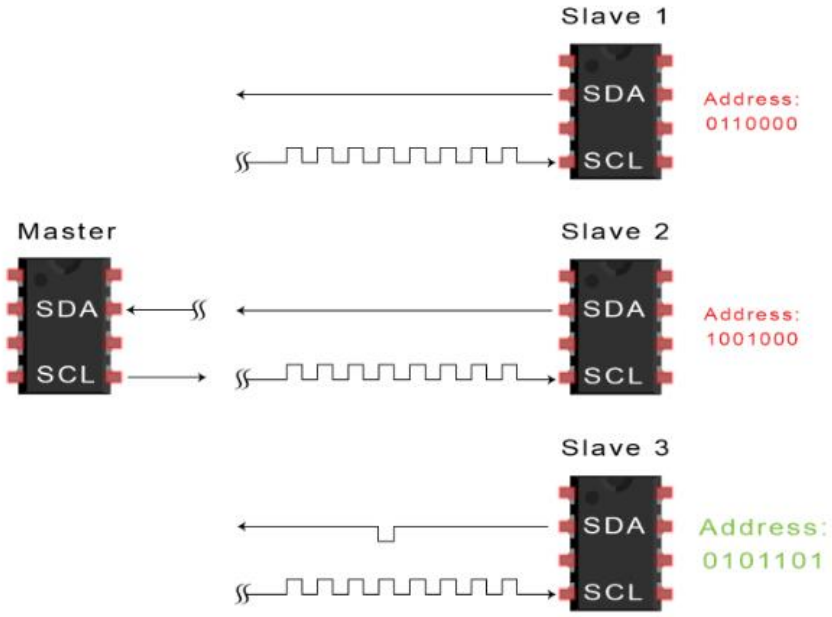

Fig 8

4. The master sends or receives the data frame:

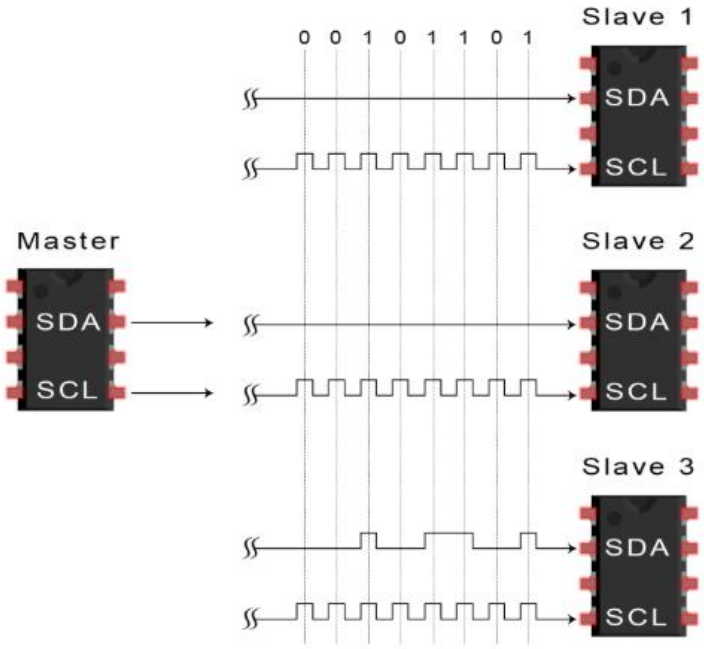

Fig 9

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:

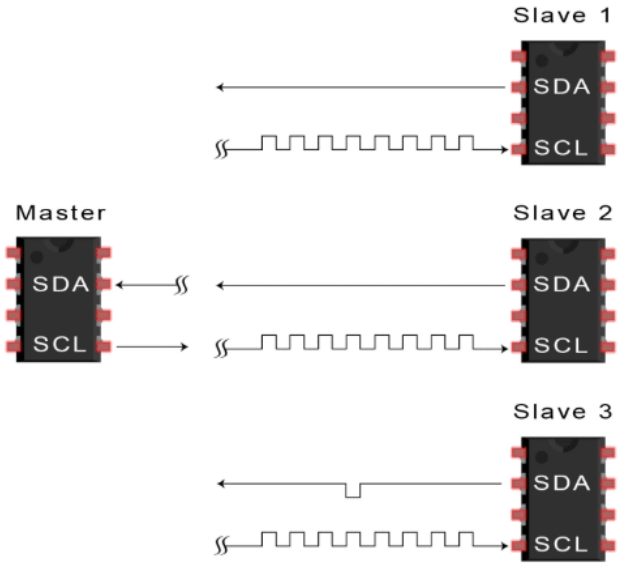

Fig 10

6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:

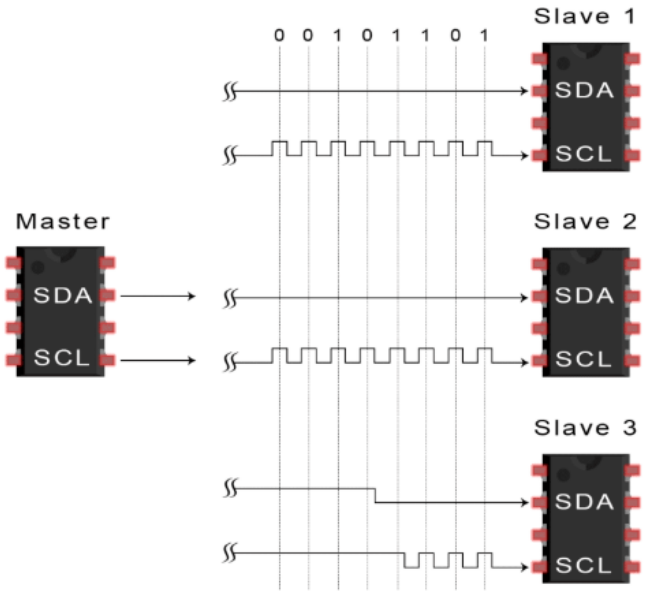

Fig 11

SINGLE MASTER WITH MULTIPLE SLAVES

Because I2C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (2^7) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (2^{10}) 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:

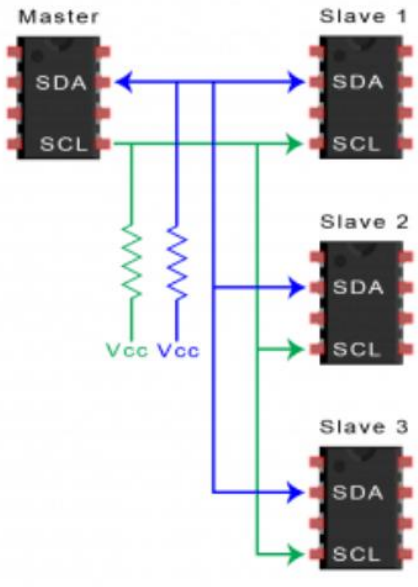

Fig 12

MULTIPLE MASTERS WITH MULTIPLE SLAVES

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:

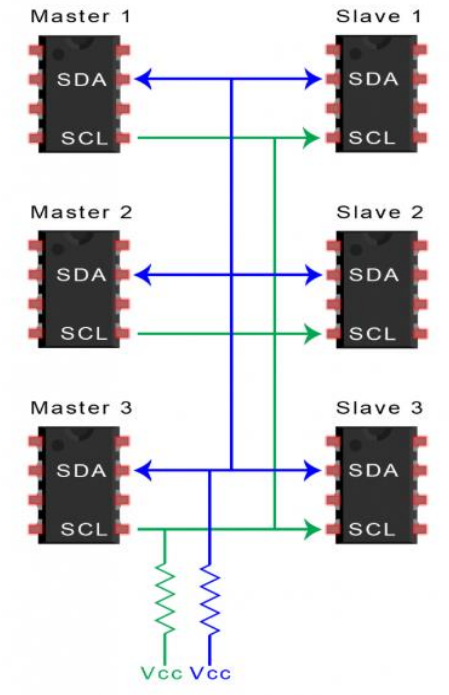

Fig 13

Arduino I2C Pins

Table that lists the different board form factors and what pins are for I2C.

Form factor	SDA	SCL
UNO	SDA/A4	SCL/A5
Nano	A4	A5
Mega	D20	D21

ADVANTAGES 0F I2C

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

Limitations of I2C

- Slowe data transfer rate.
- The size of the data frame is limited to 8 bits.
- More complicated hardware needed to implement.