



VACATION TASKS

SET-1

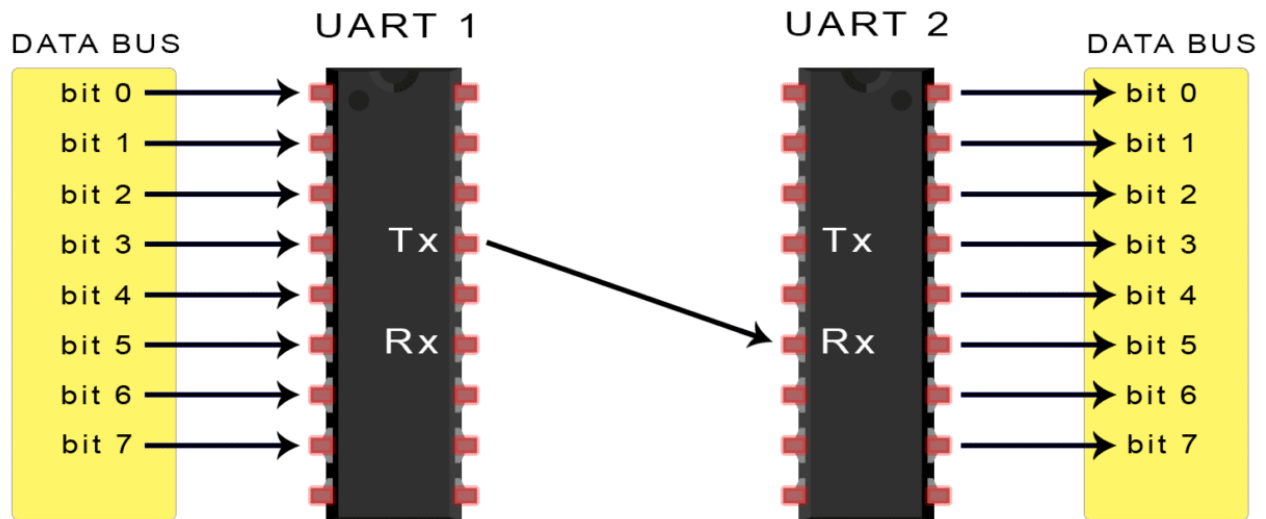
Types of Communication Protocols

UART and I2C

-

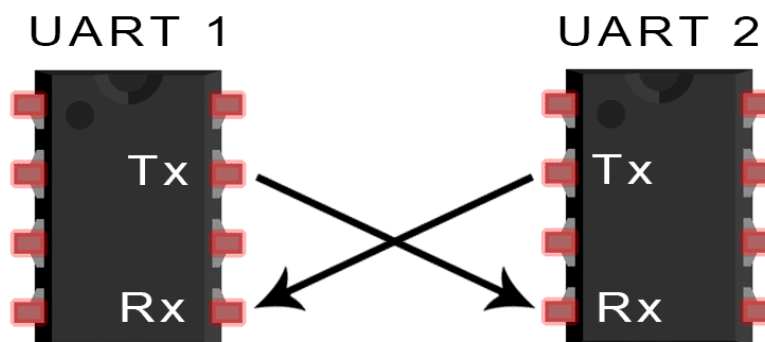
Tanush Biju

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.

In UART communication, two UART's 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 UART's. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:



UART's 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.

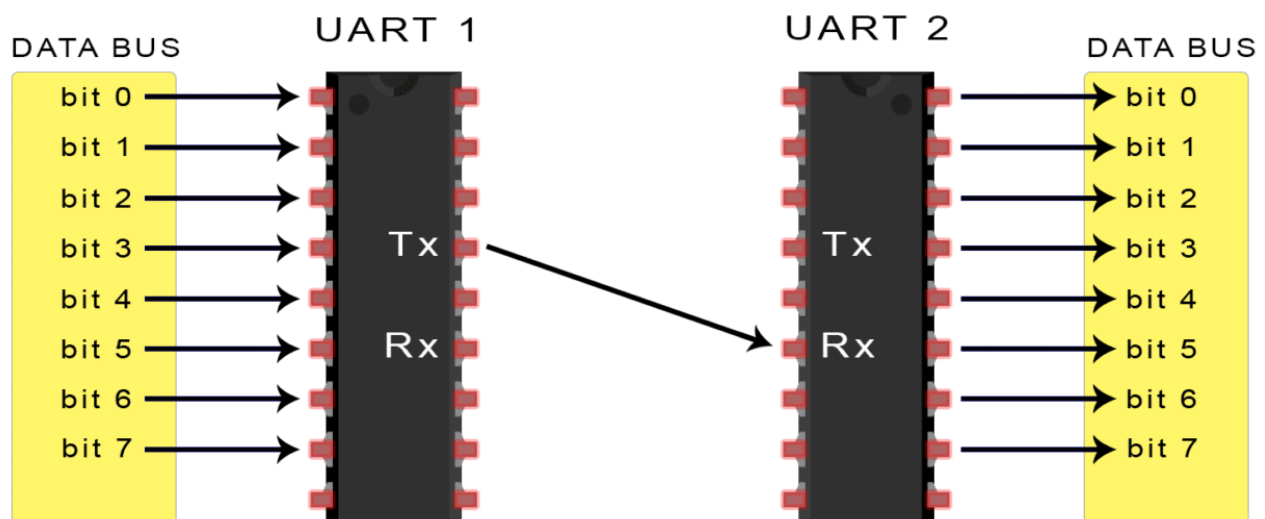
When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UART's must operate at about the same baud rate. The baud rate between the transmitting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

Both UART's must also must be configured to transmit and receive the same data packet structure.

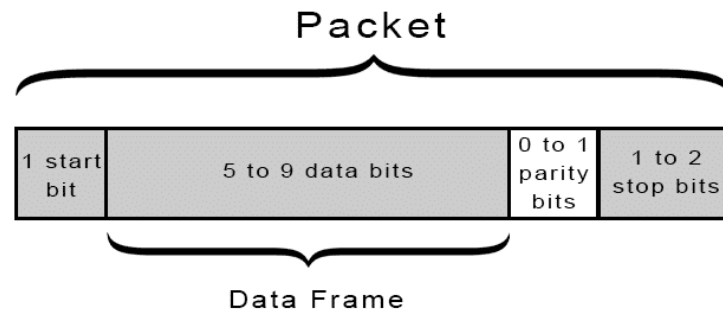
Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous?	Asynchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	1

Working of UART

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. 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. 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:



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:



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 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.

DATA FRAME

The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits long if a parity bit is used. 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.

PARITY

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. 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.

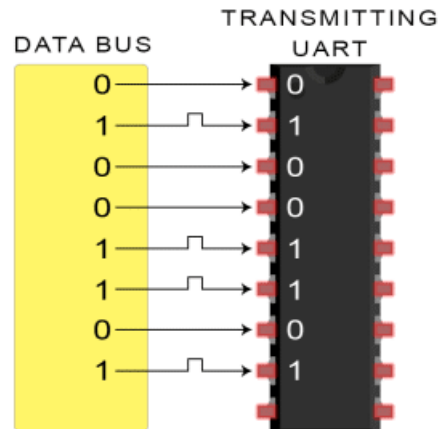
If the parity bit is a 0 (even parity), the 1 bits in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bits in the data frame should total to an odd number. When the parity bit matches the data, the UART knows that the transmission was free of errors. But if 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

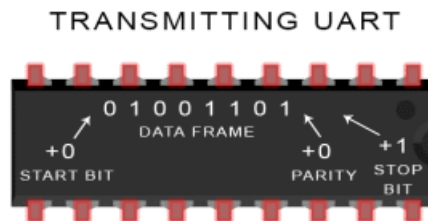
To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

STEPS OF UART TRANSMISSION

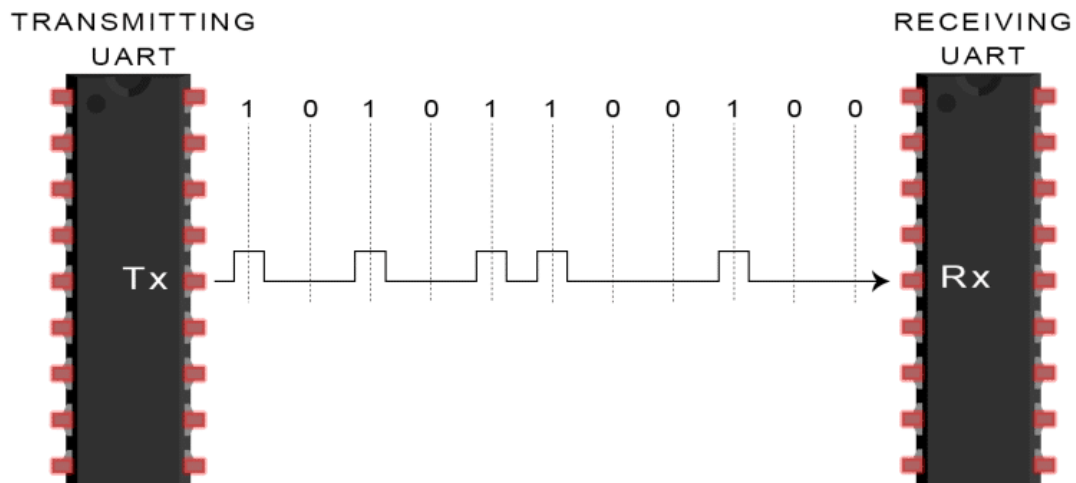
1. The transmitting UART receives data in parallel from the data bus:



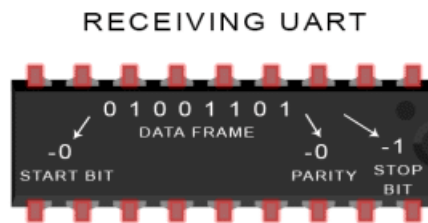
2. The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame:



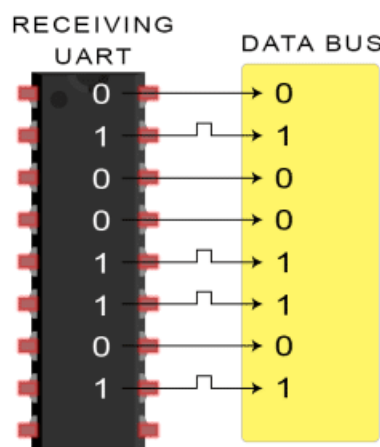
3. The entire packet is sent serially from the transmitting UART to the receiving UART. The receiving UART samples the data line at the pre-configured baud rate:



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



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



ADVANTAGES AND DISADVANTAGES OF UART's

No communication protocol is perfect, but UART's are pretty good at what they do. Here are some pros and cons to help you decide whether or not they fit the needs of your project:

ADVANTAGES

Only uses two wires

- No clock signal is necessary
 - Has a parity bit to allow for error checking
 - The structure of the data packet can be changed as long as both sides are set up for it
- Well documented and widely used method

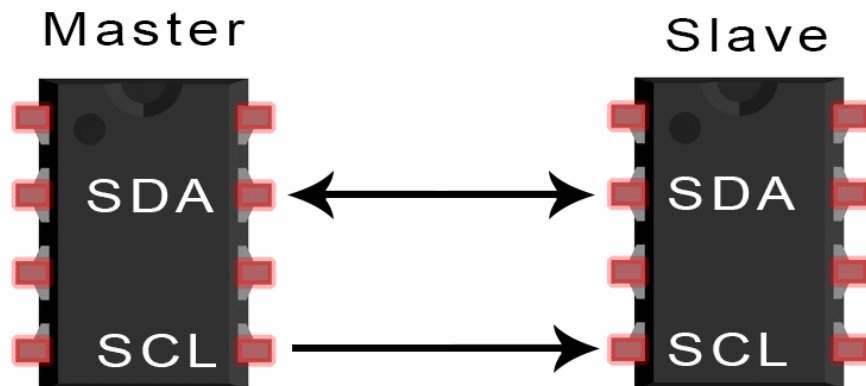
DISADVANTAGES

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

I2C Communication

Inter-Integrated Circuit, or I2C combines the best features of SPI and UART's. 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:



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.

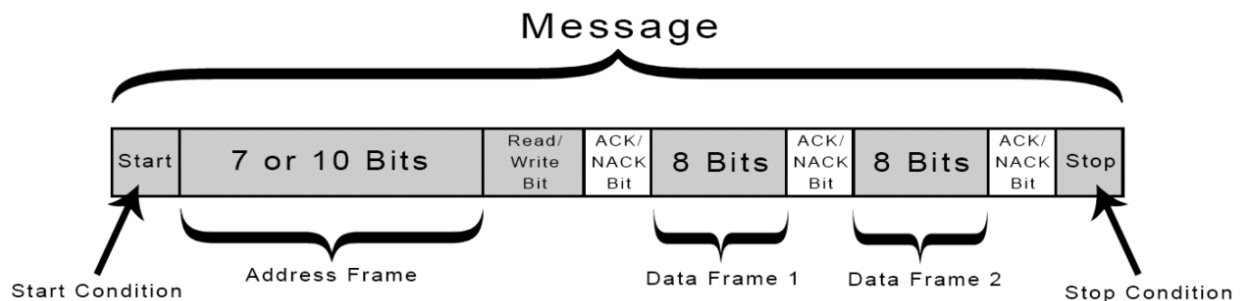
I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

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 Used	2
Maximum Speed	Standard mode= 100 kbps
	Fast mode= 400 kbps
	High speed mode= 3.4 Mbps
	Ultra fast mode= 5 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	Unlimited
Max # of Slaves	1008

Working of I2C

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:



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

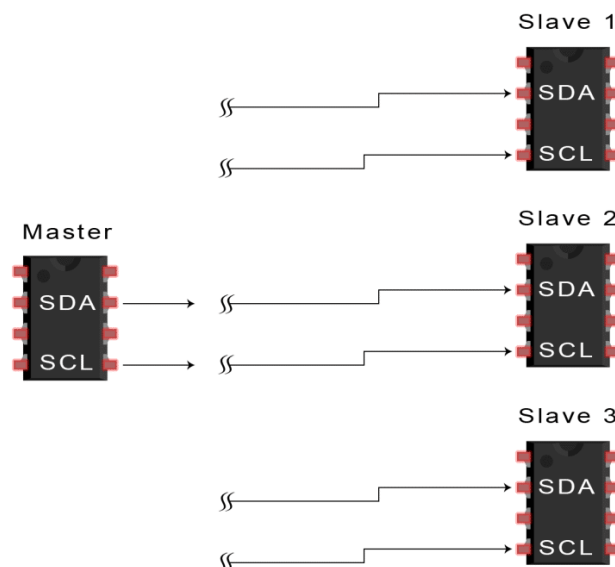
After the master detects the ACK bit from the slave, the first data frame is ready to be sent.

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.

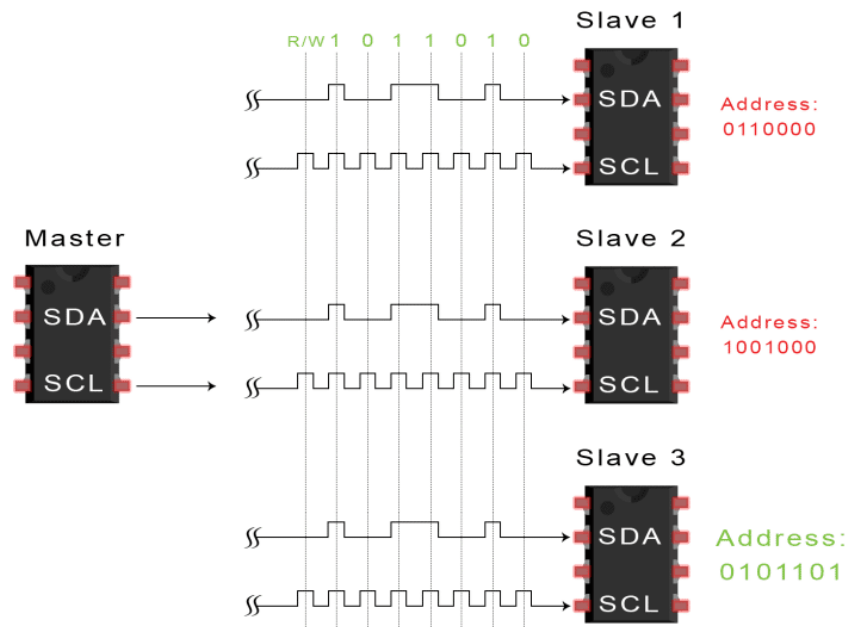
After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission. The stop condition is a voltage transition from low to high on the SDA line after a low to high transition on the SCL line, with the SCL line remaining high.

STEPS OF I2C DATA TRANSMISSION

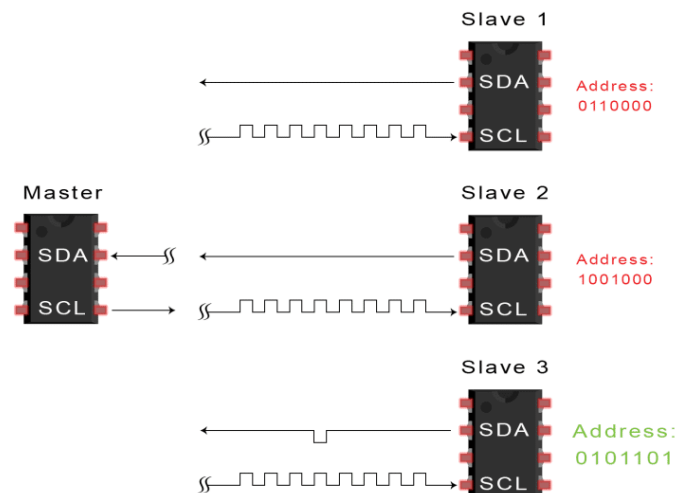
1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low:



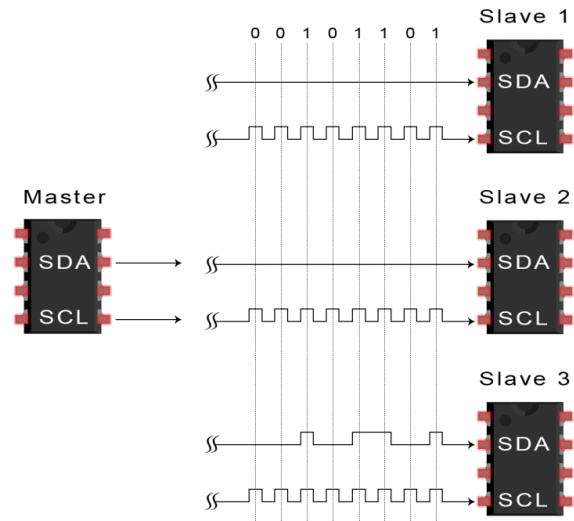
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:



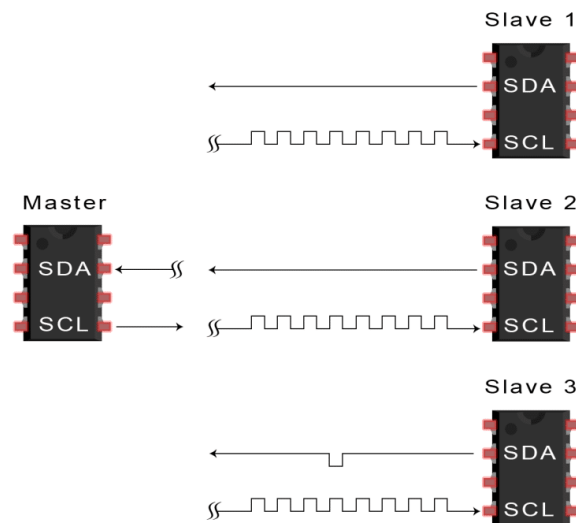
3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.



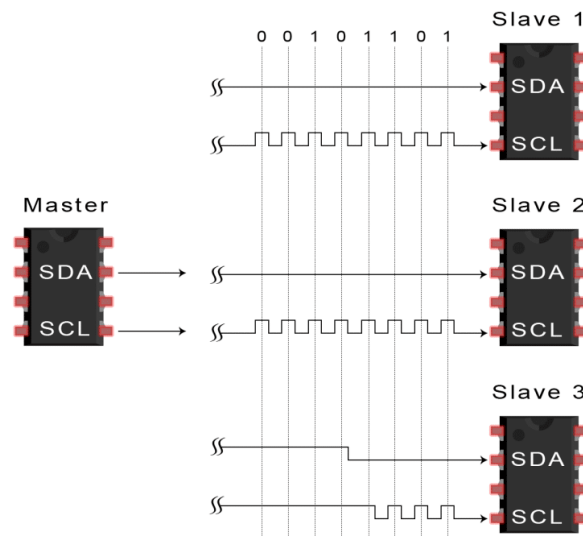
4. The master sends or receives the data frame:



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:

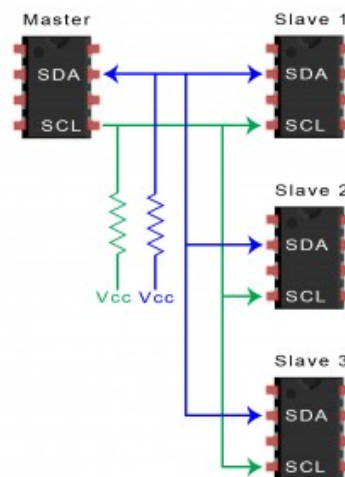


6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:



SINGLE MASTER WITH MULTIPLE SLAVES

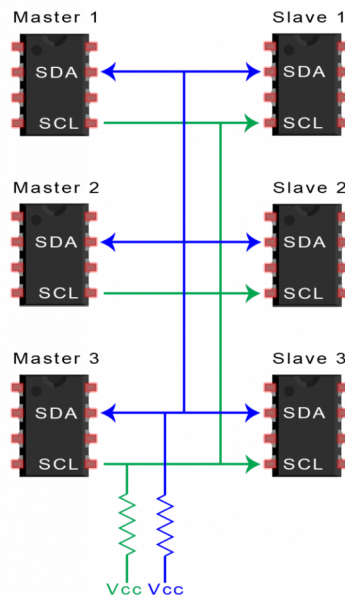
Because I2C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (27) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (210) unique addresses. To connect multiple slaves to a single master, wire them like this, with 4.7K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



MULTIPLE MASTERS 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:



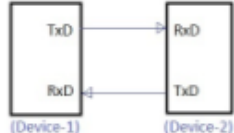
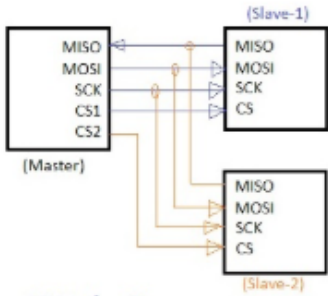
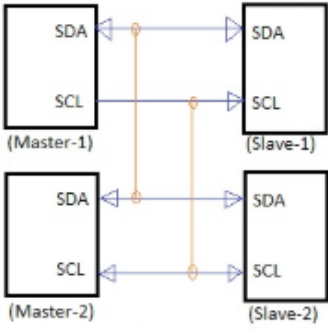
ADVANTAGES AND DISADVANTAGES OF I2C's


ADVANTAGES

- 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 UART's
- Well known and widely used protocol

DISADVANTAGES

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

Features	UART	SPI	I2C
Full Form	Universal Asynchronous Receiver/Transmitter	Serial Peripheral Interface	Inter-Integrated Circuit
Interface Diagram	 <p>UART Interface Diagram</p>	 <p>SPI Interface Diagram</p>	 <p>I2C Interface Diagram</p>
Pin Designations	TxD: Transmit Data RxD: Receive Data	SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select	SDA: Serial Data SCL: Serial Clock
Data rate	As this is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.	Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps	I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps.
Distance	Lower about 50 feet	highest	Higher
Type of communication	Asynchronous	Synchronous	Synchronous
Number of masters	Not Application	One	One or more than One
Clock	No Common Clock signal is used. Both the devices will use there independent clocks.	There is one common serial clock signal between master and slave devices.	There is common clock signal between multiple masters and multiple slaves.
Hardware complexity	lesser	less	more

Protocol	For 8 bits of data one start bit and one stop bit is used.	Each company or manufacturers have got their own specific protocols to communicate with peripherals. Hence one needs to read datasheet to know read/write protocol for SPI communication to be established. For example we would like SPI communication between microcontroller and EPROM. Here one need to go through read/write operational diagram in the EPROM data sheet.	It uses start and stop bits. It uses ACK bit for each 8 bits of data which indicates whether data has been received or not. Figure depicts the data communication protocol. 
Software addressing	As this is one to one connection between two devices, addressing is not needed.	Slave select lines are used to address any particular slave connected with the master. There will be 'n' slave select lines on master device for 'n' slaves.	There will be multiple slaves and multiple masters and all masters can communicate with all the slaves. Upto 27 slave devices can be connected/addressed in the I2C interface circuit.
Advantages	<ul style="list-style-type: none"> • It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface. 	<ul style="list-style-type: none"> • It is simple protocol and hence so not require processing overheads. • Supports full duplex communication. • Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. • SPI uses push-pull and hence higher data rates and longer ranges are possible. • SPI uses less power compare to I2C 	<ul style="list-style-type: none"> • Due to open collector design, limited slew rates can be achieved. • More than one masters can be used in the electronic circuit design. • Needs fewer i.e. only 2 wires for communication. • I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. • It uses open collector bus concept. Hence there is bus voltage flexibility on the interface bus. • Uses flow control.
Disadvantages	<ul style="list-style-type: none"> • They are suitable for communication between only two devices. • It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled. 	<ul style="list-style-type: none"> • As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is concerned. • Master and slave relationship can not be changed as usually done in I2C interface. • No flow control available in SPI. 	<ul style="list-style-type: none"> • Increases complexity of the circuit when number of slaves and masters increases. • I2C interface is half duplex. • Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/microprocessor.