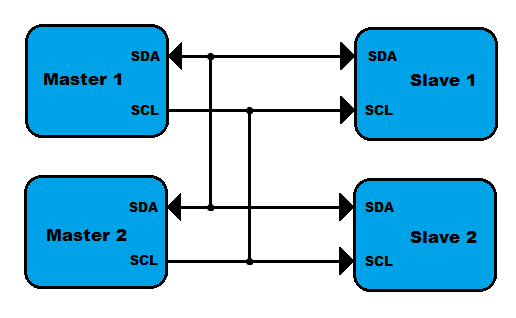
**I2C PROTOCOL AND UART PROTOCOLS**

**I2C PROTOCOL**

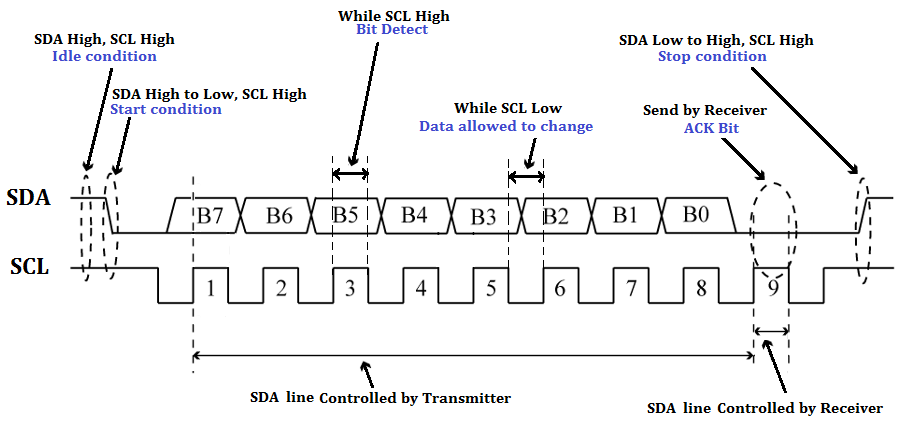
UART, SPI, I2C, USB and Ethernet are some of the protocols widely used in embedded systems for serial data communication. Here we are bound to I2C Protocol.

                 The **I**nter-**i**ntegrated **C**ircuit (I2C) Protocol was originally developed in 1982 by Philips(now known as NXP) used to connect multiple devices with only two wire. Its is also called two wire interface protocol.



* I2C requires only two line(two wire), but those two wires can support up to 1008 slave devices.
* Unlike SPI, I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can’t talk to each other over the bus and must take turns using the bus lines).
* Data rates of I2C devices can communicate at the rate of 100kHz to 400kHz.

**Single Byte Transfer**



* **Idle condition**: As you can see in above figure, at the initial condition SDA and SCL both High because of both line pulled High with pull-up resister.
* **Start condition**: Data transfer is start when SDA line pulled down mean High to Low when SCL line is High.
* **Bit Detect**: Slave can understand bit reception when it receives single bit ‘1’ or ‘0’ when SCL line is High.
* **Data allowed to change**: Transition of bit from 1 to 0 or 0 to 1 is only allowed when SCL line is Low as seen in pick.
* **ACK Bit**: Transmitter must release the SDA line after transmitting 8 bit to allow the Receiver to pull SDA line Low to acknowledgement the previous 8 bit data reception.
* **Stop condition**: is only understood when SDA line goes Low to High while SCL line is High.

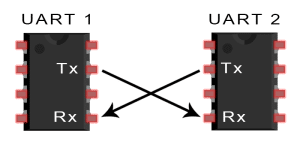
**UART**

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.

**INTRODUCTION TO UART COMMUNICATION**

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:

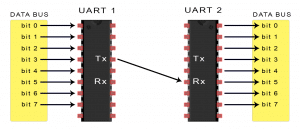
[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Basic-Connection-Diagram.png)

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.

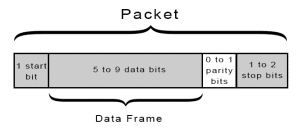
When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate.*Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps).Both UARTs must operate at about the same baud rate.

**HOW UART WORKS**

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:

[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Data-Transmission-Diagram.png)

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:

[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Packet-Frame-and-Bits-2.png)

**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**

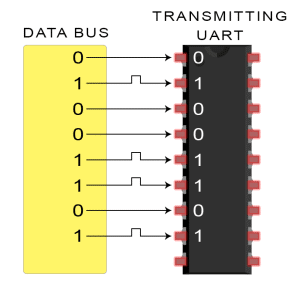
Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long distance data transfers. After the receiving UART reads the data frame, 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**

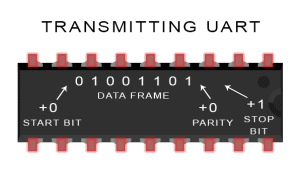
To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

**STEPS OF UART TRANSMISSION**

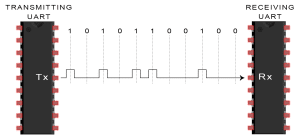
1. The transmitting UART receives data in parallel from the data bus:

[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Data-Transmission-Diagram-UART-Gets-Byte-from-Data-Bus.png)

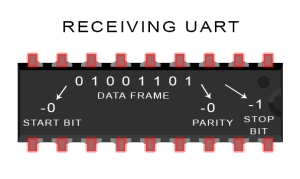
2. The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame:

[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Data-Transmission-Diagram-UART-Adds-Start-Parity-ad-Stop-Bits-2.png)

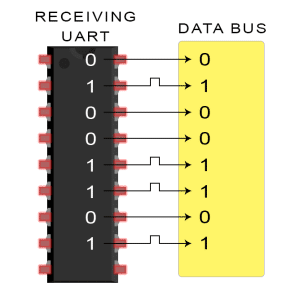
3. The entire packet is sent serially from the transmitting UART to the receiving UART. The receiving UART samples the data line at the pre-configured baud rate:

[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Data-Transmission-Diagram-Transmitting-UART-Sends-Data-Packet-Serially-to-Receiving-UART.png)

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

[](http://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-UART-Data-Transmission-Diagram-UART-Removes-Start-Parity-and-Stop-Bits-2.png)

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

[](http://www.circuitbasics.com/wp-content/uploads/2016/02/Introduction-to-UART-Data-Transmission-Diagram-Receiving-UART-Sends-Byte-to-Data-Bus-2.png)

**ADVANTAGES AND DISADVANTAGES OF UARTS**

No communication protocol is perfect, but UARTs 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

**DIFFERENCE BETWEEN UART AND I2C PROTOCOL**

* In UART, separate clock line is not used with data transmission so both device should agree on same Baud-rate. If differences occur between Baud-rate on either end will cause garbled data while in I2C separate clock line is used.
* UART uses hardware overheads like start bit, stop bit, partiy bit or CRC bit in each frame of transmission which will lead to increase in transmission time while I2C use only acknowledgement bit(ACK) or no acknowledgement bit.
* Another drawback of UART is that they are suited to communications between two, and only two, devices while I2C can used with multiple devices and different modes like multi-master or single master, etc.
* Data transfer speed is also an issue for UART, they are limited to some extent and maximum baud-rate is around 230400 bps.