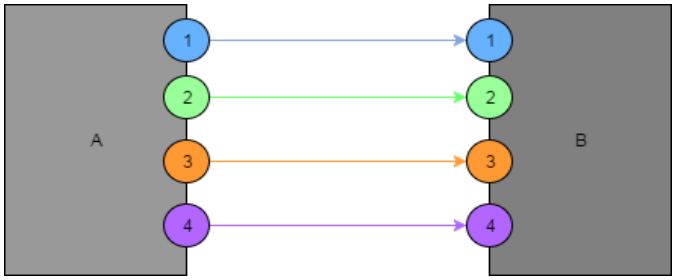
Serial Communications

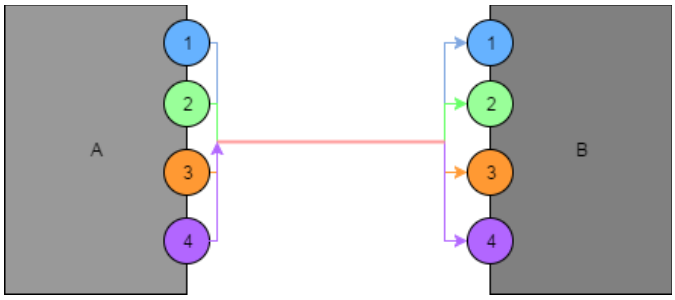
Introduction

Lets say you need to get some collection of data from point device A to device B by using some kind using some system of wires connecting the two devices. There are a couple of different approaches you could use to implement this tranfer of data.

Parallel – One approach is to create a parallel bus, in which many individual wires connect the two device, and down each of these wires we can transfer some amount of data simultaneously. This approach has the advantage of sending the data more quickly and easily; but has the disadvantage of requiring far more wires.

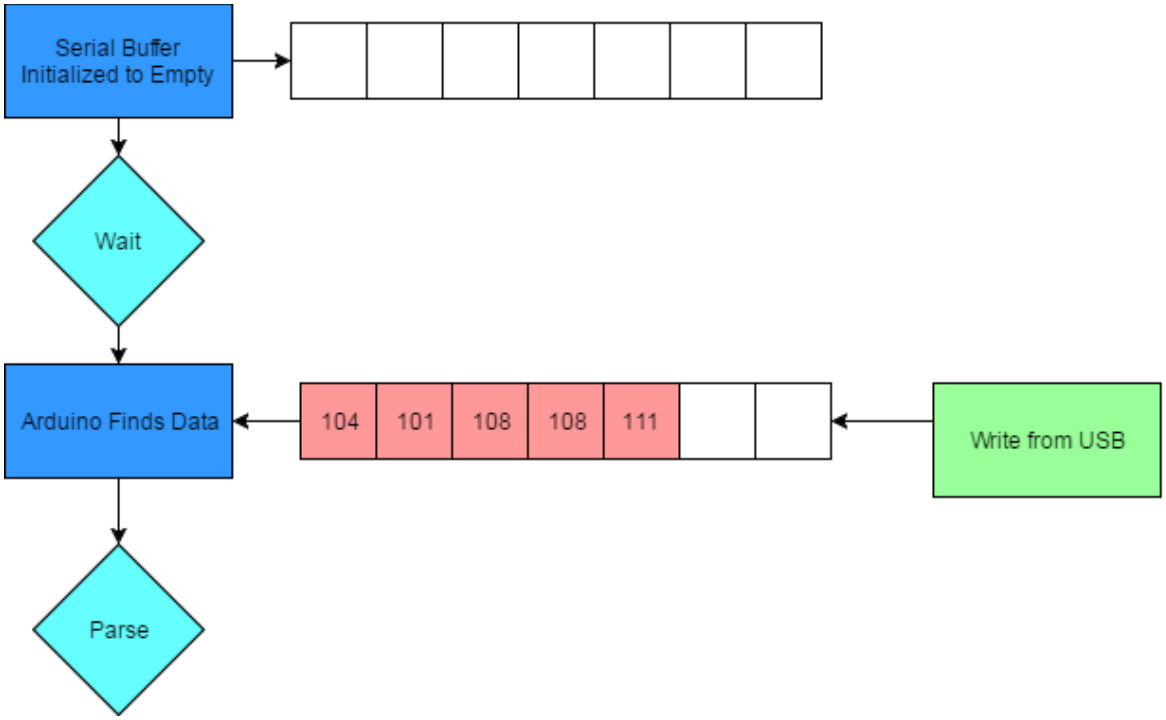


Serial – Another approch is to create a serial wire in which the data is put into a list and sent, one element at a time, down a single wire. This has the advantage of requiring only a single wire connecting the devices; but has the disadvantage of being slower and requiring some processing on each end to serialize then deserialize the data.



Technically speaking (the best kind of speaking), all the digital interfaces we will examine in this course (Serial, SPI, I^2C) are examples of serial interfaces. However, when we use the term serial interface, we are usually referring to the type of serial interface that is implemented by USB and the Arduino, and that's an asyncronous hardware serial interface. Asyncronous: because the two devices are not syncronized by a shared clock line. Hardware: because the Arduino's UART chip handles the communication so we don't need to run software on the Arduino's processor to implement the interface.

Arduino Serial Interface



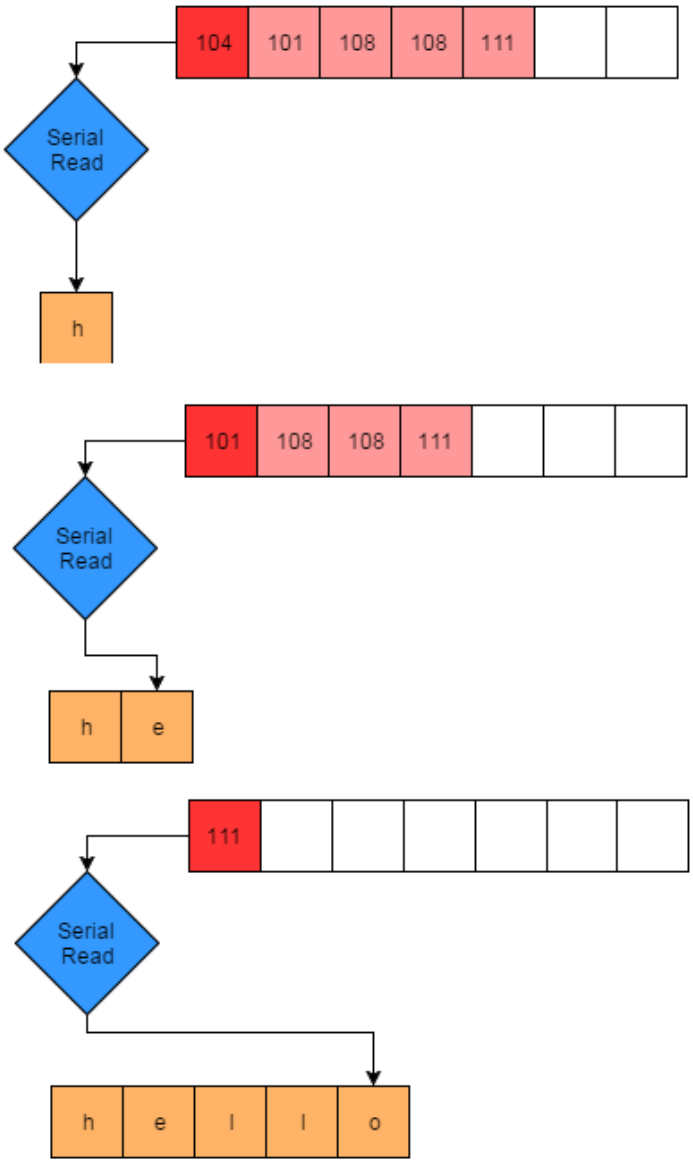
First, we initialize the serial interface using the `Serial.begin()` word. This creates an empty buffer ready to accept data form the serial connection.

Next, the Ardiuno enters into a loop where it is constantly polling the serial buffer to see if there are any bytes written into it. This is done in the `loop` function where we are calling

```
if(Serial.available( ) > 0)
```

This is asking for the number of bytes written into the serial buffer, and continuing if that number is greater than zero.

At some point, the device on the other side of the serial connection, in our case the computers running the Arduino Serial Monitor, will send some bytes of data which are written into the buffer. The Arduino detects that there are available bytes in the buffer.



Finally, the Arduino loops over all the bytes in the buffer. It reads and clips the first item in the buffer using the `Serial.Read( )` funciton.

It casts this byte as a `char` and then adds that character to the end of a list of characters, commonly known as a String, to contruct the message.

When the process is finished, the Serial buffer is empty and ready for new data to be written to it.