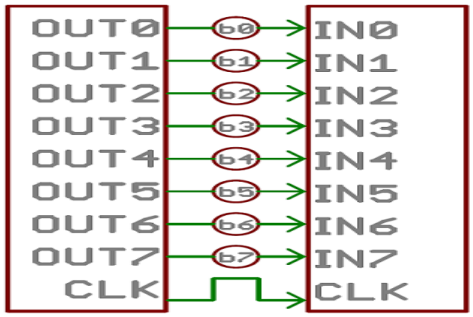
# SERIAL COMMUNICATION

## Introduction

Embedded electronics is all about interlinking circuits (processors or other integrated circuits) to create a symbiotic system. In order for those individual circuits to swap their information, they must share a common communication protocol. Hundreds of communication protocols have been defined to achieve this data exchange, and, in general, each can be separated into one of two categories: parallel or serial.

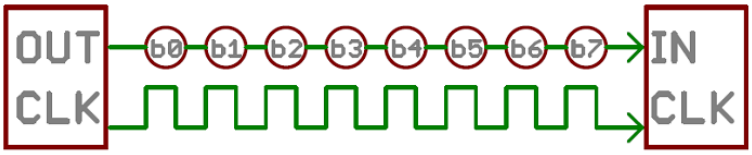
### Parallel vs. Serial

Parallel interfaces transfer multiple bits at the same time. They usually require **buses** of data - transmitting across eight, sixteen, or more wires. Data is transferred in huge, crashing waves of 1's and 0's.




*An 8-bit data bus, controlled by a clock, transmitting a byte every clock pulse. 9 wires are used.*

Serial interfaces stream their data, one single bit at a time. These interfaces can operate on as little as one wire, usually never more than four.




*Example of a serial interface, transmitting one bit every clock pulse. Just 2 wires required!*

Think of the two interfaces as a stream of cars: a parallel interface would be the 8+ lane mega-highway, while a serial interface is more like a two-lane rural country road. Over a set amount of time, the mega-highway potentially gets more people to their destinations, but that rural two-laner serves its purpose and costs a fraction of the funds to build.

Parallel communication certainly has its benefits. It's fast, straightforward, and relatively easy to implement. But it requires many more input/output (I/O) lines. If you've ever had to move a project from a basic Arduino Uno to a Mega, you know that the I/O lines on a microprocessor can be precious and few. So, we often opt for serial communication, sacrificing potential speed for pin real estate.

## Asynchronous Serial

Over the years, dozens of serial protocols have been crafted to meet particular needs of embedded systems. USB (universal *serial* bus), and Ethernet, are a couple of the more well-known computing serial interfaces. Other very common serial interfaces include SPI, $I^2C$, and the serial standard we're here to talk about today. Each of these serial interfaces can be sorted into one of two groups: synchronous or asynchronous.

A synchronous serial interface always pairs its data line(s) with a clock signal, so all devices on a synchronous serial bus share a common clock. This makes for a more straightforward, often faster serial transfer, but it also requires at least one extra wire between communicating devices. Examples of synchronous interfaces include SPI, and $I^2C$.

*Asynchronous* means that data is transferred **without support from an external clock signal**. This transmission method is perfect for minimizing the required wires and I/O pins, but it does mean we need to put some extra effort into reliably transferring and receiving data. The serial protocol is the most common form of asynchronous transfers. It is so common, in fact, that when most folks say “serial” they’re talking about this protocol. The clock-less serial protocol is widely used in embedded electronics. If you’re looking to add a GPS module, Bluetooth, XBee’s, serial LCDs, or many other external devices to your project, you’ll probably need to whip out some serial-fu.

Source: https://learn.sparkfun.com/tutorials/serial-communication