Lecture 3.07
Digital Communication Protocols

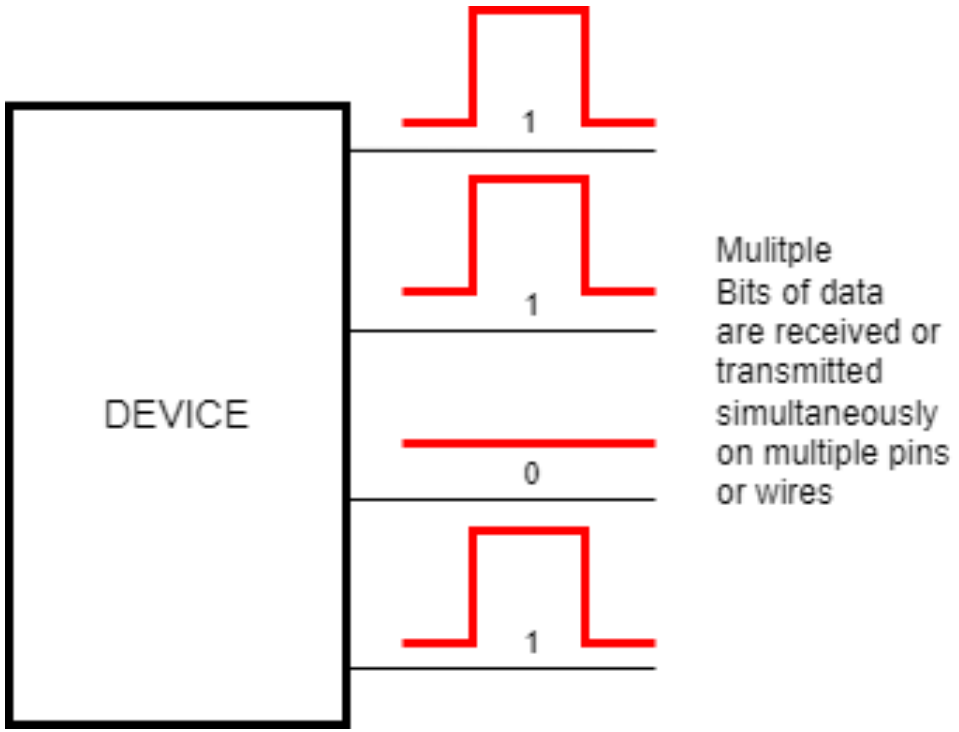
Digital Communication

- We have seen the basics of how to send a single bit of data using a single digital pin. But we will need to use multiple pins and time-varying signals to send more complex data to and from the microcontroller
- The format of the signals is called a communication protocol

- First, we want to define some terms we will use describe the protocols

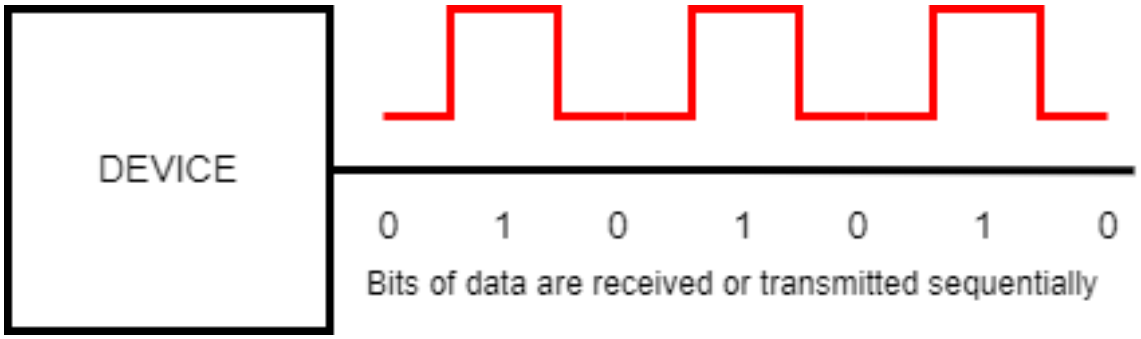
Parallel Interface

- Multiple bits of data are transmitted or received at the same time (in **Parallel**)
- The group of data lines is called a **bus**
- Width of data bus is usually 1 or byte-wide (8 data bits)
- Became less common as speeds increased



Serial Interface

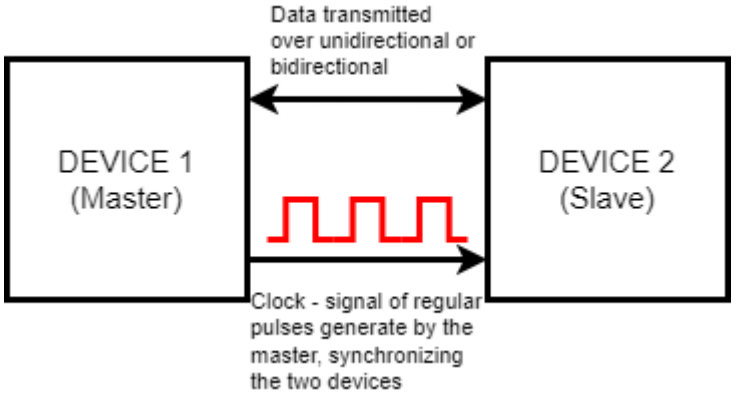
- Data bits arrive one after another on a single interface (in **Series**)
- All of the communications protocols we will discuss are serial
- The **Serial** library is an example of **A** serial communication protocol it is not the only one
  - Outside of this slide, when we use serial, we usually will mean the specific **Serial** library



Synchronous vs Asynchronous

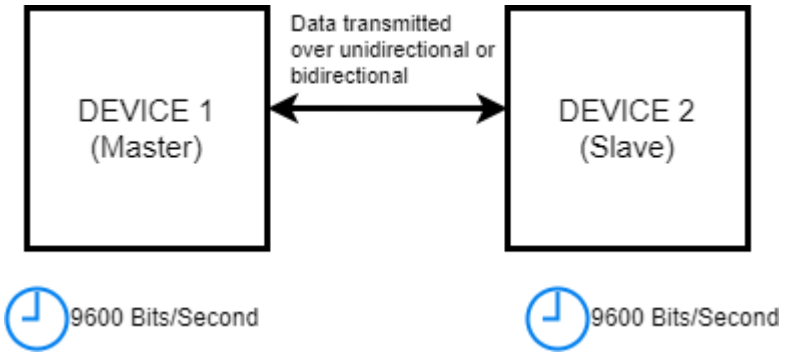
Synchronous Communication

- One device(called the master) transmits a clock signal to one or more (slave) devices
- In time with the clock signal, devices either read input pins and set the value of output pins
- Trigger on the rising or falling edge of the clock signal



Asynchronous Communication

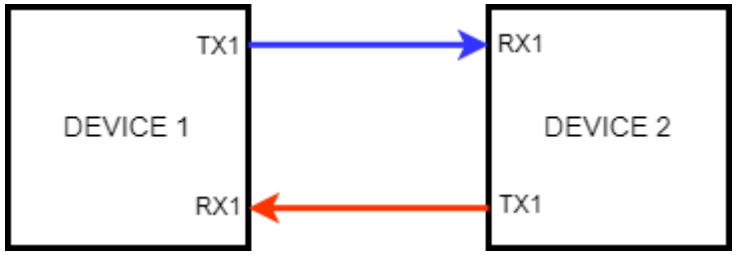
- To start a communication, some initiating signal is sent (can be as simple as going from high to low)
- Each device then checks the data at some interval measured by an internal timer
  - Each device must agree on the rate of data transfer and how many bits of data are being sent per transmission



Bidirectional vs Unidirectional

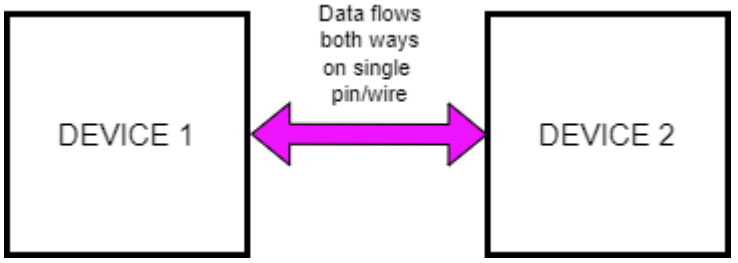
Unidirectional

- Devices have separate pins for each direction of data flow
  - Send data on transmit (**TX**) pins
  - Listen for data on receive (**RX**) pins
- One device's transmit must be connected to another's receive



Bidirectional

- Devices share a common data line to both Transmit and receive data
- This usually means one device needs to coordinate
  - Usually used with a synchronous clock signal since that allows the master device to control data flow



Hardware vs Software Implementation

Software

- You could implement communication with a combination of digitalWrite() and digitalRead() functions
- Any digital pin could be used
- With software, you could come up with your own data transmission protocol
- Has significant disadvantages
  - Very slow
  - Your program will not cannot do any other operations while sending or waiting for data
  - **Hardware** implementation is usually preferred

Hardware

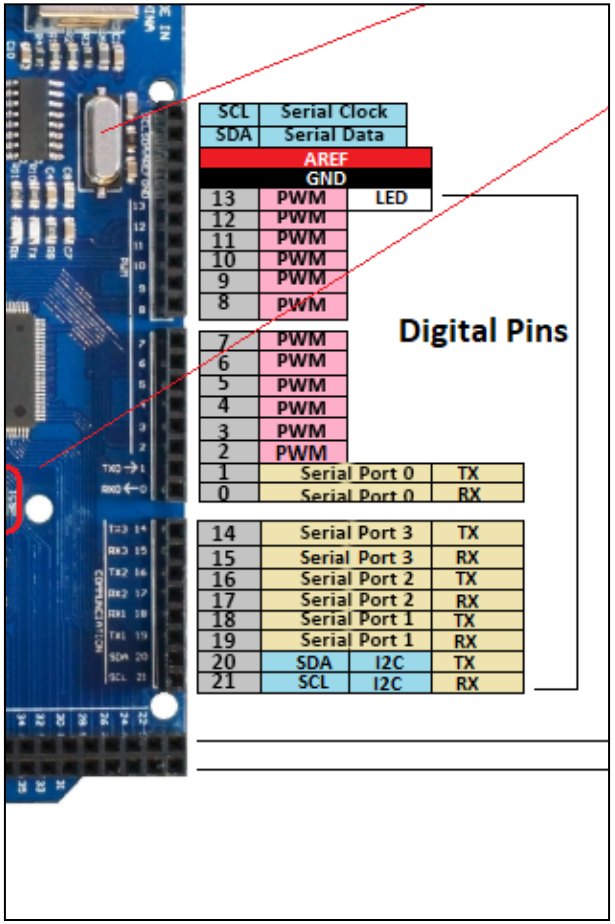
- Most microcontrollers have internal circuitry that will automate the process of sending and receiving the data
  - Have special sections of memory called **buffers**
  - When bits are received, they are automatically copied to the buffer until the software checks the buffer
  - When transmitting can quickly queue the data into the buffer and it is automatically sent at the correct rate until the buffer is empty
- Since it is dependent on hardware, restricted to specific pins on the microcontroller

Arduino Communication Protocols

- The Arduino Mega hardware supports 3 protocols: UART(Serial), SPI, and I2C.
- Each of these are **A** serial protocol and groups the transmitted data into bytes
- UART (**U**niversal **A**synchronous **R**eceiver **T**ransmitter)
  - Unidirectional
  - Asynchronous
- SPI (**S**erial **P**eripheral **I**nterface)
  - Unidirectional
  - Synchronous
- I2C (**I**nter-**I**ntegrated **C**ircuit)
  - Bidirectional
  - Synchronous

Arduino Mega Serial(UART) Pins

- Uses Pairs of TX and RX pins grouped in “Ports”
- The Arduino Mega has four hardware serial ports
  - Serial on pins 0(RX) and 0
  - Serial1 on pins 19 (RX) and 18 (TX)
  - Serial2 on pins 17 (RX) and 16 (TX)
  - Serial3 on pins 15 (RX) and 14 (TX)
- Each serial port can only talk to one other device

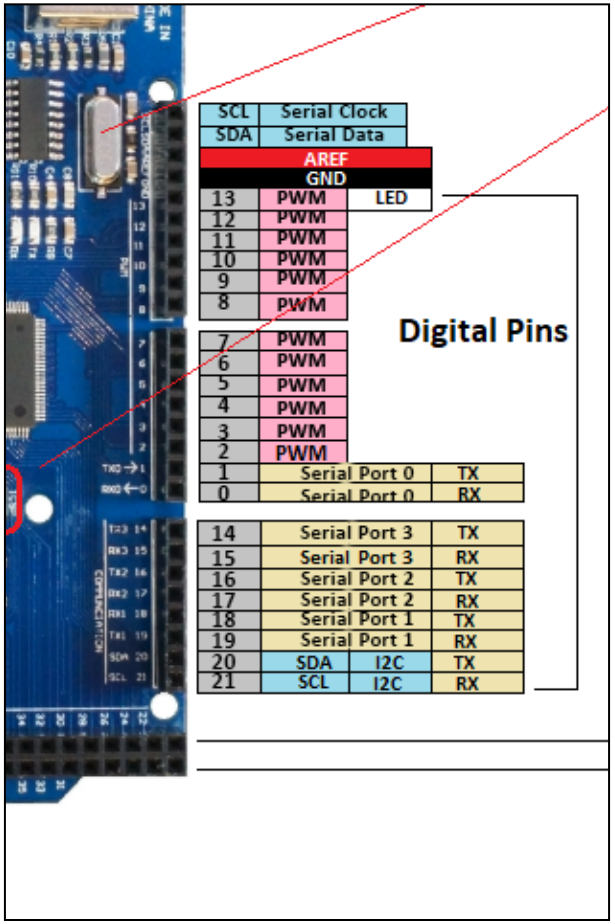


Serial Library for the Arduino Mega

- We have already been using this library to talk to the talk our computer with **Serial.print()**
- The reason we have to set the baud rate is that this is asynchronous
- In addition to the computer, this is how we will talk to the GPS
- In general, bytes are sent one at a time with the data
  - 1 character at a time, for example
  - The print function changes its argument into a series of bytes and sends them to the serial buffer
  - You read data in 1 byte at a time until the buffer is empty
  - Either device can just start sending data
- https://docs.arduino.cc/language-reference/en/functions/communication/serial/

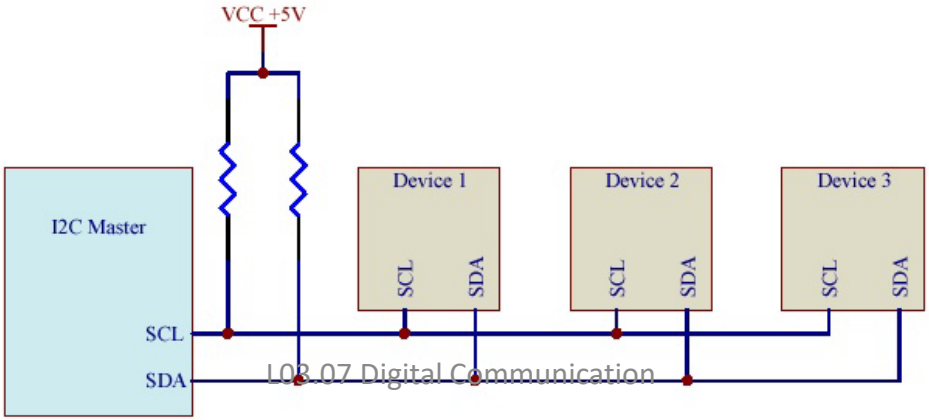
I2C Pins

- I2C uses a single data line on Pin 20 called SDA (Serial Data)
- The Arduino acts as the master device, generating a clock signal on Pin 21 called SCL (Serial Clock)
- The second set of SCL and SDA pins is connected to the first to provide compatibility with other microcontrollers
- The single Master device (Arduino) can be connected to multiple slave devices
  - Each device has must a unique 7-bit code called an address



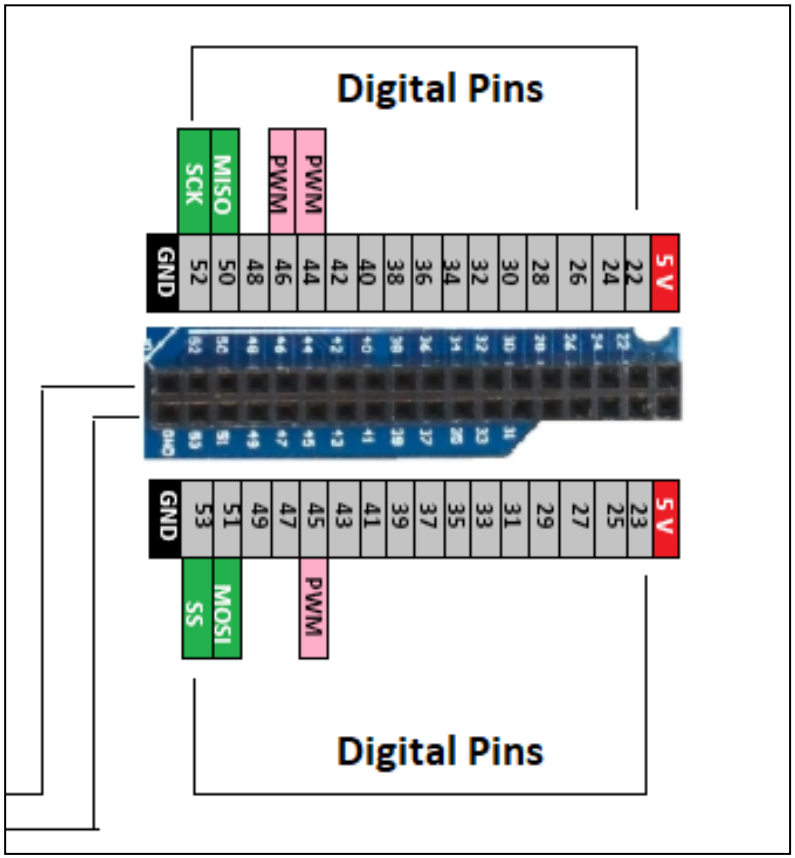
I2C Communications

- The library used for I2C is called Wire https://docs.arduino.cc/language-reference/en/functions/communication/wire/
- The Master device starts the clock and sends a Start Byte
  - The start is the 7-bit address of the target slave device; the final bit indicates that the master wants to **Read** or **Write** data
  - In **Write,** the Master sends bytes of data and then sends a stop signal
  - In **Read**, the Slave device replies with a data byte, then the Master sends a **Stop** or **Continue** signal. If continue, the Slave sends another data byte
- Usually, you use a device library to handle the communication



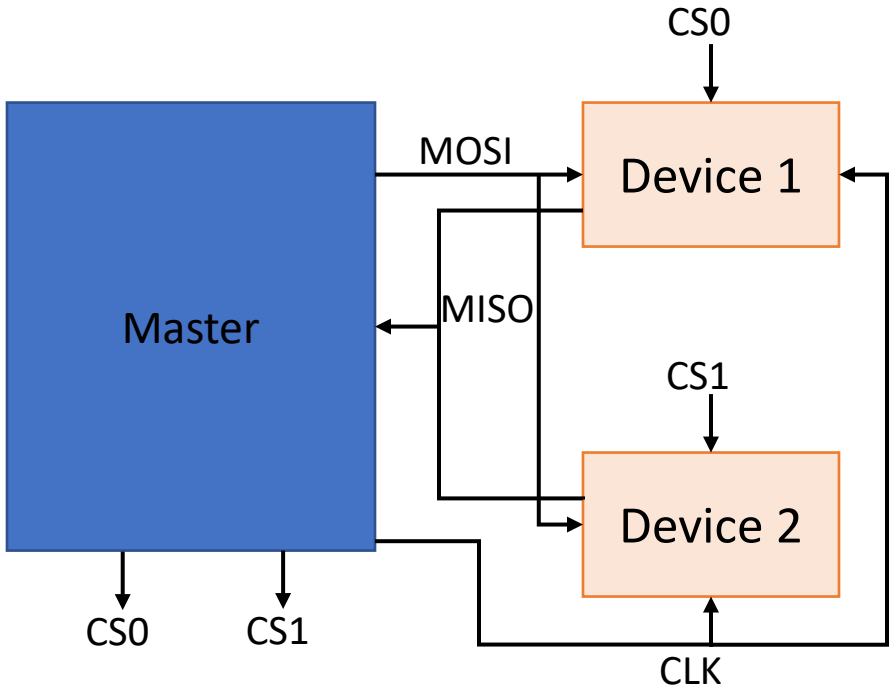
SPI Pins

- Again, the Arduino acts as a Master Device generating a Clock Signal on Pin 52 SCK (Serial Clock)
- SPI uses two unidirectional data lines
  - MISO (Master In Slave Out) on pin 50
  - MOSI (Master Out Slave In) on pin 51
- SPI can also have multiple slave devices
  - Each slave uses a 4th Pin SS (slave select) or CS (chip select) to tell the slave to activate
  - Any digital pin can be used but a different pin must be used for each slave



SPI Programming

- The SPI library is called SPI
  - https://docs.arduino.cc/language-reference/en/functions/communication/SPI/
- SPI communications are much less standardized than Serial or I2C
  - Different clock settings
  - Different data sizes and formats
- We will use SPI for writing data to the SD Card but the SD library will handle the SPI communications



Measure Communication Signal

- You will need to measure these digital signals
- These signals are very fast, with a single transmitted bit only lasting a few microseconds
- This is too fast to be measured by a multimeter
- Instead, we will use a device called an oscilloscope to plot the voltage signal as a function of time
- The scope can capture a very small time interval of voltage data and keep it displayed until cleared
- If you are having trouble with digital communications, your first step will be to use a scope to look at your data
- We will cover oscilloscope operation in an activity.