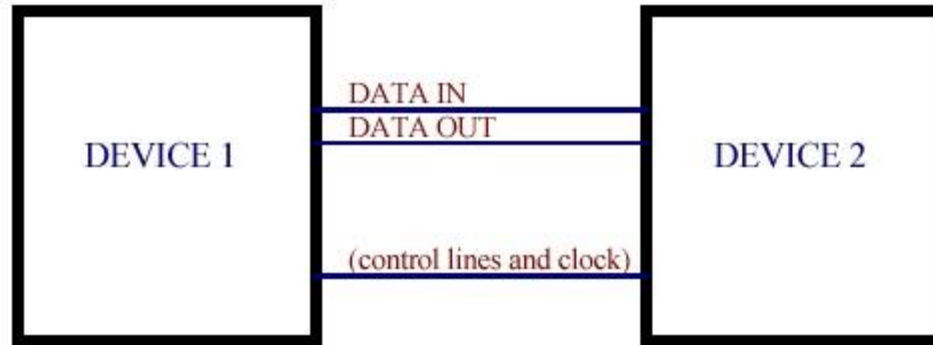
LaACES Student Ballooning Course

Lecture 11.01

Serial Communication Protocols

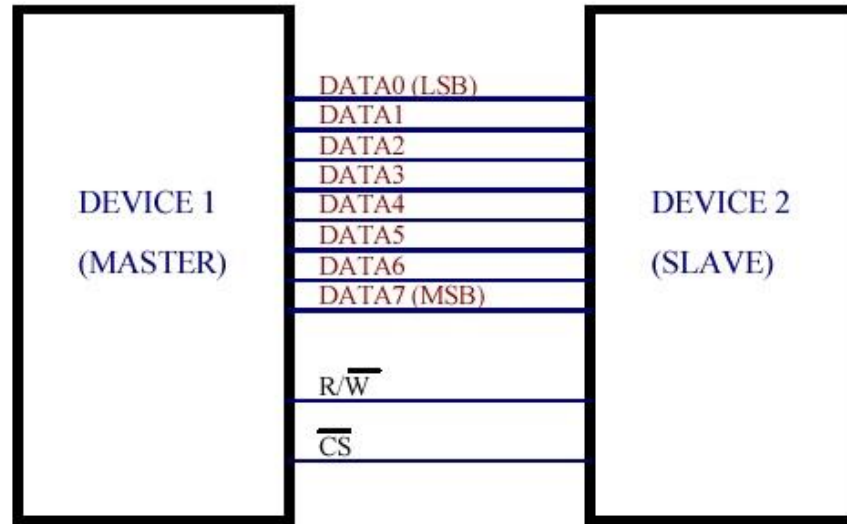
Serial Interface



- Asynchronous serial can be implemented with data lines only.
  - Each device generates its own clock (Baud Rate Generator).
  - Handshaking lines can be used to signal status of devices.
- Synchronous serial interfaces will have a separate Clock line.
  - Clock is generated by a Master device.
- One bit is transferred for each clock cycle.

LaACES
Student
Ballooning
Course

Parallel Interface



- Data lines may be unidirectional or bi-directional.
- Width of data bus is usually byte-wide (8 data bits).
- A full byte of data is transferred on each R/W clock cycle.
- Chip Select (CS) allows multiple devices to share bus.

LaACES Student Ballooning Course

Serial I/O on the Arduino Mega

- Communications can be either via “Hardware” or “Software”
- Hardware means there is dedicated circuitry in the microcontroller for handling signal
- Software means the software must manually read and manipulate the pins
- Because of this Software is significantly slower Hardware, but Software can usually work on any set of pins
- Base Arduino Libraries will use the Hardware pins

LaACES Student Ballooning Course

Hardware Serial

- Mega Supports Hardware UART, SPI, I2C protocols on certain pins
- This means these pins are connected to hardware triggers (called **interrupts**) and memory (**buffers**) that can automatically receive the data
- This means we don't need to be actively "listening" all the time to receive this data
- But the storage space is limited so if data is not copied into memory quickly (compared to speed its being sent) data can be overwritten and lost

Synchronous and Asynchronous Serial Communication

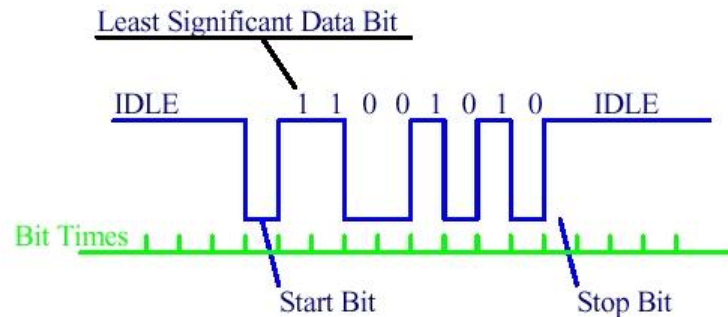
- Synchronous means the devices involved use the same clock-signal when communicating, while asynchronous means the devices use their own individual clocks at set rate (**Baud Rate**).
- Pins 0,1 & 14-19 use a *Universal Asynchronous Receiver/ Transmitter* (UART) for asynchronous communication.
- These are the **Serial** software objects in Arduino IDE
- No two clocks are perfectly matched, so asynchronous communication is slower because extra data must be sent periodically to ensure both devices are in sync.

LaACES Student Ballooning Course

Asynchronous Serial Communication

Serial communication usually involves sending or receiving "characters" using the ASCII code. For example, the character "**S**" is represented by the binary number "**01010011**" or **0x53** in hexadecimal.

An asynchronous transmission of "S" begins with a start bit, followed by 8 data bits and ending with a stop bit. There are numerous options for number data bits, speed and an optional parity bit.



LaACES
Student
Ballooning
Course

Serial Functions for the Arduino Mega

If (Serial)
available()
availableForWrite()
begin()
end()
find()
findUntil()
flush()
parseFloat()
parseInt()
peek()
print()
println()
read()
readBytes()
readBytesUntil()
readString()
readStringUntil()
setTimeout()
write()
serialEvent()

The Arduino website has excellent explanations of how these functions work and a plethora of examples of how to use them.

Arduino Mega Serial Ports

The **Arduino Mega** has four hardware serial ports

- Serial1 on pins 19 (RX) and 18 (TX)
- Serial2 on pins 17 (RX) and 16 (TX)
- Serial3 on pins 15 (RX) and 14 (TX)

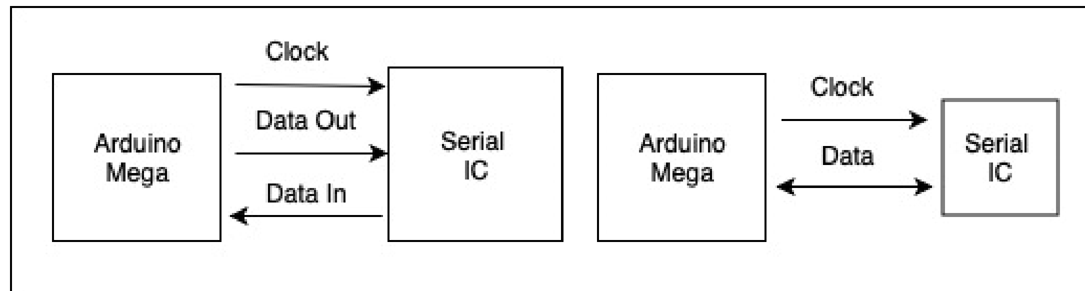
To use these pins to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the Mega's USB-to-serial adaptor.

LaACES Student Ballooning Course

Synchronous Serial I/O

Synchronous serial I/O uses a separate line for a CLOCK signal. The synchronous serial clock, data lines, and Arduino all use TTL logic levels, so no level converters or line drivers/receivers are required.

There are several protocols in use. Some use a bi-directional data line while others use separate Data-In and Data-Out lines. The **Master** generates the **clock** and **initiates** and **controls** data transfer.

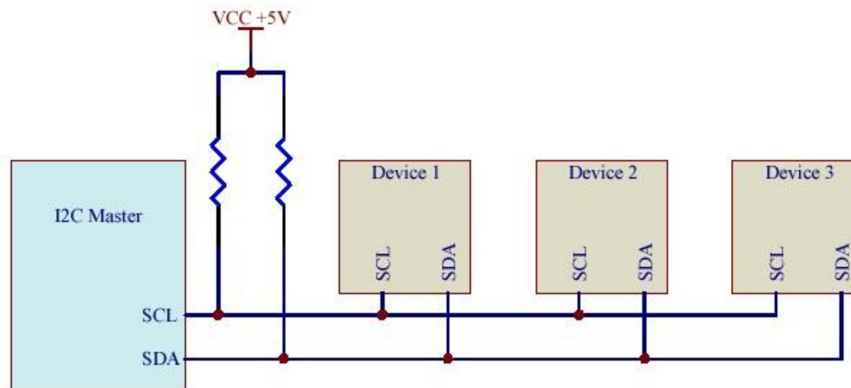


The I2C Bus

•Inter-Integrated-Circuit or I2C (pronounced I-too-see or I-squared-see) is a synchronous serial protocol that uses a bi-directional data line and supports multiple slave devices controlled by a I2C bus master.

•Defined by Phillips Semiconductor and became an industry standard.

•The clock line is called **SCL**, the bidirectional data line **SDA**



I2C Addresses

- The I2C bus master generates SCL and initiates communication with one of the slave devices. Each device has a unique address for device selection.

- Each slave device has a **7 bit address** that uniquely identifies it.

- Some addresses are “hard-wired” into the chip design and one or more pins on the device. These pins can be wired High or Low to select an address that doesn’t conflict with other devices on the I2C bus. Others maybe set via software

- Pull-up resistors are required on both the clock and data lines. Some chips may have internal pull-up resistors on specific pin. Arduino Mega has these

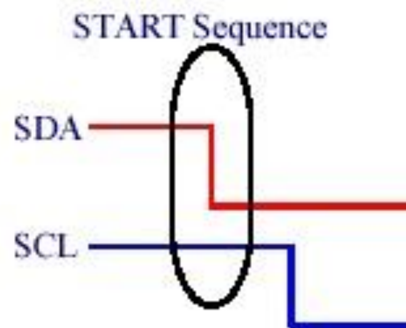
LaACES Student Ballooning Course

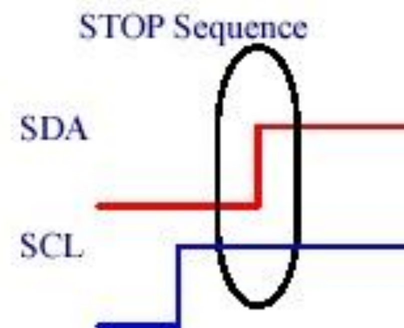
I2C START and STOP

•A specific sequence signal the beginning and end of the transmission

•A **START** sequence begins a bus transmission by transitioning **SDA** from **High to Low** while **SCL** is **High.**

•A **STOP** sequence ends a transmission. The **Stop** sequence occurs when the master brings **SDA** from **Low to High** while **SCL** is **High**.





LaACES Student Ballooning Course

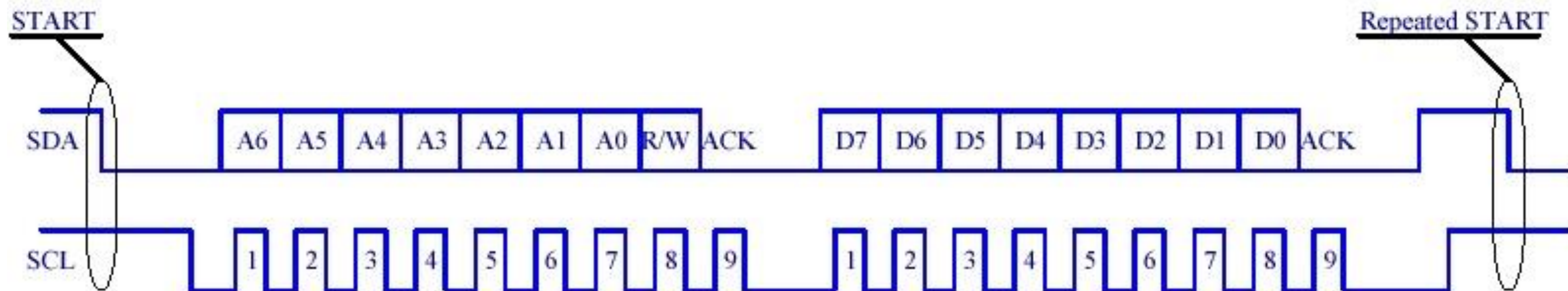
I2C Reads and Writes

- There are 2 basic types of communications a **read** and a **write**
- In a **read** the Master is requests data from the slave, the slave then responds with data bytes
- In a **write** the Master sends data bytes after the address which the are then interpreted by the slave device
- In both cases there can be multiple data bytes
- All transmissions are grouped into individual bytes

LaACES Student Ballooning Course

I2C Write Sequence

- A typical I2C bus sequence for writing to a slave device:
  - Send a START sequence
  - Send the I2C device address with the R/W **Low** (for Write)
  - Send the data byte
  - Optionally send additional data bytes (after repeating START)
  - Send the STOP sequence after all data bytes have been sent
- The Slave responds by setting the ACK bit (Acknowledge) after every byte.

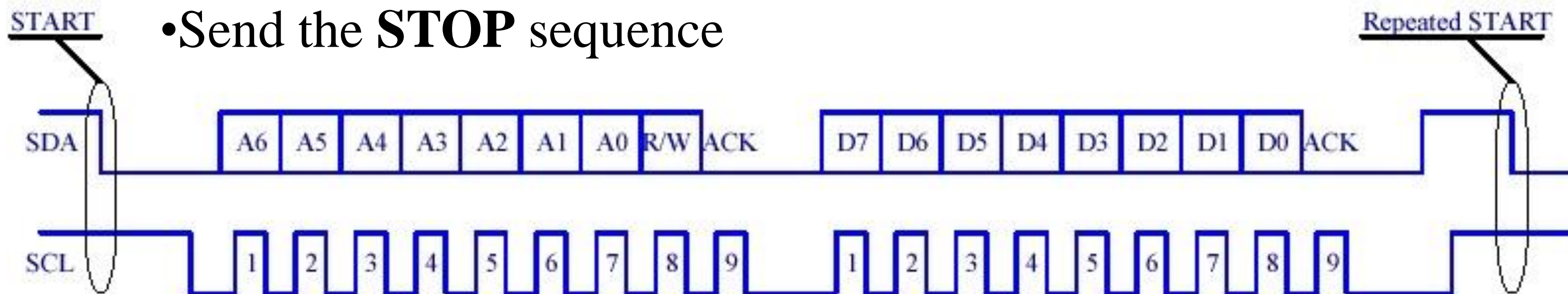


LaACES Student Ballooning Course

I2C Read Sequence

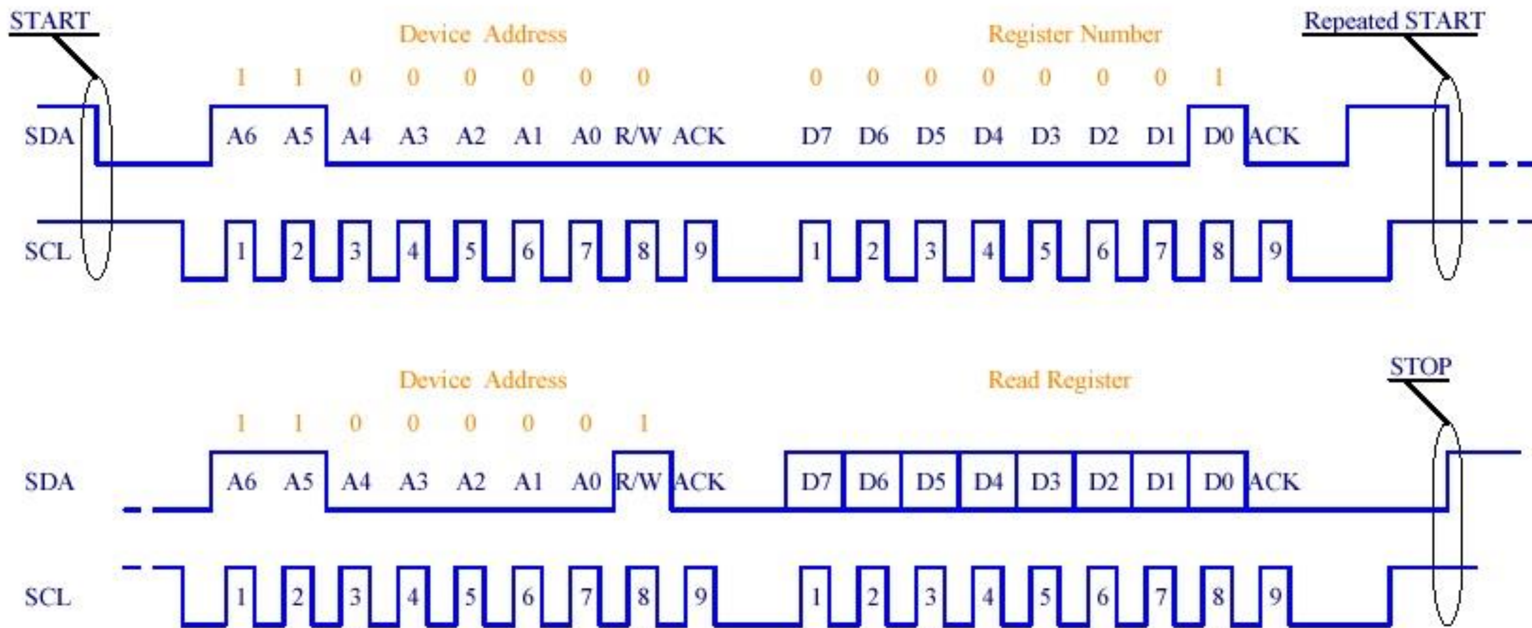
- Reading an I2C Slave device usually begins by writing to it. You must tell the chip which internal register you want to read.

- I2C Read Sequence
  - Send the **START** condition
  - Send the **device address** with R/W held Low (for a Write)
  - Send the number of the **register** you want to read
  - Send a repeated START condition
  - Send the device address with R/W set **High** (for a Read)
  - **Read** the data byte from the slave
  - Send the **STOP** sequence



LaACES Student Ballooning Course

I2C Read example using device address 1100000 and reading register number 1.



I2C Programming on the Arduino Mega

The **Wire** library is used to communicate to devices using the I2C bus. The functions available are:

- begin()
- requestFrom()
- beginTransmission()
- endTransmission()
- write()
- available()
- read()
- SetClock()
- onReceive()
- onRequest()

The Arduino website has excellent explanations of how these functions work and a plethora of examples of how to use them.

LaACES Student Ballooning Course

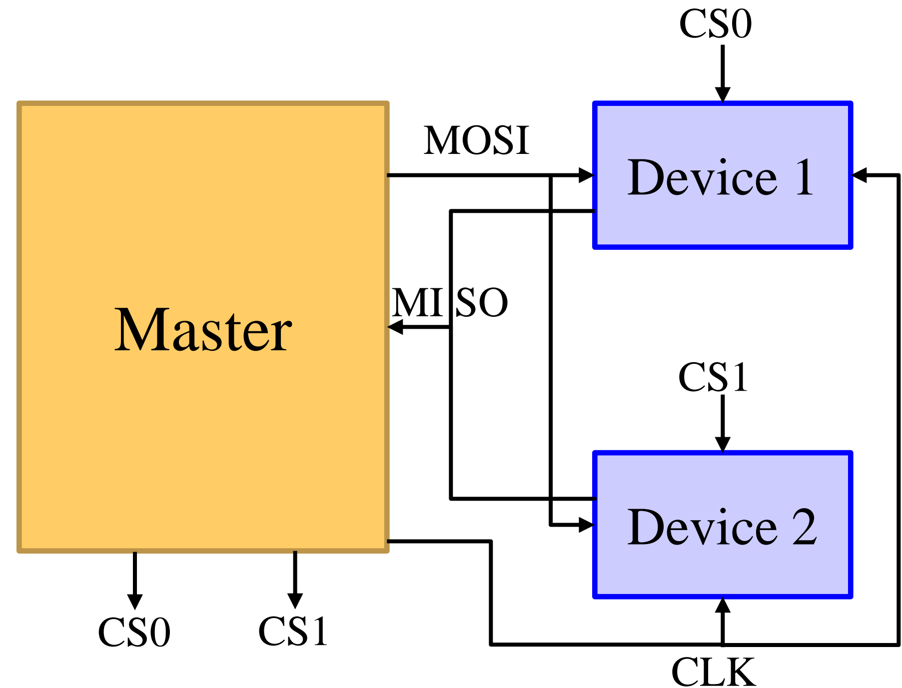
SPI Protocol

- Developed By Motorola
- Less standardized that I2C, varies device by device
- 4 Clock Modes
  - Clock can idle high or low
  - Can trigger on rising or falling edge
- Devices are not fixed in individual bytes(ex. 13 bits long) but Arduino library reads an transmits in bytes so bitwise operation will be required (shifting and combining bytes)

LaACES Student Ballooning Course

SPI Pins

- Master Controlled **CLK** line to synchronize signal
- 2 Unidirectional Data, **MISO** and **MOSI**
  - Can simultaneously transmit and receive
- Each device requires a separate Chips Select (CS) pin, Master sets low to activate the device



Level Shifting

Protocols can operate at different logic levels. For example, the Arduino Mega digital pins communicates with +5V, while the Raspberry Pi communicates with +3.3V. It is possible for a Raspberry Pi to communicate with a Mega through level shifting.

Level Shifting is the conversion of logic signals from one voltage level to another. It converts the “HIGH” voltage level of the input to a different voltage. In the example above, a level shifter would change the 3.3V logic to 5V logic or vice versa.

Level shifters may bidirectional or unidirectional and may work at different voltage levels and frequencies so be sure to check the datasheet to find on that works with your application.