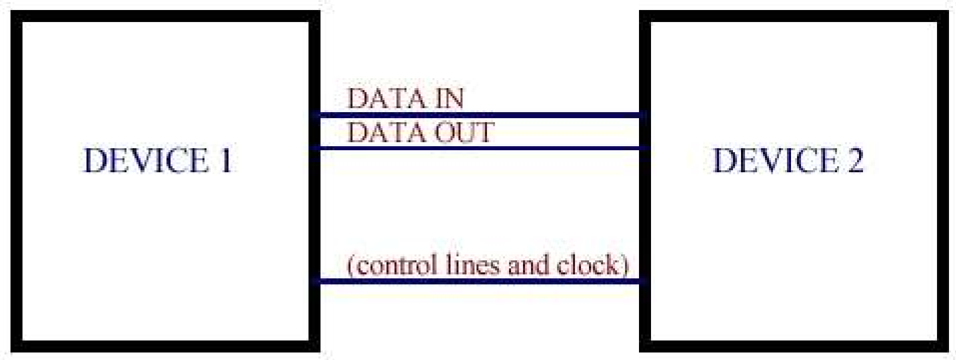
LaACES Student Ballooning Course

Serial I/O

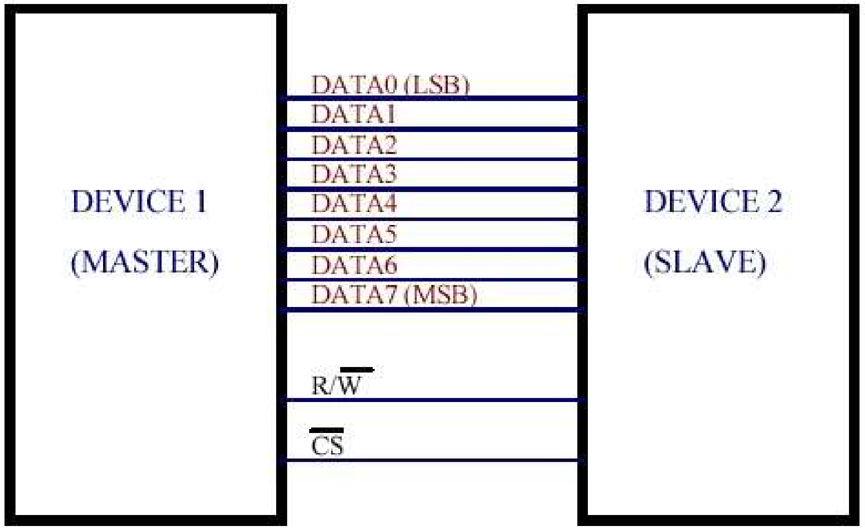
Serial communication sends one data point at a time, as opposed to *parallel* communication which sends multiple data points at once.

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.

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

Serial hardware communicates via electrical pulses that represent 1’s and 0’s. The Arduino uses a Universal Serial Bus (USB) to connect to the serial hardware.

Serial communication is also possible via software libraries, but this is not as efficient.

The Mega has one USP port and 10 serial pins

LaACES Student Ballooning Course

Synchronous and Asynchronous Serial Communication

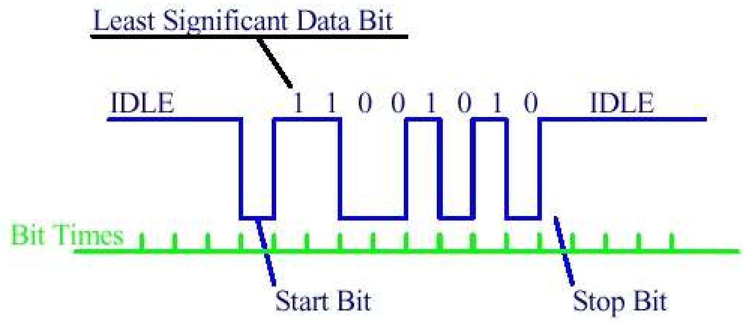
- Synchronous means the devices involved use the same clock-signal when communicating, while asynchronous means the devices use their own individual clocks.
- Pins 1 & 14-19 use a *Universal Asynchronous Receiver/Transmitter* (UART) for asynchronous communication. Yet, no two clocks run the same, so 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 **53** 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.



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.

LaACES Student Ballooning Course

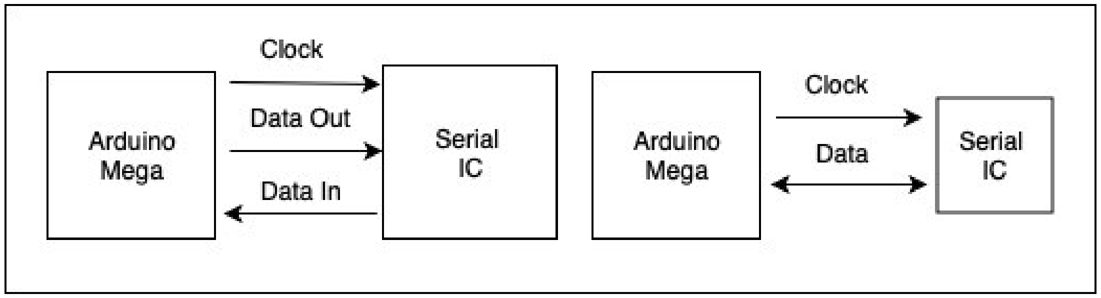
Extra Serial Ports

The **Arduino Mega** has three additional 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.

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.



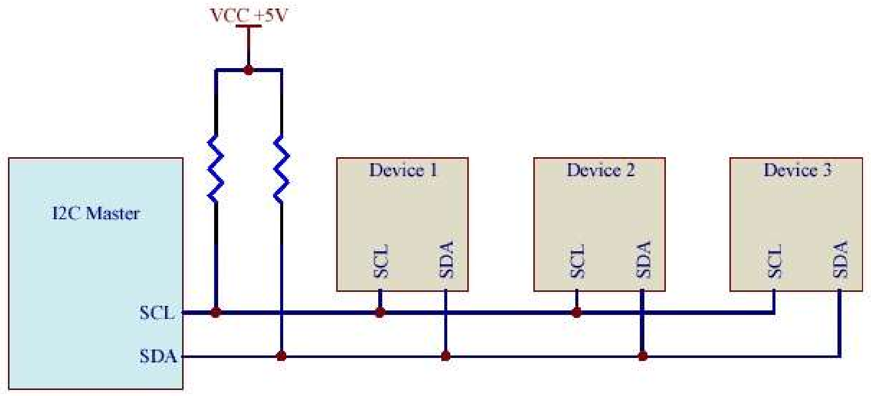
LaACES
Student
Ballooning
Course

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 data line **SDA**



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

•A **device address** can be a combination of bits that 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.

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

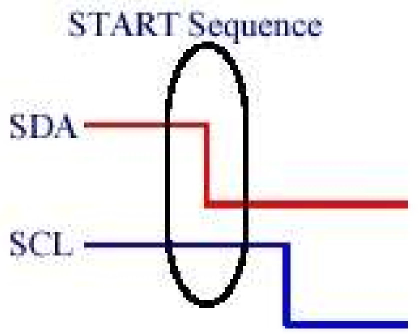
•A variety of special function integrated circuits are available with I2C interfaces, including memory chips, analog-to-digital converters, digital-to-analog converters and real-time-clocks.

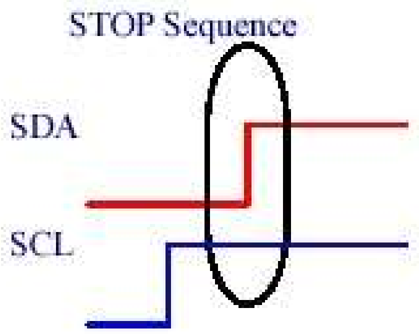
LaACES
Student
Ballooning
Course

I2C START and STOP

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

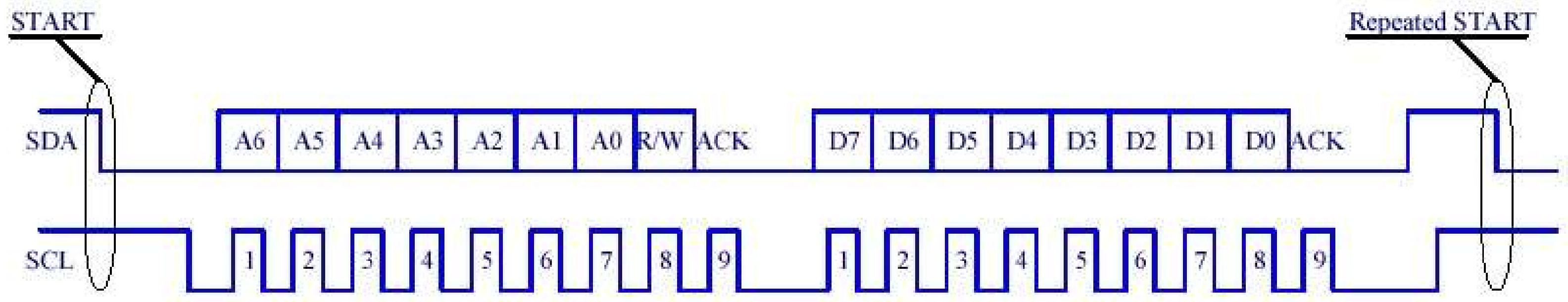
•A **STOP** sequence ends a transmission. The **Stop** sequence occurs when the master brings **SDA** from **Low to High** while **SCL** is **High**.





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)



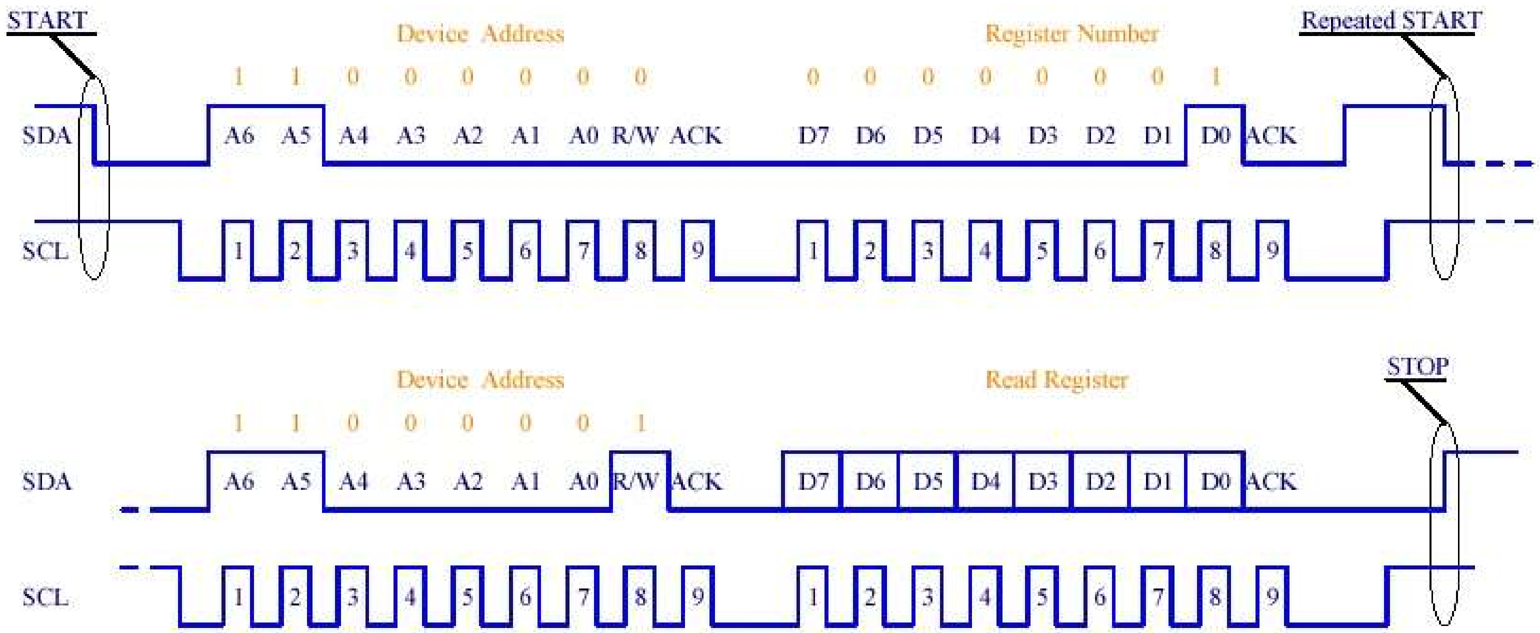
I2C Read Sequence

•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.



LaACES Student Ballooning Course

I2C Programming on the Arduino Mega

The wire library allows you to communicate to slave devices using the I2C bus. The functions available are as fallows:

- 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.

Other Synchronous Serial Protocols

Several IC manufacturers have developed their own protocols for synchronous serial communication. The data sheets and application notes from the chip manufacturer should be consulted for design and troubleshooting tips.

- A popular serial I/O protocol is
  - **SPI** (serial peripheral interface from Motorola)