Summary:

Students will form small groups and use the *Arduino Mega* to demonstrate serial communication between 2 Arduino microcontrollers. Students will connect the Arduinos, designating one as the master, and the other the slave, and write 2 *simple* programs allowing the Arduinos to communicate via their I2C's.

Materials:

Each student team should have the following materials, equipment, and supplies:

1. Two Arduino Megas w/ USB cable
2. A computer with Arduino IDE installed
3. Jumpers or wires

Procedure:

Note: If available you can hook up an oscilloscope to the SDA and SCL lines to look at the signal as they are transmitted. Do not forget to connect the grounding clip to an Arduino ground.

Setup

1) Connect the grounds of both Arduinos to each other.
   Note: This is called "sharing a common ground", we need to do this to ensure 0V for the Arduino to be the same for both.
2) Pick one Arduino to be the Master and the other to be the Slave.
3) You will need to upload your code to each Arduino separately, this is easiest to do if you partner with each with 1 computer connected one Arduino. One computer can be used, you will just need to take care you upload to the correct Arduino and may have to manually select the port from the tools menu when you switch.
4) Connect the master board to your computer via USB.
   Note: If powering the boards independently is an issue, connect the 5V output of the Master to the VIN pin on the slave. This will let us power the Master Arduino from the USB and the Slave will be powered from the Master.
   Note: When using the Serial Monitor be sure the correct port is selected, also you will need to close the serial monitor before you can upload code when switching back and forth.
5) Connect the wire from pin 20 on the master to pin 20 on the slave, and the wire from pin 21 on the master to pin 21 on the slave. This connects the 2 Arduinos SCL and SDA lines.
6) In this activity the slave Arduino will be taking the role that would normally be filled by some sort of IC chip like a clock or a sensor.

Part 1: String Response

7) The first thing that we want to do is write two simple programs. The master will send a read request expecting a response of fixed number of bytes, it will receive those bytes from the slave and display them on serial monitor. The library for I2C is the Wire library so we will need to include it in both our Master and Slave code.
.
8) First let's write the program for the slave Arduino.

```
void setup() {
  Wire.begin(16); //Here we initialze the I2C bus, note we have the address 16 in the
                  //parenthesis for begin() because this device will be the slave
  Wire.onRequest(requestEvent); //This tells the Arduino what function to call when it get a read request from the master
}
```

Here we need to give an argument of 16 in Wire.begin(), setting the Arduino up as slave device with the address of 16. The Arduino will only respond to communications with that address and will ignore all others.

The Wire.onRequest() sets up the function that will execute when the Arduino receives a read request from the master. So this is saying we later have a function **void requestEvent()** later in our sketch. Every time the Arduino sees a read with address 16 it will execute that function.

We could have named the function something else, the name just needs to be the same between the what is inside the onRequest() and the function definition. But the function does need to be a void and cannot have any arguments.

When the request is received the program will pause what is doing and go run the requestEvent(). This happens because there is hardware on the SDA and SCL pins that will trigger when they see the start request on those pins. This sort of behavior is called an **INTERRUPT** because it interrupts the normal flow of the program

9) The loop for the slave is simple, we want the slave Arduino to do anything other than respond to I2C so we will just add a short delay().

10) We now need to write requestEvent().

```
void requestEvent(){ //This function is what executes when the read request occurs, remember we want to send 5 bytes
  Wire.write("hello");  //We have a few responses here that all work so we can change the response only on the slave and
                        //see the response on the master

  //Wire.write("hola ");  //Other responses we could change to by commenting and uncommenting the lines
  //Wire.write("Snail");  //We only want one of these to be active at the once, otherwise we will be sending more bytes
                          //than the master is expecting
  /* //Remember the master will only get raw bytes so if we send numbers will assume they are characters
  Wire.write(49); //ASCII 1
  Wire.write(32); //ASCII space
  Wire.write(50); //ASCII 2
  Wire.write(32); //ASCII space
  Wire.write(51); //ASCII 3
  // */
}
```

All we need in this function is a Wire.write(). Whatever we put in the parenthesis will be the bytes sent back to the master.
We want the response to be 5 bytes long. And we have a few different options shown that we can select my commenting. Remember each character is single byte.

We could also the bytes individually by using 5 separate writes. Also remember that this will be sent as raw bytes, so for the numbers in the 3rd example the program will convert those into the binary values for those number and send them each as a single byte. But the master has no way of knowing what type the data is so will assume that they are characters, and what it will display is “1 2 3” because the numbers we have chosen are the ASCII values for those characters.

11) That’s all we need for the slave so we should upload the sketch to our slave Arduino.
12) The setup on the master is very simple, we just need to initialize the Serial port and the I2C bus object.

```
void setup() {
  Wire.begin(); //Here we initialze the I2C bus, note we have no address in the
                //parenthesis for begin() because this device will be the master
  Serial.begin(9600); //We initialize the serial port so we can send our output to the serial monitor
}
```

13) Note that we do not need an argument inside the Wire.begin() because this Arduino is the Master, so it does not need an address.
14) The loop is a bit more complicated. We need to first send a read request and read the bytes that get sent back. We also want to declare a variable that will temporarily store each character in as we receive them.

```
void loop() {
  char c; //This is the variable where we will temporarly store the characters we receive

  // First we will request the response from the slave of a known length
  //We will have the slave join the I2C bus with address of 16, remember addresses are limited to 7 bits

  Wire.requestFrom(16, 5);//This line will request 6 bytes from the device with address 16
                          //The data received will be raw bytes so we need to know the appropriate way to interpret the bytes
                          //In this case since we are writing the code for the other Arduino we know these are going to be
```

Wire.requestFrom() does 2 things it sends are read request to the device with the address of the first argument, the 16 in this case, and then it reads number of bytes given by the second argument, 5 in this case, into a space in memory set up by the library called a buffer.

15) Now we need to copy each byte out of buffer and display them on the Serial monitor until the buffer is empty.

```
  while (Wire.available()){   //Wire.availble() returns the number of bytes in the buffer, so will evaluate to true as long as there
                              //more bytes to read
    c = Wire.read();          //Returns the next available byte in the buffer then deletes it
    Serial.print(c);
  }
  Serial.print("\n");   //Starts a new line
  delay(1000);        //Just a delay to prevent spamming
}
```

We do this with a loop. Wire.Available() returns the number of bytes in the buffer so while loop will execute until the buffer is empty.

Wire.Read() will return the first byte in the buffer and then delete it out of the buffer so we want make sure we copy it into the temporary variable c.

Then we will print it out to serial monitor.

16) The last 2 lines just start a new line and has 1 second delay.
17) So now once we upload the code to the Master Arduino, we should see the appropriate string displayed on the serial monitor when both Arduinos are powered and correctly connected. Try changing the response string and uploading the change to the slave Arduino. You could also change the length of the string to see the behavior when it does not correctly match.

Part 2: Sending a Math Request

18) Next, we want to use the Wire.write() to send a series of bytes that will send 2 numbers and bytes indicating which type of operation to do on the numbers, either addition or subtraction.

19) Let's, start with the slave Arduino first. We are going to need variables to store our values for

```
#include <Wire.h>

byte a=0;     //This first number we receive from the Master
byte op=0;    //this is the byte we will send to tell which operate to perform on the 2 numbers
byte b=0;     //This is the second number we receive the Master
byte result=0;
//Sample Code for Activity 11.1
//This is the code for the Slave Arduino for Part Two
```

2 numbers, the operation and we will access these variables in multiple functions so let's make them global. Also declare them as bytes so we are sure they all only a single byte

20) Our setup of the slave is very similar except now we need to add a second event.

```
void setup() {
  Wire.begin(16); //Here we initialze the I2C bus, note we have the address 16 in the
                  //parenthesis for begin() because this device will be the slave
  Serial.begin(9600);
  Wire.onRequest(requestEvent); //This tells the Arduino what function to call when it get a read request from the master
  Wire.onReceive(receiveEvent); //This is the function called when it gets a write operation

  //Remember I2C has 2 types of communications a read request where the slave sends bytes back
  //and a write where the address is followed by a number of bytes that are read the slave device
}
```

onRequest() again tells the function to excute on a read. onReceive() designates the function to call when the Arduino gets a I2C write. We also intialize the Serial in case we want to hook out serial monitor to troubleshoot.

21) The basic idea that we are going to use is the master will send a write that will send the first number, the desired operation, and the second number. The slave will read those bytes and do the math. Next the Master will send a read and the Slave will reply with the result.

22) The loop again does nothing so we just have small delay.

```
void loop() {
  delay (10); //This should not really be doing anything in the loop
}
```

23) So in our receiveEvent() we need to read each individual byte and story it in the correct variable. We will need to make sure we send the values in the master in the right order that matches the order we read here ie: number 1, operation, number 2.

```
void receiveEvent(){  //Here we will read in the 3 bytes we are expecting from the Master

  a=Wire.read();    //Read the first number
  op=Wire.read();   //Read the operation byte
  b=Wire.read();    //Read the second number
  Serial.println(a);
  Serial.println(op);
  Serial.println(b);

}
```

We are also printing the received values out to serial port (if it is connected) so that troubleshoot if something does not work right.

Also, we do not need a Wire.requestFrom() because we are reacting to write which will include the data.

24) We will hand the actual math in the requestEvent(). Remember this is what is going to execute when the Master sends a read request. We are going to have 3 possible actions depending on the op byte.

```
void requestEvent(){ //This function is what executes when the read request occurs, we will send back the result of the operation
  //We will have 2 possible operations, a 1 op byte will tell the slave to do an addition
  // a 2 op byte will tell the slave to do a multiplication

  if (op==0){
    result=0; //Send a 0
  }

  if (op==1){
    result=a+b;
  }

  if (op==2){
    result=a*b;
  }

  Wire.write(result);
```

The point of the op==0 part is in case the read gets sent before the a write has been received. In each case a result is calculated and sent back as a single byte.

25) Now our slave code is done we should upload it to the slave Arduino.

26) For the master our set up is the same as part 1.

```
#include <Wire.h>

//Sample Code for Activity 11.1
//This is the code for the Master Arduino for Part one

void setup() {
  Wire.begin(); //Here we initialze the I2C bus, note we have no address in the
                //parenthesis for begin() because this device will be the master
  Serial.begin(9600); //We initialize the serial port so we can send our output to the serial monitor
}
```

27) We will do everything in the loop, we need up our variables for our 2 numbers, we will try 3 and 8 first.

```
void loop() {
  int c; //This is the variable where we will temporarly store the result we receive

  byte a=3;
  byte b=8;   //The two values we will try use for our math operations
```

28) Now we want to send the write in to correct order, we will ask for the slave to add the 3 and 8 first. On the master the write works a bit different. You tell the Arduino you are going to send a write to a device with a certain address using the Wire.beginTransmission().

Then you add bytes to the transmission buffer using Wire.write().

Then you actually close the buffer and send the bytes by calling Wire.endTransmission(). This will send all of the bytes queued by the Wire.writes() since the beginTransmsion() in

```
//Requests an addition
Wire.beginTransmission(16); //This will build up a write transmission to device with address 16
Wire.write(a);    //Queue the first byte
Wire.write(byte(1));  // Queue the operation
Wire.write(b);     //Queue the second byte
Wire.endTransmission(); //Ends the queue and sends the byte string
```

order.

29) Then we need to to a read to request the result.
30) This works just like part 1 except we are only expecting 1 byte this time

```
Wire.requestFrom(16, 1);//This line will request the result byte from the device with address 16
c = Wire.read();          //Returns the next available byte in the buffer then deletes it
Serial.println(c);

delay(1000);        //Just a delay to prevent spamming
```

31) We can then repeat the whole thing except we will send a 2 for the operation byte, asking for a multiplication.

```
//Requests an multiplication
Wire.beginTransmission(16); //This will build up a write transmission to device with address 16
Wire.write(a);    //Queue the first byte
Wire.write(byte(2));  // Queue the operation
Wire.write(b);     //Queue the second byte
Wire.endTransmission(); //Ends the queue and sends the byte string

Wire.requestFrom(16, 1);//This line will request the result byte from the device with address 16
c = Wire.read();          //Returns the next available byte in the buffer then deletes it
Serial.println(c);

delay(1000);        //Just a delay to prevent spamming
```

32) So again, we upload our code to the Master and if we check our connection and make sure both Arduino are powered, we should see alternating lines of 11 and 24.
33) You should feel free to change the numbers and reupload the master code and see the results change. Without having to modify the slave code.

In general, this is how I2C readouts work, you send a write command to device (usually to tell it what value you want it to return) and then follow with a read to get the result.

Reference:

Arduino Wire Library Reference
https://www.arduino.cc/en/reference/wire