Embedded Hardware Design

Monsoon 2017

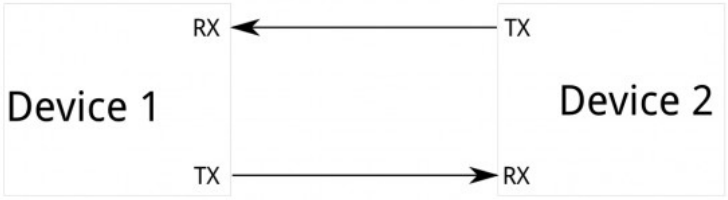
Experiment 5

Date : 14-09-2017
Topics: I^2C

Standard Communication Interfaces in Arduino-

In this lab, which happens to be the last lab on Arduino, we look at a very important and useful feature of microcontrollers- communication interfaces. In numerous applications of embedded systems, there is a need to transmit data from one device to the other. Arduino has 3 inbuilt wired communication interfaces- UART, SPI and I^2C. These interfaces allow you to connect devices such as RTCs, memories for parameter storage, sensor modules and even other microcontoller or computers to your microcontroller. The main focus of this lab would be on I^2C since UART has already been covered.

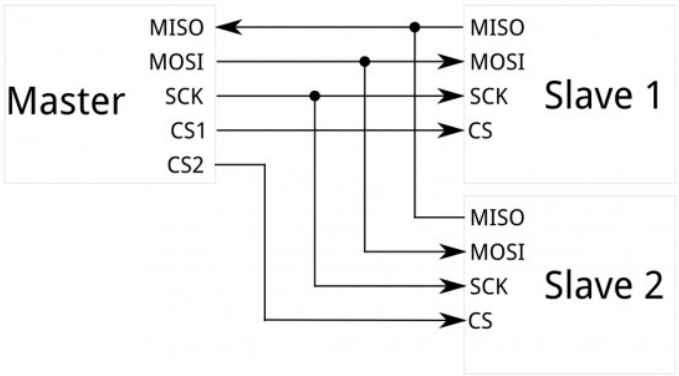
1. **UART-** Universal Asynchronous Receiver Transmitter is an asynchronous serial link having only 2 bus lines- Tx and Rx. Since no clock is involved in transmission and reception, the baud rate of the two devices must be fixed before starting communication.



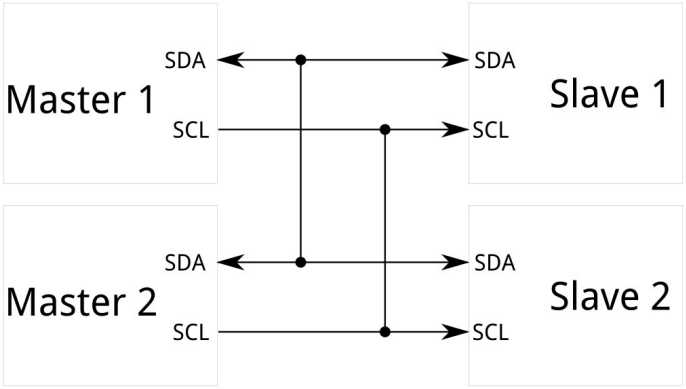
   We've been using UART since day one! Remember the *Serial* functions that you've been using up until now for your Serial Monitor? Your computer and your arduino have been communicating via UART (digital pin 0 and 1 are connected to the USB bus). You can also connect two Arduinos via UART and easily establish a connection using the same commands.

2. **SPI-** Serial Peripheral Interface is a synchronous protocol that supports multiple slaves connected to a single master. In short, you can have a Master Arduino connected to several slave Arduinos/SPI devices and perform a point-to-point full-duplex communication over the bus. The drawback of SPI is the

number of pins required for communication. *MISO*: Master-In-Slave-Out, *MOSI*: Master-Out-Slave-In, *SCK:* Serial Clock, *CS:* Chip Select.

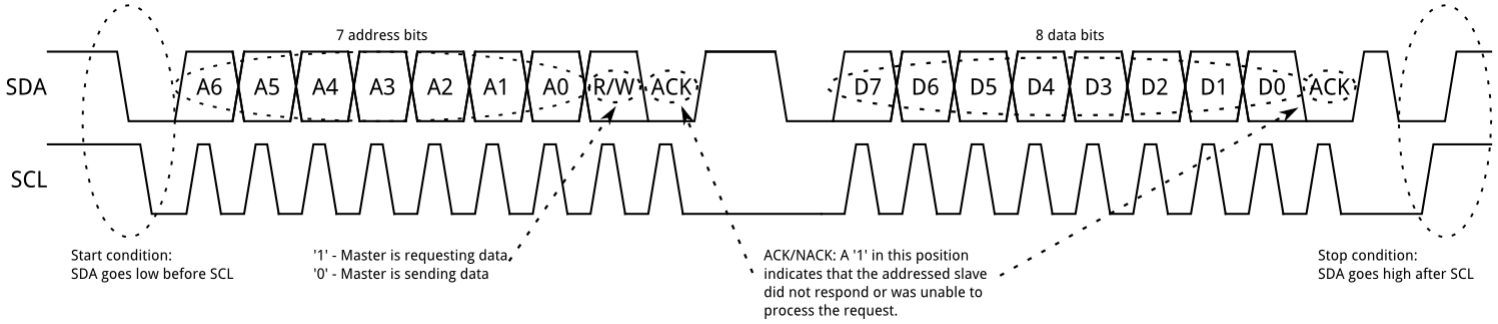


3. **I²C-** The Inter-integrated Circuit is a synchronous, two-wire protocol which allows multiple slave and master devices on the network. Like SPI, it is only intended for short range communication. Like UART, it only requires two signal wires to exchange information. *SDA:* Serial Data, *SCL:* Serial Clock.



I²C Protocol-

Communication via I²C is more complex than UART or SPI. The signalling must adhere to a certain protocol for the devices on the bus to recognize it as valid I²C communication.



First up, the messages are broken up into two types of frames: an address frame, where the master indicates the slave to which the message is being sent, and one or more data frames, which are 8-bit data messages passed from master to slave or vice versa. Note that each slave on the network has a unique 7-bit address. Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high.

1. **Start Condition**
   To initiate the address frame, the master leaves SCL high and pulls SDA low. This puts all slave devices on notice that a transmission is about to start.

2. **Address Frame**
   The address frame is always first in any new communication sequence. For a 7-bit address, the address is clocked out with MSB first, followed by a R/W bit indicating whether this is a read (1) or a write (0) operation. The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address). Once the first 8 bits (address + r/w) of the frame are sent, the slave with the same address is given control over SDA. If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that the receiving device either did not receive the data or did not know how to parse the message.

3. **Data frame**
   After the address frame has been sent, data transmission begins. The master will simply continue to generate clock pulses at regular intervals and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation.

4. **Stop condition**
   Once all the data frames have been sent, the master will generate a stop condition. Stop conditions are defined by a low to high transition on SDA after a low to high transition on SCL, with SCL remaining high. During normal data writing operation, the value on SDA should not change when SCL is high, to avoid false stop conditions.

The I^2C Wire library in Arduino-

The A4 pin of Arduino performs an alternate function of I^2C SDA and A5 pin, I^2C SCL. The I^2C protocol is supported natively by the Arduino with <*Wire.h*> header. The following table lists some common functions of the *Wire* library-

Function	Description
Wire.begin(*address)*	Joins the I^{2}C bus as a master if address is not given. If an address is specified as a parameter, the Arduino joins the bus as a slave with that address. This function is called in the *setup.*
Wire.beginTransmission(*address)*	This is for Arduino configured as I^{2}C master: initiate transmission with the slave that has the given address.
Wire.write(*data)*	Writes data over I^{2}C. *Data* can be a byte, integer or a floating point. For master configured in Write mode, this function is called after *beginTransmission.* For slave, this is called in *onRequest* function.
Wire.endTransmission()	Ends the current transmission.
Wire.onReceive(*func_name)*	Registers a function to be called when a slave device receives a transmission from a master.
Wire.read()	Used by slave to read bytes from buffer after master's transmission. Used by master to read bytes from buffer after calling *requestFrom.*
Wire.requestFrom(*address, bytes)*	Request a specified number of bytes from the slave with the given address. This changes the Write configuration of master to Read.
Wire.onRequest(*func_name)*	Register a function to be called when a master requests data from this slave device.
Wire.available()	Returns the number of bytes available for retrieval with read(). This should be called on a master device after a call to requestFrom() or on a slave inside the onReceive() handler.

Experiment 5A: Establish a simple I^{2}C communication between 2 Arduinos

1. Connect two Arduinos as follows-

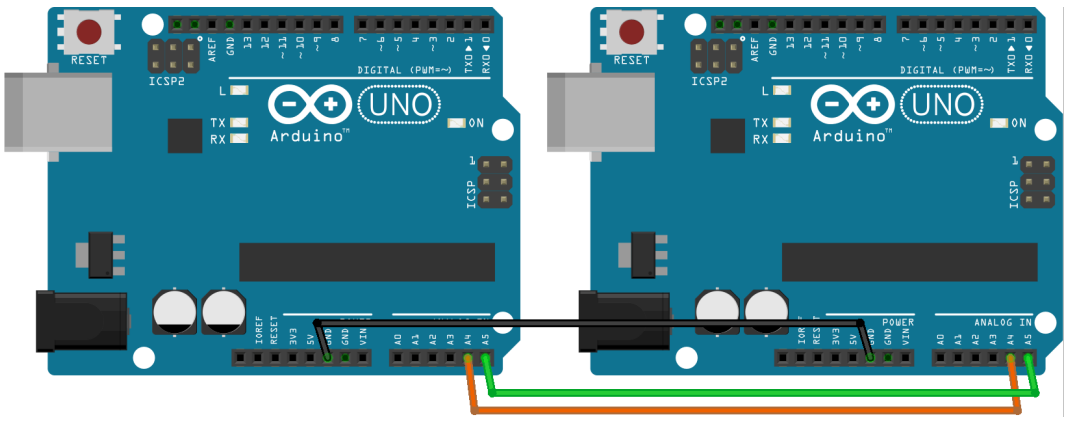


Illustration 1: Two Arduinos connected as I2C Master-Slave pair

A4 (SDA) --- A4 (SDA), A5 (SCL) --- A5 (SCL), Gnd --- Gnd

2. Configure one Arduino as Master and the other as Slave.
3. In Master, implement a 4-bit up counter and a 5-bit down counter. The value of the counters should be altered and sent to the slave every second.
4. Configure slave to receive both values and print it on the Serial Monitor.
5. Use the following starter code-

```
//MASTER
void setup() {
     // join i2c bus as Master
}

byte count1=0;
byte count2=31;
void loop() {
 // begin transmission to slave device
 // write the current value of count
 // End transmission

 // Counter increment/decrement logic
 delay(1000);
}
```

```
//SLAVE
void setup() {
     // join i2c bus as Slave with Address
     // set onReceive function
     Serial.begin(9600); //For Serial monitor
}

void loop() {
     delay(20);
}

void receiveEvent(int howMany) {
     while(Wire.available()!=0) //Till buffer is not empty
     {
          // read byte
          // print byte to Serial Monitor
     }
}
```

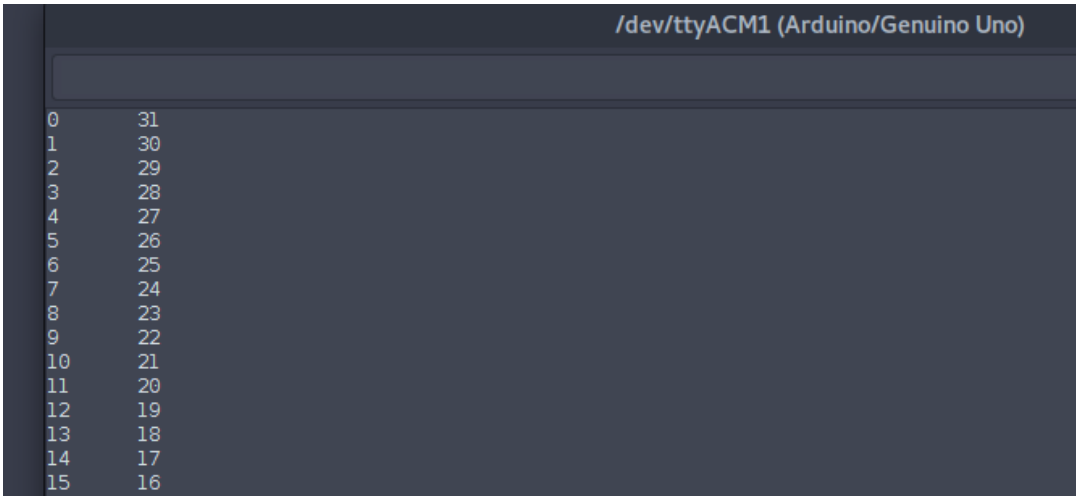
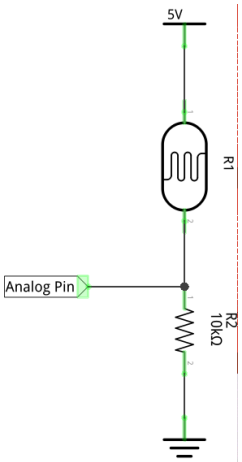


Illustration 2: Expected output format at Slave end in Exp 5A

Experiment 5B: Sensor Network

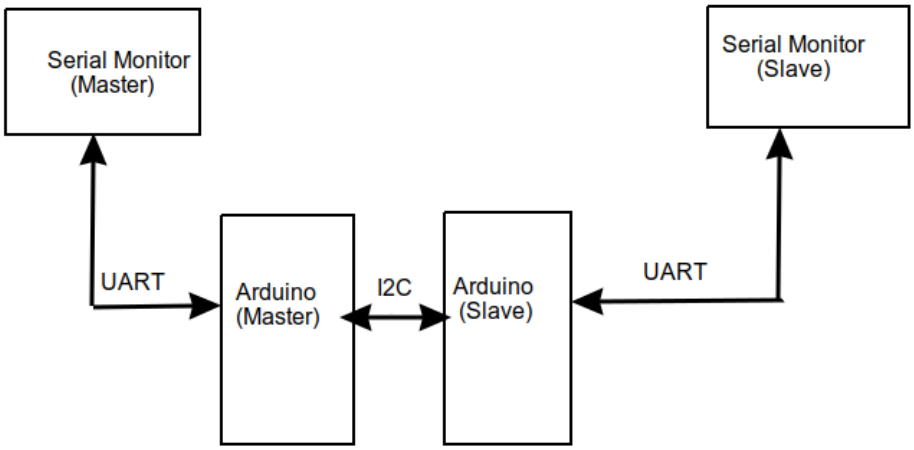
Communication protocols are widely used in sensor networks. Several slaves nodes are deployed at different places with sensors attached to them and they send the data to the master at regular intervals. In this experiment, we use an LDR attached to the slave to control the brightness of an LED attached to the Master.

1. Connect two Arduinos as per figure 1.
2. Configure one of them as Master and the other as slave.
3. Interface an LDR to the analog pin of Slave and an LED to a digital pin.



4. In Slave, keep on sampling the current analog value of the LDR. Map the value to 0-255 and store it in a byte global variable.
5. Write the requestEvent routine to send the value to the Master.
6. In Master, request data from slave every 0.5s. Print the value on Serial Monitor and based on the value, control the LED brightness (PWM).

Experiment 5C: Chat Box



1. Configure both Arduinos as slaves by providing separate addresses in the *begin* function.
2. Configure *onReceive* function for both to output data from buffer to serial monitor.
3. The loop function in both should check if UART buffer is empty or not. If not, take role as Master, enable transmission and send data to the other device till it is empty.