Episode 3

Communication Protocols

Why do we need them? Well, all communication needs a well defined set of rules and procedures to be effective, and that is what a communication protocol is. Common protocols are I2C, UART and SPI. The aforementioned examples each have their pros and cons and specific use-cases.

Key concepts

- Serial communication- Data is sent one bit at a time over a single transmission line
- Parallel communication- Many bits of data are sent over a set of transmission lines “parallel”
- Synchronous communication- All data transfers are timed and performed in sync with a common clock signal
- Asynchronous communication- There is no clock reference to govern the timing of data transfer. Here, transfer of data is usually timed by means of handshake signals, ie. Acknowledgements for confirming that the requested data has been sent/received successfully.
- Master vs slave- Communication is always between sets of at least two devices. One of them is usually designated the master and instructs other devices connected to it. The other one, naturally, is called the slave, that services the requests of the master device.

Universal Asynchronous Receiver/Transmitter (UART)

This is a simple yet effective means of communication that is used often. It is clear that this protocol utilises asynchronous transfers of data between a master and slave device. Timing however, is not done with the help of handshake signals. Special start and stop bits and a parity bit are used to assist data transfers. This protocol uses two transmission lines- TX (transmit), and RX (receive). The corresponding TX and RX pins of the devices in use are simply connected to each other, along with a common ground connection. The TX and RX pins on an Arduino UNO are pretty easy to find as they are clearly labelled.

Data is sent in the form of packets containing 5 to 9 bits. The start and stop bits indicate the start and end of a packet. The parity bit is used verify the integrity of the transmitted packet.

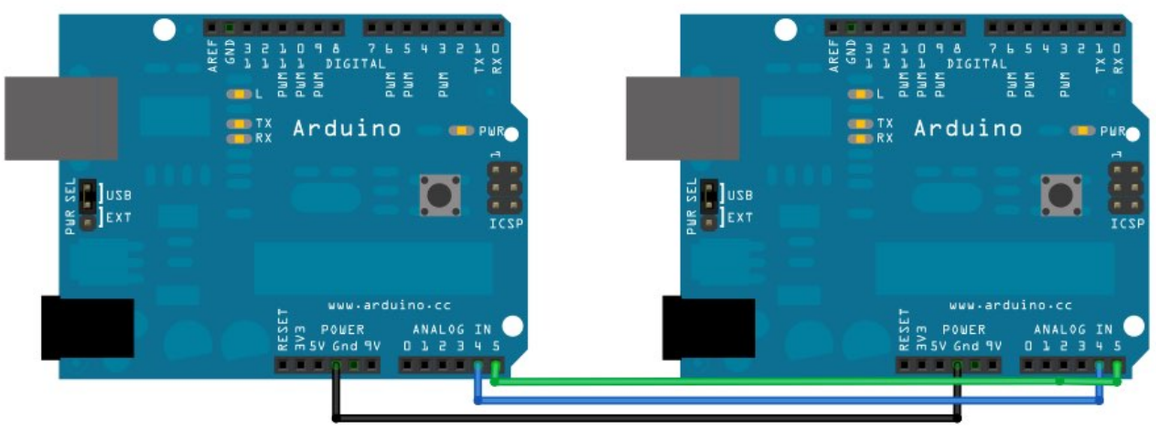
Example:

```
/* Transmit */
void setup()
{
	Serial.begin(9600);
}

void loop()
{
	Serial.write("Hello!!"); // Send 7 chars (7 bytes) to TX line
	delay(1000);
}
```

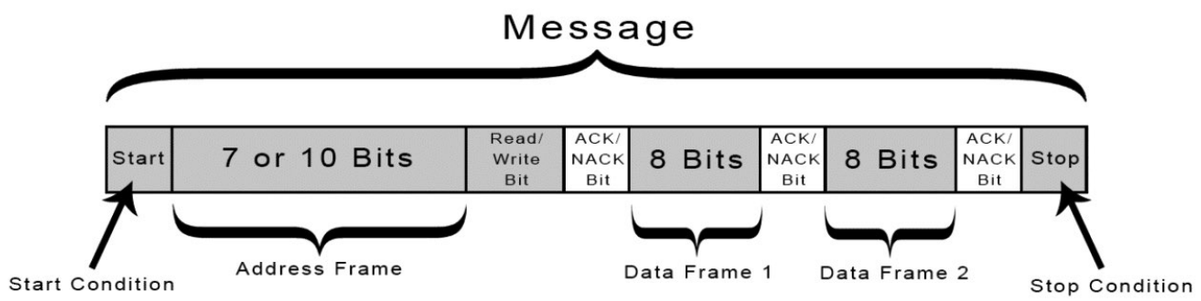
```
/* Receive */
char str[10];
void setup()
{
        Serial.begin(9600);
}

void loop()
{
        Serial.readBytes(str, 7); // Read 7 bytes at a time from RX line
        Serial.println(str);
        delay(1000);
}
```

Inter-Integrated Circuit (I2C) Bus Protocol

The I2C protocol is another popularly used communication protocol. The idea here is that devices connected to each other are either designated as a master or a slave. The master device has the ability to request data from the slave device or write to a slave device. Data is serially transferred but the transfer is clocked, making this a synchronous protocol. Devices are designated a data and a clock line and they are connected to each other, along with a common ground. For 2 Arduino UNO's, the scheme will be like this-
Pin A4 is the data pin (SDA)
Pin A5 is the clock pin (SCL)



Messages are sent a bit differently here. A message is split into several frames. The below scheme is followed-



Start Condition: The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.

Stop Condition: The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.

Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

ACK/NACK Bit: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

This protocol even allows for multiple slave devices (such as several sensors) to be connected to a single master (such as a microcontroller). It is also a rather fast way to transfer data, allowing speeds up-to 5Mbps.

For use with Arduino devices, the Wire.h library must be used. The various functions available are listed below-

- requestFrom() – Request data from a slave device
- begin() – Initialise I2C bus for data transfer
- beginTransmission()
- endTransmission()
- write() – Write data to I2C data line
- available() – Used to check whether data is available on the data line
- read() – Read data from I2C data line
- SetClock()
- onReceive()
- onRequest()

It must also be known that the code for the master and the slave differ. Here is an example for a master read operation-

```
#include <Wire.h>
/* Master Reader, to be uploaded to master device */
void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start Serial for output
}

void loop()
{
  Wire.requestFrom(2, 7);    // request 7 bytes from slave device #2
  while(1 < Wire.available()) // slave may send less than requested
  {
```

```
    char c = Wire.read(); // receive a byte as character
    Serial.print(c);         // print the character
  }
  int x = Wire.read();
  Serial.println(x);
  delay(500);
}

/* Slave Writer, to be uploaded to slave device */
void setup()
{
  Wire.begin(2);
  Wire.onRequest(requestHandler);
}

byte num = 0;

void loop()
{
  delay(100);
}

// Function that executes whenever the master requests
// data from the slave
void requestHandler(int byteCount)
{
  Wire.write("Hello ");
  Wire.write(num);
  num++;
}
```

Serial Peripheral Interface (SPI) Protocol

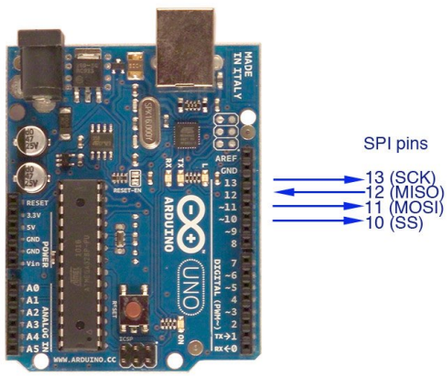
The SPI protocol is another commonly used system for data transfer. Here, the communication happens by sending data as continuous stream of bits through the wires physically. Here too, we have Master -Slave relationship. Both mandatorily contain 4 pins which are :

MOSI (Master Out Slave In) : Line for Master to send data to Slave.

MISO (Master In Slave Out) : Line for Slave to send data to Master.

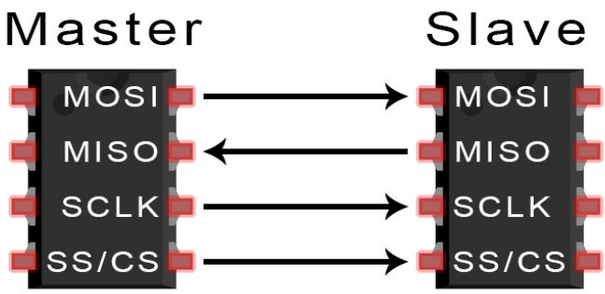
SCK/SCLK (Clock Signal) : Line for Clock Signal ; and

SS/CS (Slave/Chip Select) : Line for Master to select which slave to send data.



Working :

SCLK from Master synchronises the bits sent to Slave. One bit is sent in one clock cycle. Hence, can be concluded that speed is controlled by the frequency of the clock. Master chooses the Slave by pulling the respective SS line low. In idle state, SS line is kept high. If Master has multiple SS pins, then correspondingly multiple Slaves can be interfaced. If Master has a single SS pin, then Daisy Chain fashion can be deployed to connect the Master.
Master now sends the data one bit at a time in MOSI and Slave reads the stream of bits as received. In case response is needed, Slave utilises MISO to respond and Master reads the bits as received.



To employ SPI Protocol, we make use of the library SPI.h.

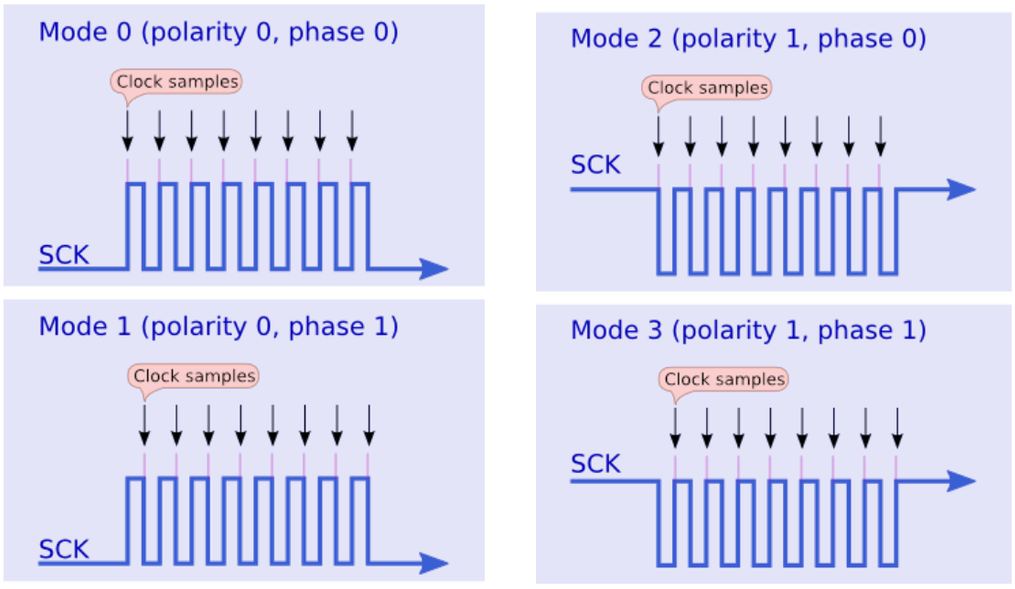
To initialise and use the communications protocols, various functions available are listed below-

- SPISettings() : This function takes three parameters which are frequency of data transfer, MSBFIRST or LSBFIRST and SPI_MODE.
- begin() : Initialises SPI bus and pulls SS low.
- end() () : Disables SPI bus.

- beginTransaction() : For using the SPI Ports.
- endTransaction() () : Used to end the communication but does not change SPI Settings.
- transfer() : Used to transfer the data.

To choose the SPI_MODE, we need to decide whether to send the data on Rising/ falling edge of clock i.e. polarity (CPOL) and phase of the clock i.e. clock is high or low (CPHA).

For example, SPI.beginTransaction(SPI.Settings(14000000, MSBFIRST, SPI_MODE0)) ; The above statement sets the clock frequency to 14MHz, data is ready to be sent from MSB initially and the Mode 0 is selected.



Advantages of SPI Protocol:

- No interruption in communication due to absence of Start/Stop bits.
- Not a complicated communication protocol.
- Higher rate of data transfer.
- Separate MOSI and MISO lines enables sending and receiving of data simultaneously.

Limitations of SPI Protocol:

- Makes use of four wires.
- No ACK signal (Acknowledgement) after data transfer.
- Error checking is not done.
- Only one Master can be used.