Throughout the course of H61AEE communications between Arduino microcontroller boards [1][2] and a range of peripheral devices will need to be established. The Arduino is equipped with a number of communication buses and this tutorial sheet covers I²C

I²C Communications

What is I²C?

I²C or Inter-integrated Circuit is a serial computer bus commonly used for establishing communication between devices and microprocessors or microcontrollers over short distances, [3]-[5]. In a typical implementation communication takes place between a 'master' and one or more 'slave' devices. The bus requires two data lines over which to establish communication, SDA (for data) and SCL (synchronisation clock) and the specification allows for multiple devices communicating on the same bus. Up to 112[1] devices can communicate on the same two-wire bus as each device on the bus will have a unique address which is used to choose which of the devices the master communicates with.

I²C on the Arduino

Connections to the Arduino's I²C bus are provided via two pins that can supply the SDA and SCL lines and communication is controlled via the use of the Arduino Wire library [6] which is included in the default installation of the Arduino Software (IDE) [7].

The location of the pins is dependent on the Arduino Board being used. The boards used throughout this module are the Arduino Nano v3.x [8] and the SDA and SCL lines are connected to the analogue IO pins A4 and A5 respectively, Figure 1.

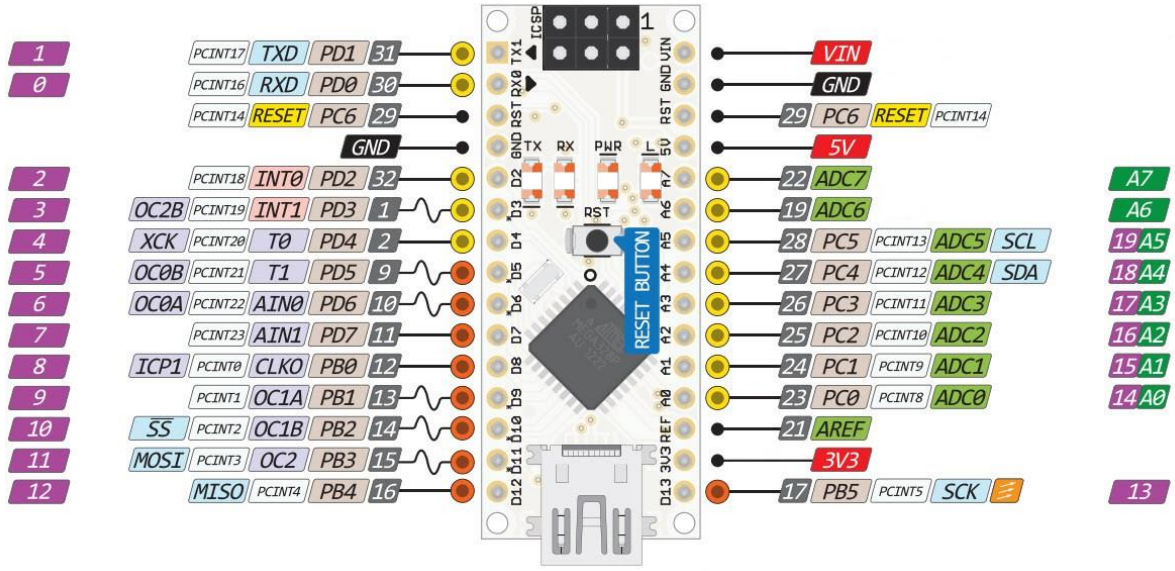

Figure 1. Complete pin assignment for the Arduino Nano version 3.x

[1] 112 devices if using 7-bit addressing

Note: earlier versions of the Arduino Nano use a different microcontroller and the lines are assigned to different pins. Also, there is a fair amount of incorrect information on the internet regarding the I2C pin assignment.

Connecting devices together via the I²C bus is straightforward and is schematically illustrated in Figure 2. The need for pull-up resistors will be dependent upon the devices connected to the bus, some may have them integrated and some may not. It's therefore essential that the datasheets are referred to when deciding whether they are required and if specific values are advised.

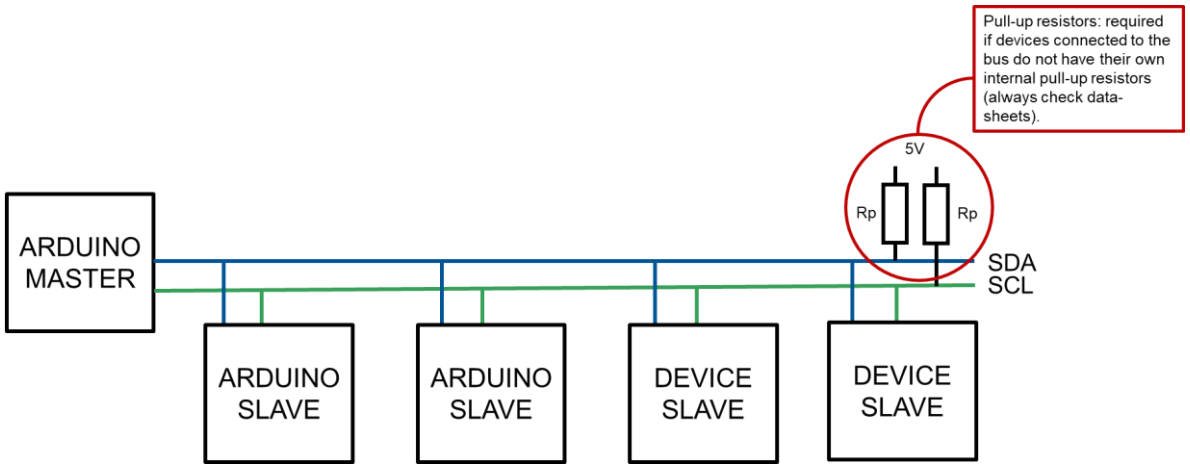

Figure 2 Illustration of the I2C bus wiring that also show the use of pull up resistors on the lines.

In the configuration shown in Figure 2 the master can read data from or write data to the slave devices on the bus. As previously mentioned, each device will have its own unique address on the bus which is used to direct the read/write requests to the intended device.

The Arduino Wire library is used to control communications over the I2C bus and the basic elements required for initialising the bus and for sending or receiving data are outlined in the following examples.

Example 1: Master Arduino Writing to Slave Arduino

With respect to the Arduino master, include the Wire library and then within `void setup()` join the I²C bus:

```
#include <Wire.h>          //include Wire library

void setup() {
  Wire.begin();            //join i2c bus (address optional for master)
}
```

All that is required to send data from the master to a slave device is the unique address of the intended recipient device and the data to be sent. In the following example communication will be with a slave Arduino whose unique address is defined within the code that will be uploaded to that device. In general slave devices will use a unique address that has been reserved for that device [9] which will be detailed on that device's datasheet. Therefore, if choosing an address for a device care must be taken to avoid using an address that may already be in use.

The essential elements required to send data from the master device (note that the 7-bit slave device address is written in hexadecimal, denoted by the prefix 0x):

```
Wire.beginTransmission(0x08);    //transmit to slave device address 8
Wire.write("hello");             //send five bytes, 1 for each character
Wire.endTransmission();          //end transmission
```

`Wire.beginTransmission` notifies the slave device with address '8' to expect some data. `Wire.write` is then used to send the actual data and in this example, it is the string "hello" which is equivalent to 5 bytes of data. Finally `Wire.endTransmission` ends transmission and frees up the I²C bus.

Off-the-shelf slave devices may come pre-programmed and all that is needed to control them is to send the necessary data. For example, the data required to set the value of a digital potentiometer. In this example however, an Arduino is to be programmed to act as a slave device that will simply receive data.

For the slave device again the Wire library is included and the I²C bus is joined within `void setup`. However, this time the address of the slave device is set upon joining the bus (here it is 8 represented in hexadecimal) and the 'handler' function called when the device receives data is defined, i.e. when the device receives data the function `recieveEvent` will be called within the Slave Arduino's application:

```
#include <Wire.h>                //include Wire library

void setup() {
  Wire.begin(0x08);              //join i2c bus with address 8
  Wire.onReceive(receiveEvent); //create a receive event
}
```

Upon receiving data, the `receiveEvent` handler function is called and the essential elements of receiving data are:

```
while (Wire.available()) {//loop whilst bus is busy
  char c = Wire.read();   //receive data byte by byte, store as char
  ...                       //do something with each char
}
```

The complete programmes for the master device sending the text string "hello" across the I2C bus and the slave device receiving the message and writing the received message to the serial monitor follow.

Master Send:

```
#include <Wire.h>      //include Wire library

void setup() {
  Wire.begin();        //join i2c bus (address optional for master)
}

void loop() {
  Wire.beginTransmission(0x08); //transmit to slave device address 8
  Wire.write("hello");          //send five bytes, one for each character
  Wire.endTransmission();       //end transmission
  delay(500);
}
```

Slave receive:

```
#include <Wire.h>         //include Wire library

String message;

void setup() {
  Wire.begin(0x08);              //join i2c bus with address 8
  Wire.onReceive(receiveEvent); //create a receive event
  Serial.begin(9600);            //start serial to visualise data
                                 //received via serial monitor in the IDE
}

void loop() {
}

void receiveEvent() {
  message = "";
  while (Wire.available()) {  //loop whilst bus is busy
    char c = Wire.read();     //receive data byte by byte
    message += c;             //form complete string
   }
  Serial.println(message);    //write string to serial monitor
  delay(500);
}
```

Note: within `void setup()` the serial connection is initialise so that received data can be visualised via the Arduino IDE's serial monitor.

Figure 3shows the breadboard implementation with the inset picture showing the output from the slave Arduino via the serial monitor.

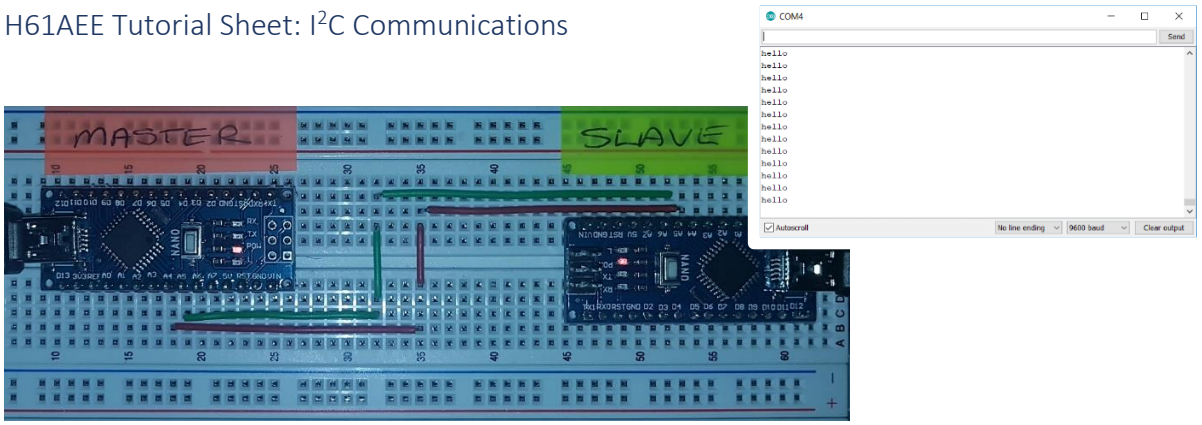

Figure 3 Master write, slave receive test set-up on a breadboard. Inset shows slave output via the serial monitor.

Example 2: Master Arduino Reading from Slave Arduino

In order for the I²C master to receive information from a slave device on the bus it first needs to join I²C bus within `void setup()` and here the serial connection is also initialised allowing data received to be visualised via the serial monitor:

```
#include <Wire.h>      //include Wire library

void setup() {
  Wire.begin();        //join i2c bus (address optional for master)
  Serial.begin(9600); //start serial to visualise data
                       //received via serial monitor in the IDE
}
```

Next the master has to request data from the specific slave device on the bus, therefore the unique ID of the slave device needs to be known in advance and data is requested using the function `Wire.requestFrom` called from within `void loop`. Data, representing a text string, is then received one byte at a time whilst the bus is active and the string 'message' is constructed one char at a time as in the previous example:

```
Wire.requestFrom(0x08, 5);       //request 6 bytes from slave device #8

  while (Wire.available()) {     //loop whilst slave sends data
                                 //i.e. whilst bus is busy
  char c = Wire.read();          //receive data byte by byte
  message += c;                  //form complete string
}
```

For the slave device again the Wire library is included and the I²C bus joined within `void setup` with its address set as 0x08. A 'handler' function is again called when the device receives a request to send data is defined also defined here, i.e. when the device receives a request for data the function `requestEvent` will be called within the Slave Arduino's application:

```
#include <Wire.h>                //include Wire library

void setup() {
  Wire.begin(0x08);              //join i2c bus with address 8
  Wire.onRequest(requestEvent); //create a receive event
}
```

Upon receiving a request for data, the `requestEvent` handler function is called and the message 'hello' is sent using the function `Wire.write`:

```
Wire.write("hello");     //respond with message 5 bytes long
```

The complete programmes for the master device sending requesting a 5 byte message across the I2C bus and the slave device responding to the request, the message received by the master being displayed via the serial monitor of the Arduino IDE follow.

Master Request & Receive:

```
#include <Wire.h>    //include Wire library

String message;

void setup() {
  Wire.begin();       //join i2c bus (address optional for master)
  Serial.begin(9600); //start serial to visualise data
                      //received via serial monitor in the IDE
}

void loop() {
  Wire.requestFrom(0x08, 5);    //request 6 bytes from slave device #8
  while (Wire.available()) {    //loop whilst slave sends data
                                //i.e. whilst bus is busy
    char c = Wire.read();       //receive data byte by byte
    message += c;               //form complete string
  }
  delay(500);
}
```

Slave Send:

```
#include <Wire.h>                //include Wire library

void setup() {
  Wire.begin(0x08);              //join i2c bus with address 8
  Wire.onRequest(requestEvent); //create a receive event
}

void loop() {
  delay(500);
}

void requestEvent() {
  Wire.write("hello");           //respond with message 5 bytes long
}
```

Example 3: Master Arduino Send/Receive To/From Slave Arduino

Combining elements of both examples bidirectional communications can be achieved with the master sending a message that requests specific data from a slave device and the salve responds accordingly, Figure 4.

Note: these final master/slave examples are by no means optimal.

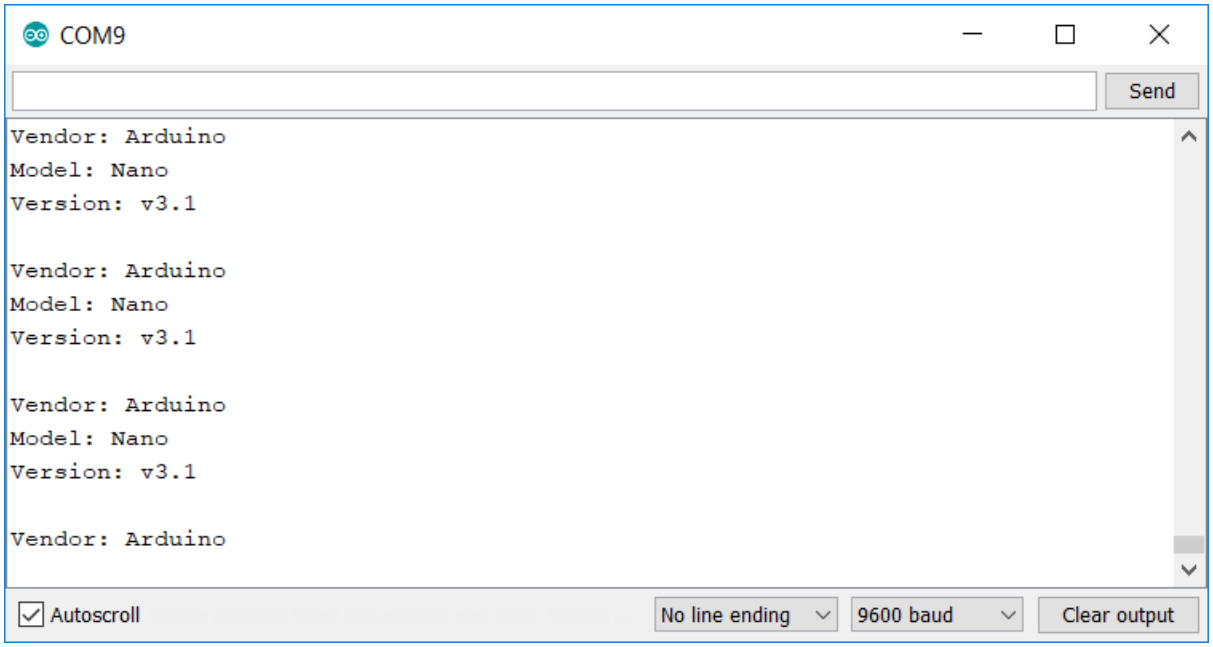

Figure 4 Serial monitor output showing return of requested information from slave device.

Master Send & Receive:

```
#include <Wire.h>

String message_rec;

void setup() {
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}

void loop() {
  reportVendor();
  Serial.print("Vendor: ");
  Serial.println(message_rec);
  delay(500);

  reportModel();
  Serial.print("Model: ");
  Serial.println(message_rec);
  delay(500);

  reportVersion();
  Serial.print("Version: ");
  Serial.println(message_rec);

  delay(500);
  Serial.println();
}

void reportVendor(){
  message_rec = "";
  Wire.beginTransmission(0x08);  // transmit to device #8
  Wire.write("vendor");          // sends six bytes
  Wire.endTransmission();        // stop transmitting

  Wire.requestFrom(0x08, 32);

  while (Wire.available()) {  //loop whilst bus is busy
    char c = Wire.read();     //receive data byte by byte
    if (c == '#') break;      //check for end of message
    message_rec += c;         //build complete message
  }
}
void reportModel(){
  message_rec = "";
  Wire.beginTransmission(0x08);  // transmit to device #8
  Wire.write("model");           // sends five bytes
  Wire.endTransmission();        // stop transmitting

  Wire.requestFrom(0x08, 32);

  while (Wire.available()) {  //loop whilst bus is busy
    char c = Wire.read();     //receive data byte by byte
    if (c == '#') break;      //check for end of message
    message_rec += c;         //build complete message
  }
}
void reportVersion(){
  message_rec = "";
  Wire.beginTransmission(0x08);  // transmit to device #8
  Wire.write("version");         // sends seven bytes
  Wire.endTransmission();        // stop transmitting

  Wire.requestFrom(0x08, 32);

  while (Wire.available()) {  //loop whilst bus is busy
    char c = Wire.read();     //receive data byte by byte
    if (c == '#') break;      //check for end of message
    message_rec += c;         //build complete message
  }
}
```

Slave Receive and Respond:

```
#include <Wire.h>                       //include Wire library

#define SLAVE_ADDRESS 0x08

String message_in, message_out;
char message[8];

void setup() {
  Wire.begin(SLAVE_ADDRESS);            //join i2c bus with address 8
  Wire.onReceive(receiveEvent);         //create a receive event
  Wire.onRequest(requestEvent);         //create a receive event
  Serial.begin(9600);                   //start serial for output
}

void loop() {
}

void receiveEvent() {
  message_in = "";
  while (Wire.available()) {            //loop whilst bus is busy
  char c = Wire.read();                 //receive data byte by byte
  message_in += c;                      //form complete string
  }

  Serial.println(message_in);

  delay (500);
}

void requestEvent() {
  if (message_in == "vendor") {
      message_out = "Arduino#";
  }
  if (message_in == "model") {
      message_out = "Nano#";
  }
  if (message_in == "version") {
      message_out = "v3.1#";
  }

  int buffer = message_out.length();

  message_out.toCharArray(message, buffer + 1);

  Serial.println(message);

  Wire.write(message);
}
```

References:

[1] https://www.arduino.cc/
[2] https://en.wikipedia.org/wiki/Arduino
[3] https://en.wikipedia.org/wiki/I%C2%B2C
[4] https://www.nxp.com/docs/en/application-note/AN10216.pdf
[5] https://www.i2c-bus.org/specification/
[6] https://www.arduino.cc/en/Reference/Wire
[7] https://www.arduino.cc/en/Main/Software
[8] https://store.arduino.cc/arduino-nano
[9] https://learn.adafruit.com/i2c-addresses/the-list