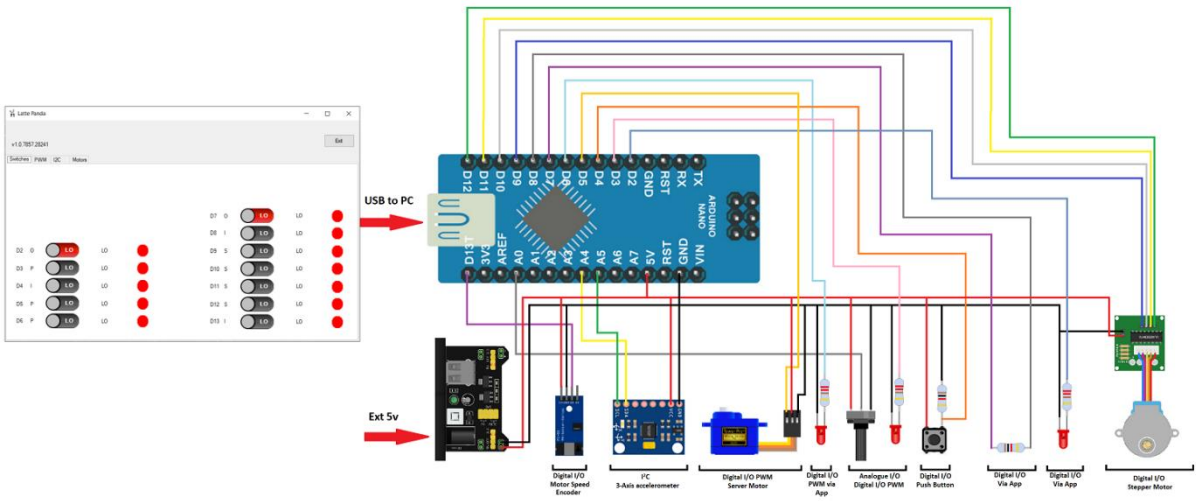
Arduino Firmata Demo



A collection of the notes I made during my exploration Firmata on Arduino

Knowledge is a process of piling up facts; wisdom lies in their simplification.
Martin H Fischer

1. Table of Contents

2. Introduction

One of the drawbacks of the LattePanda is that for development it uses a slow processor which means compiling C# .Net apps takes rather a long time. For this reason, this experiment will look at taking a stand alone Leonardo and couple it with the PC/Laptop. This may allow us to speed up the development. Once developed we can put the code into the LattePanda and run.

The intention was to use a stand-alone Leonardo but unfortunately, we do not have one to hand but do have many Nanos so we will use one of them. We will first walk you through using each part of the final board and explain how to make it work. After we have done all that we will put it altogether on one Breadboard and show it all working.

3. Links

3.1. Firmata

- Official Firmata source = https://github.com/firmata/arduino
- Arduino C# parts = https://github.com/LattePandaTeam/LattePanda-Development-Support
- .Net port = https://github.com/SolidSoils/Arduino
- VB info = http://www.acraigie.com/programming/firmatavb/default.html

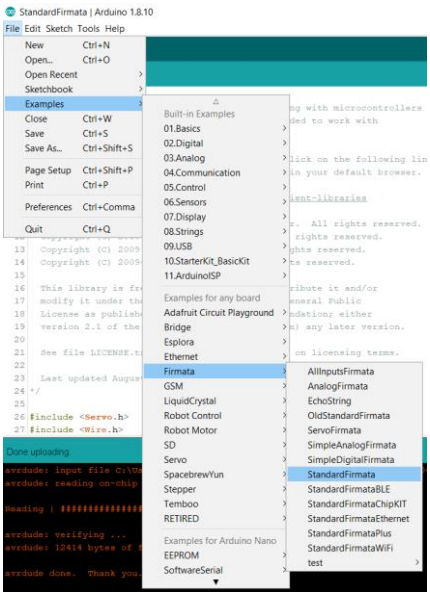
3.2. Arduino Signal Speed

This is useful so you know how many signals you get send or receive from each pin.

- https://learn.sparkfun.com/blog/1687
- https://www.codeproject.com/Articles/732646/Fast-digital-I-O-for-Arduino

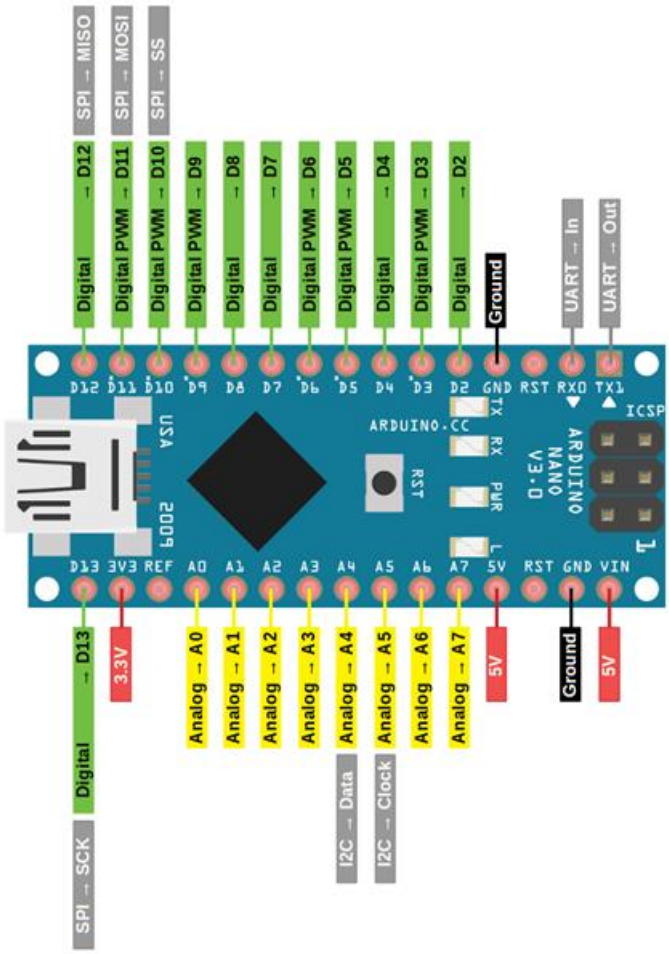
3.3. Stepper Motor Firmata

When trawling the internet looking for any Firmata Stepper motor support we came across the following https://sourceforge.net/p/firmata/mailman/message/29916081/ we found some of the links on the website do not work so we never tried it. For our implementation we use Standard Firmata which comes with the Arduino IDE as shown below.



4. Nano pinouts

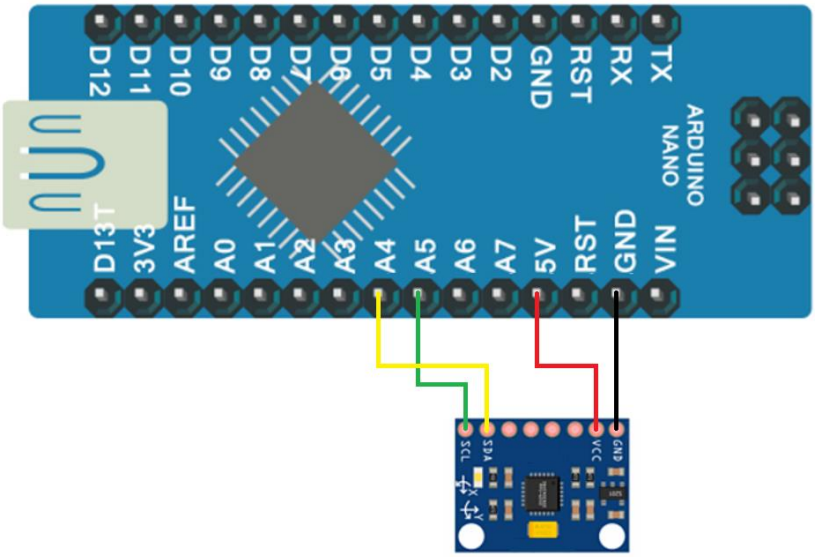
I orientate the Nano the same way as it appears on our breadboard. The image tells us the meaning of each pin especially of interest are those that are Digital providing us with PWM. We will also use I^2C and the pins for Clock and Data are also shown.



5. Using I²C

There are many different modules I could use here but as I had some ADXL345 3-axis accelerometers to hand from a previous project I used them. This is a very easy module to interface in I²C. In this demo we will just read the data returned we will at this stage not do anything with it except display the raw data.

5.1. Circuit Diagram



5.2. App

We first need to declare and assign functionality to pins. You may need to consult the data sheet for you components as they not be the same as I use here. The I²C address of the module I used is 53_{16}. You will need to check this on the one you use.

For the module I used the data sheet of the chip says before you get a reading you need to wake it up. This you do by sending binary 1000_2 to the register $2D_{16}$.

```
arduino.wireRequest(0x53, 0x2D, new Int16[] { 8 }, Arduino.I2C_MODE_WRITE);
```

Once woken up we can take a reading. This time we read 6 bytes of information from register 32_{16}. We read it continuously so it never goes back to sleep.

```
arduino.wireRequest(0x53, 0x32, new Int16[] { 6 }, Arduino.I2C_MODE_READ_CONTINUOUSLY);
```

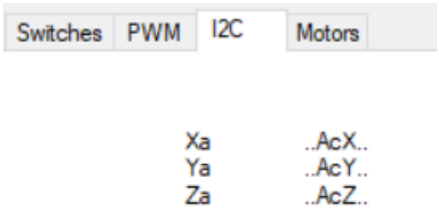
Set up an event to capture the data that is returned. Look in Arduino.cs if you need more info on how to do this.

```
arduino.didI2CDataReveive += Arduino_didI2CDataReceive;
```

Now when any data is received from I²C we will see it displayed in the screen. We use a flag to switch it off when we close the window so as it does not fall over trying to read an object which has been disposed of. As this is read on a thread that is not the same one as the UI is on, we need to invoke it to prevent cross threading.

```
private void Arduino_didI2CDataReceive(byte address, byte register, byte[] data)
        {
            if (I2CFlag)
            {
                Invoke(new Action(() =>
                {
                    lbl_AcX.Text = (BitConverter.ToInt16(data, 0)).ToString();
                    lbl_AcY.Text = (BitConverter.ToInt16(data, 2)).ToString();
                    lbl_AcZ.Text = (BitConverter.ToInt16(data, 4)).ToString();

                }));
            }
        }
```

The read data is just displayed in the tab as simple labels which get updated.



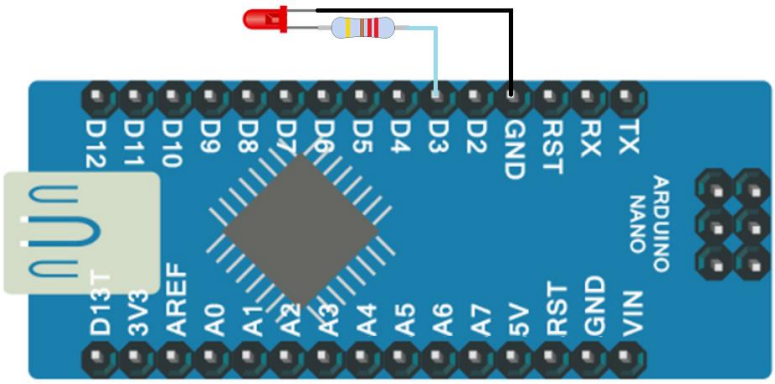
5.3. Conclusion

We can read data from I2C. We note that we may need to consult the chips data sheet to find out how to set and read it.

6. PWM driven by App

For this part we will drive an LED on the breadboard from the Win32 App. We add a slider to the app and by moving the slider we are able to change the brightness of the LED on the breadboard.

6.1. Circuit Diagram



6.2. App

We add a slider:

Then assign the PWM functionality to D3

```
arduino.pinMode(3, Arduino.PWM);
```

and create an event for when the bar is scrolled

```
private void trkbar_pwm_Scroll(object sender, EventArgs e)
        {
            TrackBar bar = (TrackBar)(sender);
            arduino.analogWrite(3, bar.Value);
            lbl_pwm_slider.Text = bar.Value.ToString();
        }
```

6.3. Conclusion

Now when you scroll the bar the LED will get brighter or dimmer.

7. Servo Motor

When using a Servo Motor with an Arduino we will need to use an external power supply that is around 5v but not exceeding 5,5v. If you draw too much current from the Arduino you can damage it and likely tell tales we have found when of drawing too much current is that some signals start to float.

The servo we will use is a small one called the SG90. It has a full 180° rotation. There are several common wiring colour schemes used in Servo Motors and these are shown below. The colour scheme of the SG90 is that of JR.

Pin	Signal Name	Colour Scheme 1 (Futaba)	Colour Scheme 2 (JR)	Colour Scheme 3 (Hitec)
1	Ground	Black	Brown	Black
2	Power Supply	Red	Red	Red or Brown
3	Control Signal	White	Orange	Yellow or White

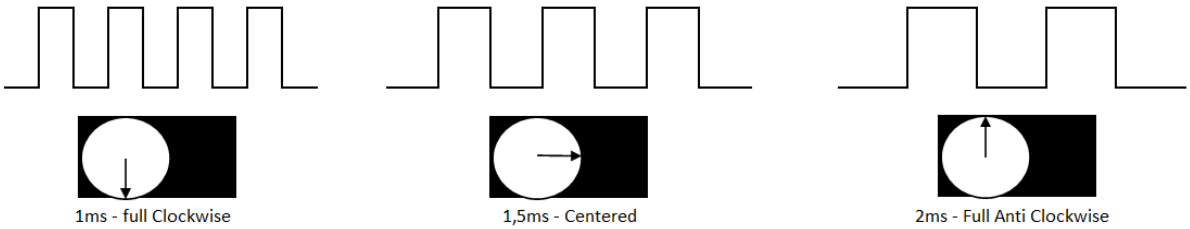
7.1. How a Servo Motor works

A servo motor will rotate a full 180°. When the servo is initialised, it will start in the middle of the range. The third wire is used as the control wire this is where the Arduino will send its signals to control the servo.

The control signals are pulsed out in a train 20ms (50 Hz) intervals and vary between 1ms and 2ms in width. The Pulse Width Modulation hardware on the Arduino is used to generate the servo control signals. There are several PWM pins available on the Arduino.

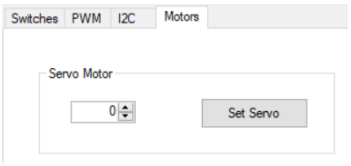
The pulses vary in width depending on where the Servo is in its rotation range. The accuracy of the Servos position is dependent on the accuracy of the signals sent from the Arduino. Below we can see how range from 1ms to 2ms will position the servo to the correct position. The start

position of the servo you use may differ from that which is shown below but the principle will be the same.



7.2. Windows App

In the Win32 App there is a tab called Motors. In this tab you will find a group box with the Servo Motor Settings in it. Put the degrees you would like the servo to be set to and click the button. The servo will then go to that position. If it is already at the position, then it will remain there. Moving it a degree or two may not do anything depending on the accuracy of the signals from the PWM.



When the button is clicked, we start an event to set the value in the servo

```
private void btn_set_servo_Click(object sender, EventArgs e)
        {
            int angle = Convert.ToInt32(Math.Round(numupdwn_degrees.Value, 0));
            arduino.servoWrite(9, angle);//tell the servo motor go to the position
        }
```

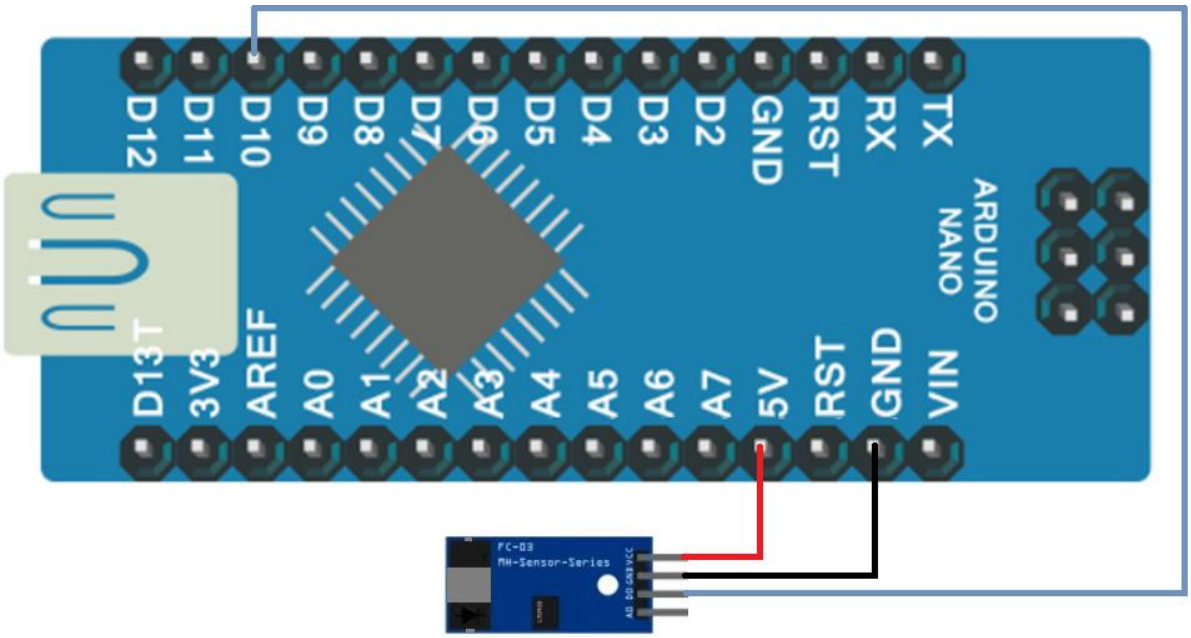
7.3. Conclusion

With the external power supply, we are able to set the servo to the correct value. If the voltage drops too low, then we get some floating of signals and the servo will keep hunting. The accuracy of the angles was not measured but in the Anti-clockwise direction they did seem to be slightly out. It will require more work to investigate this and this will be part of a different project.

8. Encoder

Encoders are used to sense movement. I have written many documents on encoders so please refer to them for more information. In this section we will use the LM393 Speed Sensor Module to tell us when the encoder has moved. We will only find out if the encoder has moved. The module adds a beam to the LM393 chip and all circuitry needed to read when something passes by it.

8.1. Circuit Diagram



8.2. Windows App

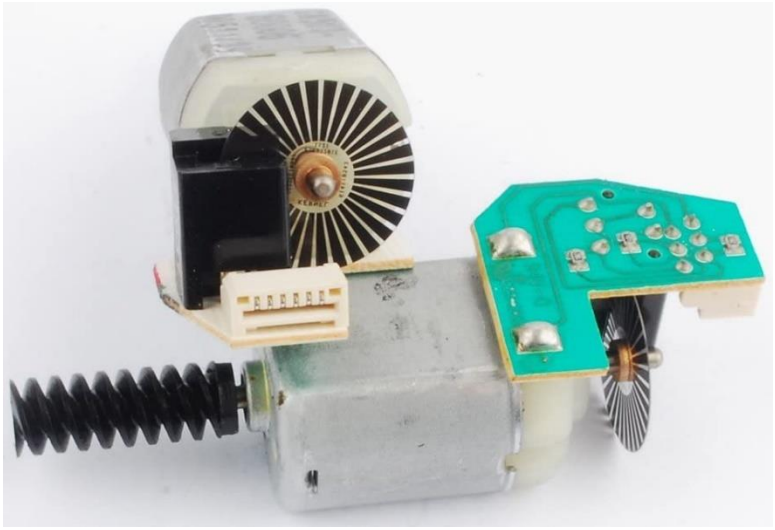
In the app we have an event that is triggered when a digital pin is changed. If we find the pin that changed is D10 then we will add one to the number.

Below you will see a Flag, this is added because we found that the sensor always gives two for every time we go past the sensor once. We did not look further into this part as we only wanted to see if this works which it does.

```
if (ActivePin == "10") //motor speed
  {
      if (MotorSpeedFlag)
      {
          MotorSpeedFlag = false;
      }
      else
      {
          count++;
          MotorSpeedFlag = true;
      }

      lbl_motor_speed.Text = count.ToString();

  }
```

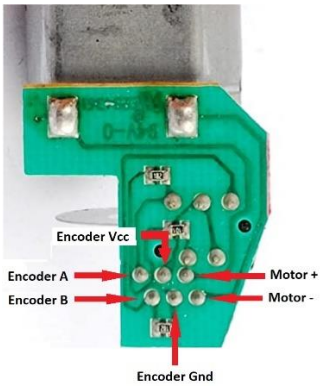
8.3. Motor with Encoder

In my box of bits is a small motor with an encoder on the end. We will try and see if we can make this work with the Nano. After lots of searching on the internet I think this is called: Johnson Standard 130 motor Green Micro DC motor with encoder



8.3.1. Connections

First, we need to work out how to wire this. Nothing is marked on the small PCB but after a few tests we work out the following:



The encoder disk has 32 markings on it, this means that we get 32 pulses per revolution. Our software will count the pulses and issue one out every 32 which will mean we have done one revolution.

We can get both A and B signals so we could tell if the motor was turning clockwise or Anti-clockwise. In our tests we will not need to use this so we will just use one signal.

8.4. Results

Running between 5.11v and 5.13v we got an average of 2650rpm. The Nano seemed to be happy getting around 84800 signals a minute or 1413 a second. This is far more than we would need.

9. Nano I/O speeds

One part that is of interest to us is the speed with which we can get data via the I/O pins through the Arduino across Firmata and into Windows App. As you can see there are several parts to this with some being non-deterministic. They are:

Time taken to

- React to change in I/O pin state.
- Atmel microprocessor to deal with the data coming in.
- Firmata to package up packet of data.
- Get data across RS232 from Arduino to Windows (Baud Rate).
- Windows reaction time to incoming data.
- App to process the data and display it on the screen.

People have tested the Nano and found that it takes 5µs to read a digital signal which means the Nano could deal with 200 000 signals a second or 12 000 000 signals a minute. We know our disk has 32 signals on it which means we can run at a max of 375 000rpm. From the Nano we would need to feed the signals to the PC/Laptop and that would take time.

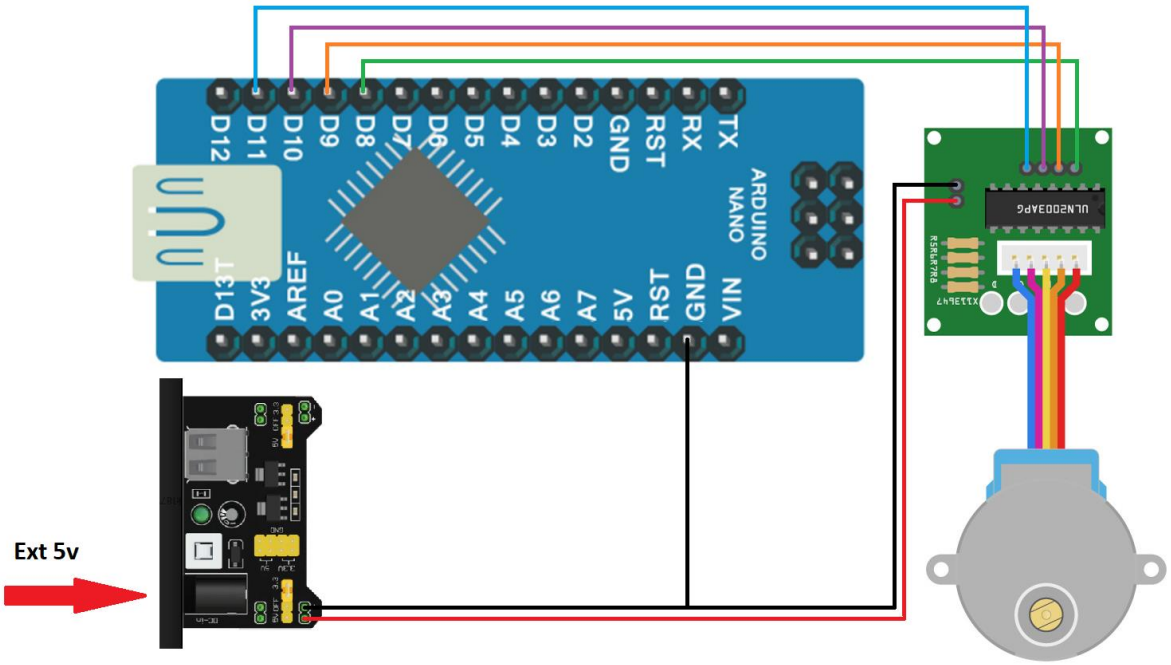
The motor we have when supplied with 5v will reach 2 650rpm. Which is 84 800 pulses and the system we built seems to be able to cope with that.

Another website @ https://learn.sparkfun.com/blog/1687 says max I/O write is 117 000Hz, we did not check this in our tests so cannot say if this is true or not.

10. Stepper Motor v1

In the first example we will use a sketch to drive the stepper motor. Our aim will be to later use Firmata to drive it so we can use it via the LattePanda.

10.1. Circuit Diagram



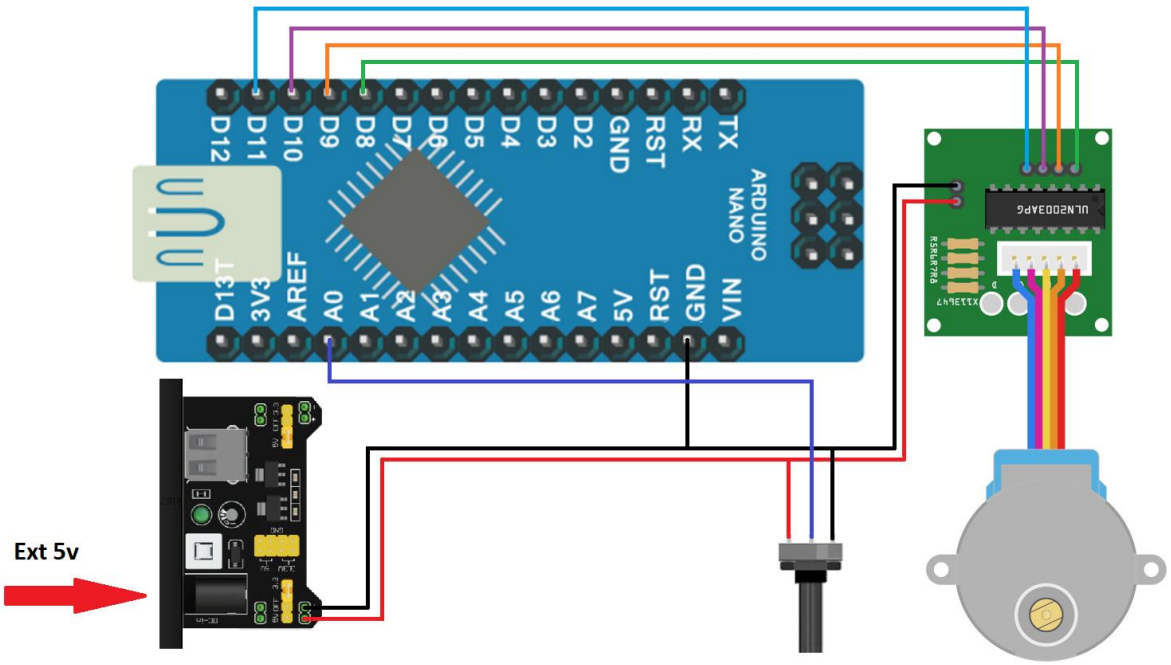
10.2. Code

```
// Include the Arduino Stepper.h library:
#include <Stepper.h>

// Define number of steps per rotation:
const int stepsPerRevolution = 2048;

// Wiring:
// Pin 8 to IN1 on the ULN2003 driver
// Pin 9 to IN2 on the ULN2003 driver
// Pin 10 to IN3 on the ULN2003 driver
// Pin 11 to IN4 on the ULN2003 driver

// Create stepper object called 'myStepper', note the pin order:
Stepper myStepper = Stepper(stepsPerRevolution, 8, 10, 9, 11);

void setup() {
  // Set the speed to 5 rpm:
  myStepper.setSpeed(5);
}

void loop() {
  // Step one revolution in one direction:
  myStepper.step(stepsPerRevolution);
  delay(500);

  // Step one revolution in the other direction:
  myStepper.step(-stepsPerRevolution);
  delay(500);
}
```

11. Stepper Motor v2

In the first example we will use a sketch to drive the stepper motor and include a 10kΩ potentiometer to control the speed.

11.1. Circuit Diagram



11.2. Code

```
int motorPins[] = {8, 9, 10, 11};
int count = 0;
int count2 = 0;
int delayTime = 500;
int val = 0;

void setup() {
  for (count = 0; count < 4; count++) { pinMode(motorPins[count], OUTPUT); }
}

void moveForward() {
  if ((count2 == 0) || (count2 == 1)) { count2 = 16; } count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[count], count2>>count&0x01);
  }
  delay(delayTime);
}

void moveBackward() {
  if ((count2 == 0) || (count2 == 1)) {
    count2 = 16;
  }
  count2>>=1;
  for (count = 3; count >= 0; count--) {
    digitalWrite(motorPins[3 - count], count2>>count&0x01);
  }
  delay(delayTime);
}

void loop() {
 val = analogRead(0);
 if (val > 540) {
    // move faster the higher the value from the potentiometer
    delayTime = 2048 - 1024 * val / 512 + 1;
    moveForward();
  } else if (val < 480) {
    // move faster the lower the value from the potentiometer
    delayTime = 1024 * val / 512 + 1;
    moveBackward();
  } else {
    delayTime = 1024;
  }
```

}

12. Using Firmata Protocol to run the Stepper motor.

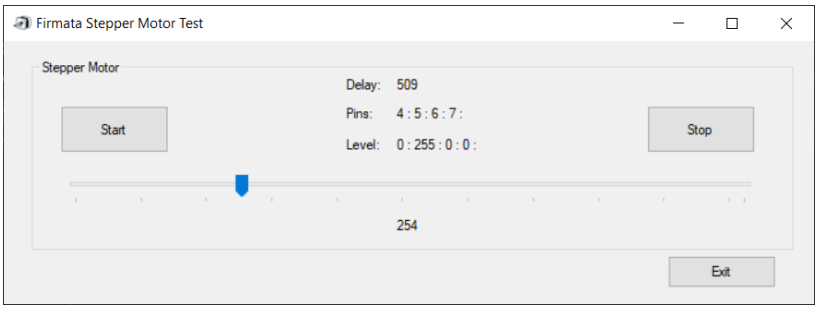
We could not find any Firmata support for Stepper Motors but will try using the Standard Firmata to run a Stepper Motor to see how much control we have. We will do this by setting the 4 stepper motor digital lines high or low and hold them there for a specified time. This will cause the motor to rotate.

12.1. Arduino Side

We use an Arduino Nano for this, but you can use other types. Upload the Standard Firmata Sketch to the Nano. The sketch is part of the built-in libraries that come with the Arduino IDE.

12.2. C# .Net Code

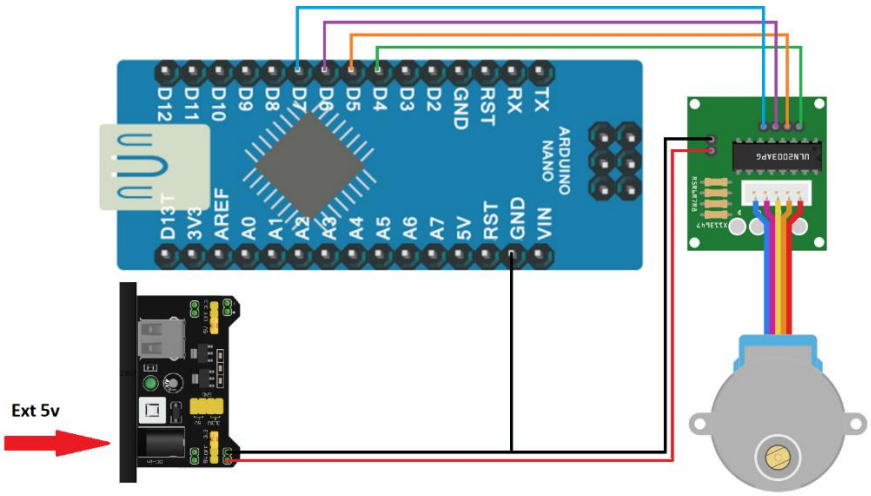
We have created a small app that will allow us to connect to the Arduino with the Firmata Protocol and to make it run forwards and backwards at different speeds. On the UI we can see the signal levels and if you will notice the high (255) signal will go clockwise and anti-clockwise depending on the position of the slider.



12.3. Power Supply

As the Stepper motor draws more current than the Nano can supply it is advised to attach an external power supply. Depending on your circuitry it may be that you can use a power supply that is different from that shown in the wiring diagram.

12.4. Wiring Diagram



12.5. Putting it all together

The aim is to show in simple terms how to use the Firmata protocol in an Arduino and driving it via a C# .Net Win32 app. We load into the Arduino the Standard Firmata. There are other versions of Firmata you could load and some may be better for some features than others. After reading this you will be able to work out how to make this work with most versions of Firmata.

We will show how you can achieve the following actions in very basic ways:

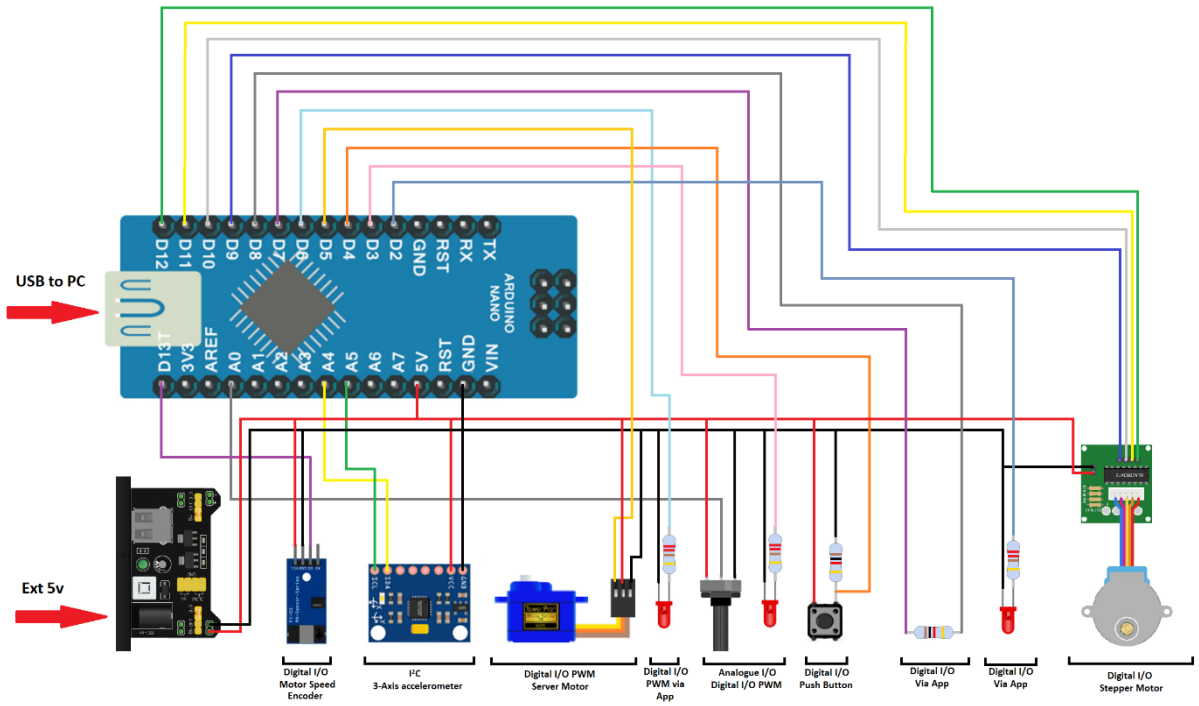
- Drive a digital I/O line
- Get Data from a Digital I/O line
- Drive pulse width modulation from a Digital I/O line
- Drive a Servo Motor
- Drive a Stepper Motor
- Read an Encoder
- Used I^2C protocol.
- Using Analogue I/O
- Use the C# .Net Winforms GUI to display data from the Arduino
- Use the C# .Net Winforms GUI to drive the Arduino via Firmata protocol

12.5.1. Demo Video

A small video showing this in operation can be found @ https://youtu.be/98JzrHrhqww

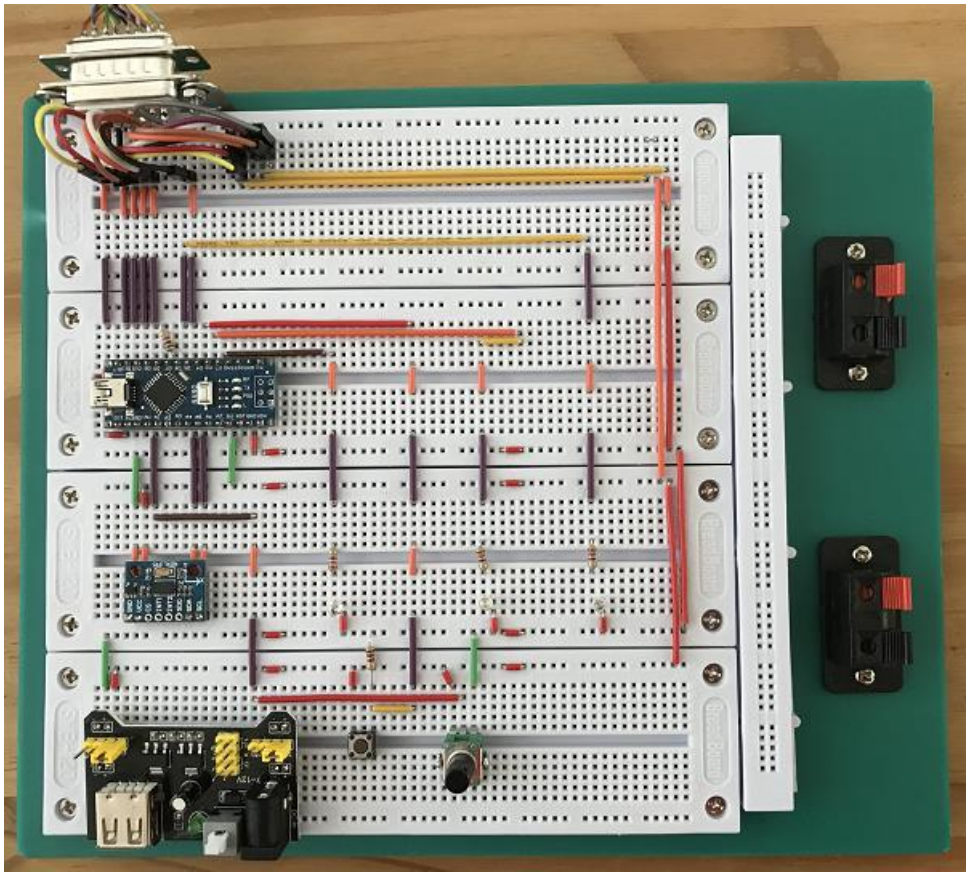
12.5.2. Circuit Diagram

Below we see the top-level schematic showing how everything is connected together. As we used all the digital pins on the Arduino, we needed to be especially careful to get the correct type of pin connected to the current piece of equipment.



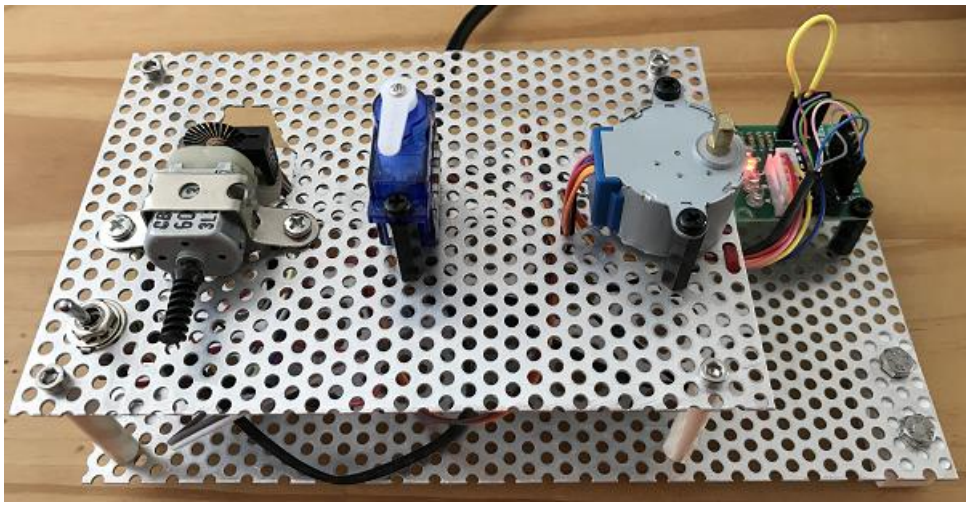
12.5.3. Breadboard layout

There are two parts to the project. One is put on a breadboard as shown below and the other is the motor board which is shown later. The two are tied together via a cable.



12.5.4. Motor Board Layout

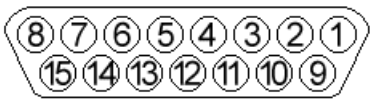
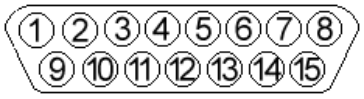
This is where the motors are mounted and run. There is a 15-way D-type cable that joins this motor board to the breadboard.



12.5.5. Pinouts of the Connecting cable

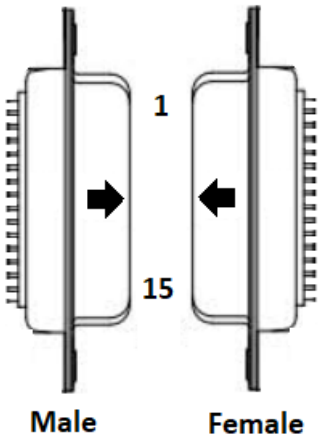
Here we are looking at the connectors from the front which is the side the two will join on.

Take care when looking at the rear of the connectors as they reversed.

		
Male Pin Numbers		Female Pin Numbers

1 = Encoder input to D13
2 = LN1 of Stepper Motor from D12
3 = PWM output to Servo Motor from D5
4 = LN4 of Stepper Motor from D10
5 = LN3 of Stepper Motor from D9
6 = LN2 of Stepper Motor from D11
7 = Blank not used
8 = Gnd: Encoder Motor
9 = Gnd: Encoder Sensor
10 = Gnd: Stepper Motor
11 = Gnd: Servo Motor
12 = Vcc: Encoder Motor
13 = Vcc: Encoder Sensor
14 = Vcc: Stepper Motor
15 = Vcc: Servo Motor

Orientation of the connectors is shown below:



12.5.6. Arduino Pinouts

The following details the pins used:

- D2 = Digital Output from App to light LED
- D3 = Digital PWM Output to LED from Potentiometer
- D4 = Digital Input triggered by pressing button.
- D5 = Digital PWM Output to Servo Motor
- D6 = Digital PWM Output to LED from App Slider
- D7 = Digital Output from App to trigger D8
- D8 = Digital Input to App triggered from D7
- D9 = Digital output to Stepper Motor Coil 4
- D10 = Digital output to Stepper Motor Coil 3

- D11 = Digital output to Stepper Motor Coil 2
- D12 = Digital output to Stepper Motor Coil 1
- D13 = Digital input from Encoder
- A0 = Analogue Input from Potentiometer
- A4 = SDA data line for I^2C
- A5 = SCL clock line for I^2C
- 5v = Power supply to the board
- GND = ground for the board

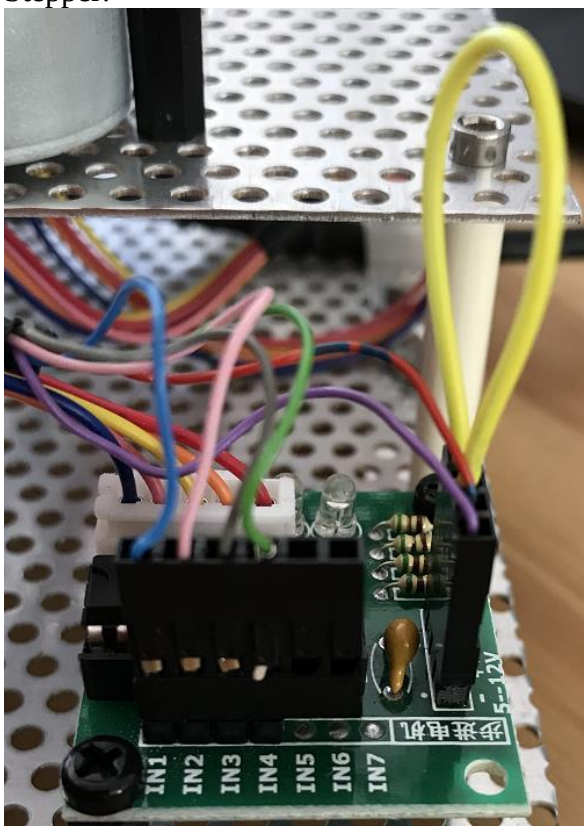
To enable us to drive the Motors we have an additional external power supply which is regulated down to 5v.

12.5.7. Motor Board Wiring

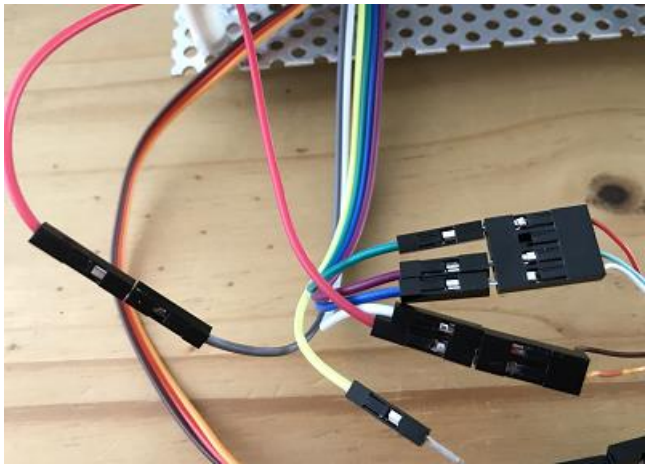
We just include pictures of our wiring for our own future reference. Your wiring may be different.

Servo:

Stepper:

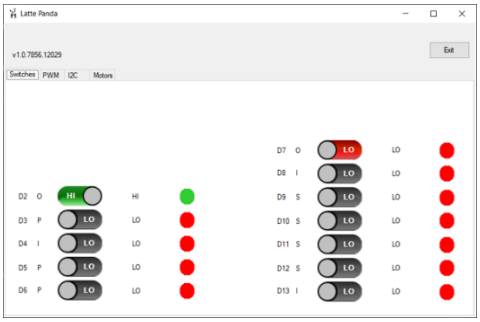


Encoder:



12.5.8. Digital Output via app

On the GUI is a table of buttons, that look like switches, one for each digital line on the Arduino Nano.



If the button is checked then the app sends the following

```
arduino.digitalWrite(1, Arduino.LOW);
```

If the button is unchecked then it sends

```
arduino.digitalWrite(0, Arduino.HIGH);
```

Note that you send either High or Low depending on whether the circuit is PNP or NPN.

12.5.9. Digital Input from Push Button

All digital pins when activated will cause an interrupt of the app and when this happens the following function will be activated with the Pin that caused the interrupt and its state of High or Low.

```
private void Arduino_digitalPinUpdated(byte pin, byte state)
```

If we sense the pin is from the pushbutton then we will change the switch on the GUI to shw it is activated.

12.5.10. Digital input from Encoder

This is the same as the Push Button but here we will count the number of pulses received. With the encoder you can have more that one signal. With two signals you can determine of the

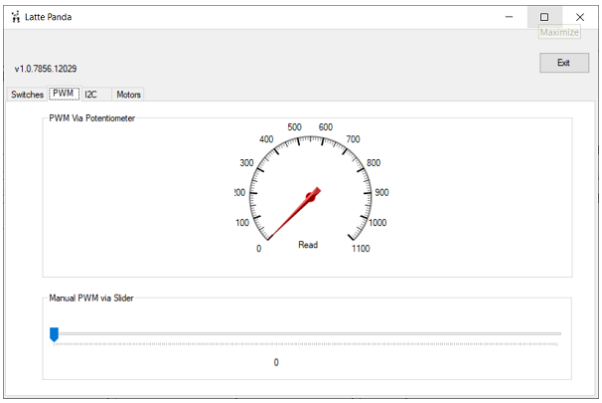
encoder is going forwards or backwards. In our example we simply count the revolutions of the motor. Our encoder gives us 32 pulses revolution^{-1}.

```
lbl_motor_speed.Text = (count / 32).ToString();
```

This value is displayed to the GUI Motor tab in the group as shown below. You can reset the count if needed.

12.5.11. Digital PWM output via App

On the GUI you will find under the PWM tab and slider bar. If you move this bar to the right the LED on the board will start to brighten. The further to the right you go the brighter the LED will become.



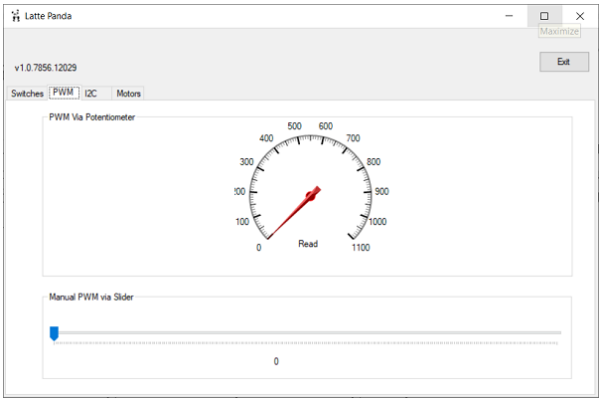
The code behind the scrollbar interrupt is as follows.

```
private void trkbar_pwm_Scroll(object sender, EventArgs e)
        {
            TrackBar bar = (TrackBar)(sender);
            arduino.analogWrite(6, bar.Value);
            lbl_pwm_slider.Text = bar.Value.ToString();
        }
```

12.5.12. Digital PWM output via Analogue input

On the Board you will find a potentiometer. When this is turned you will see an LED change its brightness. The Potentiometer sends a signal into the A0 pin and the voltage read there determines the PWM that gets send out the Digital pin to the LED.

When the Potentiometer is turned you will get an indication on the GUI which has a dial which will turn.



The code behind this is as follows, you can see how we change the input value to an output value to be sent the LED attached to a digital pin via PWM.

```
private void Arduino_analogPinUpdated(int pin, int value)
        {

            if ((pin == 0) && (AnalogueFlag))
            {
                Invoke(new Action(() =>
                {
                    lbl_read.Text = value.ToString();
                    aGauge1.Value = value;

                    arduino.analogWrite(3, (value / 4)); //div by 4 as input 0 to 1023 but output 0 to 255.
                    Thread.Sleep(4); //delay 4ms to let it settle.
                }));
            }
        }
```

12.5.13. I²C protocol

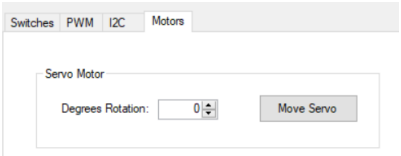
Here we attach an ADXL345 3-axis accelerometer to the board and we get back from it the data telling us its orientation. We then display this raw data on the GUI to show that I²C is working.

This data is received via an interrupt as follows:

```
private void Arduino_didI2CDataReceive(byte address, byte register, byte[] data)
        {
            try
            {
                Invoke(new Action(() =>
                {
                    lbl_AcX.Text = (BitConverter.ToInt16(data, 0)).ToString();
                    lbl_AcY.Text = (BitConverter.ToInt16(data, 2)).ToString();
                    lbl_AcZ.Text = (BitConverter.ToInt16(data, 4)).ToString();

                }));
            }
            catch (Exception e)
            {
                //carry on
            }

        }
```

12.5.14. Driving Servo Motor

Under the motor tab we will find a section entitled “Servo Motor”. Here we can set the degrees to rotate to and then press the button to move to that position.



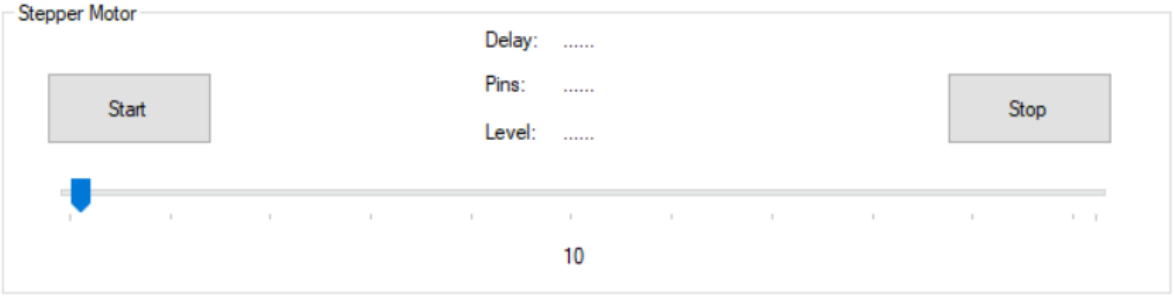
Here is the code behind the button. The function could all be written to a single line but I do it in two just to allow you to see where the angle comes from.

```
private void btn_set_servo_Click(object sender, EventArgs e)
        {
            int angle = Convert.ToInt32(Math.Round(numupdwn_degrees.Value, 0));
            arduino.servoWrite(5, angle);//tell the servo motor go to the position
        }
```

12.5.15. Driving Stepper Motor

Unfortunately, the Standard Firmata we are using does not have explicit support for Stepper Motors. As we know how a stepper motor expects to get its code, we can manipulate the app to send the data in a way that will make the motor move. This is only an illustration and more work would be needed to make it useful. There may well be versions of Firmata out there that do fully support Stepper Motors.

On the GUI under the Motors tab you will see the stepper motor group. In the group there is a slider. On the left the motor will rotate in one direction at speed and when on the right it will turn on the opposite direction at speed. In between the speed will slow. There are buttons to start and stop the action.



We make this work but getting a delay value from the value of the slider. You can change the algorithm to make the motor run faster.

```
delayTime = (2048 - ((1024 * val1) / 512)) + 1;
```

When then take this delay and use it in the function to run the motor forwards or backwards. Below we only show moving the motor forward as reverse is almost identical that you will be able to work it out in the code.

```
void moveForward()
 {
     byte valued;
     lbl_stepper_pins.Invoke((MethodInvoker)(() => lbl_stepper_pins.Text = ""));
     lbl_stepper_signal.Invoke((MethodInvoker)(() => lbl_stepper_signal.Text = ""));

     if ((countB == 0) || (countB == 1))
     {
         countB = 16;
     }

     countB >>= 1;

     for (countA = 3; countA >= 0; countA--)
```

```
        {
            valued = Convert.ToByte(0); //low
            if ((countB >> countA & 0x01) == 1) valued = Convert.ToByte(255); //high

            arduino.digitalWrite(motorPins[countA], valued);
            lbl_stepper_pins.Invoke((MethodInvoker)(() => lbl_stepper_pins.Text += motorPins[countA] + " : "));

            lbl_stepper_signal.Invoke((MethodInvoker)(() => lbl_stepper_signal.Text += valued + " : "));
        }

        Task.Delay(delayTime).Wait();
    }
```

The speed the motor turns is determined by how long we set the delay for as this is the length of time that we will put the power onto the windings.