Adventures with Latte Panda

A Windows 10 Development Board for Everything

A collection of the notes I made during my exploration with Latte Panda

1. Table of Contents

2. Introduction. 4
3. History 4
4. Why is LattePanda useful 4
5. Which one will we use? 4
   5.1. Specification 4
6. Pinouts 5
   6.1. Analog Inputs 6
   6.2. RS232 6
   6.3. External Interrupts 6
   6.4. PWM 6
   6.5. SPI 6
   6.6. LED 6
   6.7. TWI 6
   6.8. Other pins on the board 6
      6.8.1. Reset 6
7. Setting up the Development Environment 6
   7.1. Putting Win10 Home into developer mode 6
   7.2. Visual Studio IDE 7
   7.3. Board Support Package (BSP) 8
   7.4. Arduino IDE 8
8. Firmata 8
9. Windows Remote Arduino Experience 8
10. Win32 C# .Net App 10
11. Project 1 – Temperature and Humidity via Arduino 11
   11.1. DHT11 temperature & humidity sensor 11
   11.2. Code 12
12. Project 2 – Temperature and Humidity via C# .Net 13
   12.1. Arduino Code 13
   12.2. C# .Net App explanation 14
13. Project 3 – Controlling I/O to set signals 18
   13.1. Windows App 18
   13.2. Hardware Circuit 18
   13.3. Arduino code 19
   13.4. Running the Windows App. 20
   13.5. Conclusions 21
14. Project 4 – Get and Set Digital I/O signals Via Windows app 22

14.1. Circuit Diagram using Switch 22
14.2. Circuit Diagram using I/O 24
14.3. Conclusions 25
15. Project 5 – Get and Set Analogue I/O signals Via Windows app 26
16. AnalogRead...27
17. PWM 29
18. Project 2b – Temperature and Humidity...32
19. Project 2c – Using Visual Studio to set and unset signals. 34
20. Optocoupler Isolator 35
20.1. Connections 35
20.2. Basic Schematic Circuit...35
21. PC817, 4 channel opto-isolator breakout for Arduino opto-isolator optocoupler ST...36
22. Parts for starter kit...37

2. Introduction.

The point of this exploration is to see how usable the Latte Panda will be for customer applications. What we would like to have is a set of modules written in C# .Net that customers can use as starting blocks in their applications. The Latte Panda incorporates an Arduino Leonardo which will also allow for I/O signals to be used which can be directly addressed through the C# code.

3. History

LattePanda started out as a Kickstarter project based in Shanghai in 2015. It was trying to make a small Windows based SBC that could be used for IOT. They have an official website @ https://www.lattepanda.com/.

4. Why is LattePanda useful

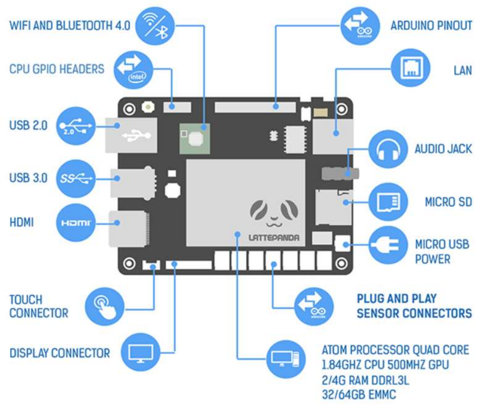
In my experience there are several types of integrators of our products in the world. In some countries it is not easy to find a software engineer who is able to talk to a device other than a PC. In some country's their applications will also not allow for the cost of a full integration so we need to help provide cheaper solutions.

I have found that many times when a software engineer takes on an integration task with our printers they just need to get started and then they are away on their own. I find that most of these engineers are writing applications in C# .Net so to help them I create small modules in C# .Net. These are "starter" projects which they take and manipulate to do what they need. With a PC it is difficult to get I/O signals to and from the printer.

LattePanda allows us to get I/O signals from the printer and to get and set via C# .Net.

5. Which one will we use?

There are many types of LattePanda available, but we will start to use the "LattePanda V1 - A Powerful Windows 10 Mini PC 2GB/32GB". A very quick overview is shown below.



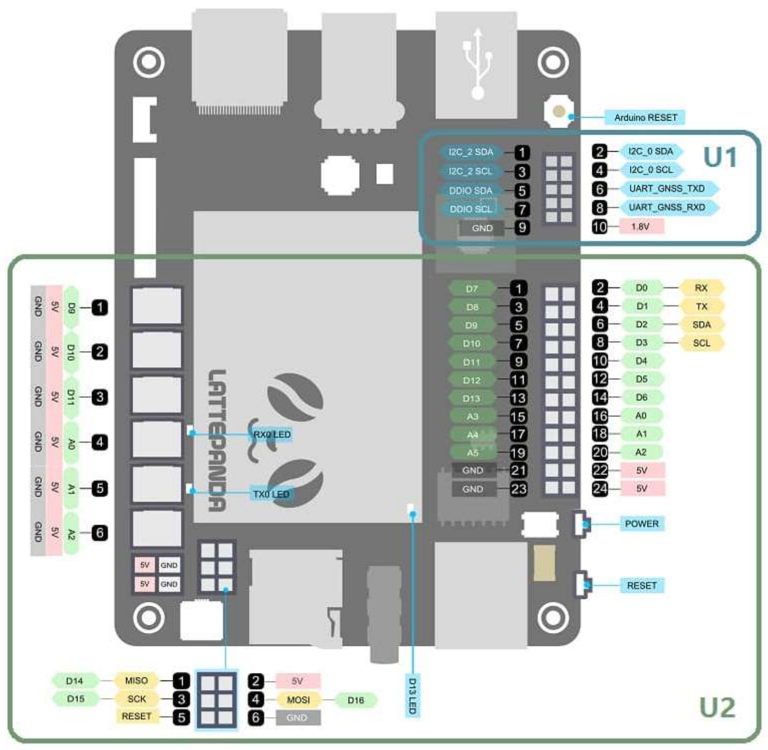
5.1. Specification

- Processor: Intel Cherry Trail Z8300 Quad Core 1.8GHz
- Operation System: Pre-installed pre-activated full edition of Windows 10 (Home version)
- Ram: 2/4GB DDR3L
- Storage Capability: 32/64GB

- USB: 1 x USB 3.0, 2 x USB 2.0
- HDMI Video Output and Ethernet port
- 3.5mm Audio Output Jack
- Micro SD Card Slot
- Touch and Display Connector
- Plug and Play Sensor Connectors
- WiFi and Bluetooth 4.0
- Co-processor: ATmega32u4
- GPIO: 2 GPIOs for Intel chip, 20 GPIOs for Arduino
- Power: 5v/2A
- Dimension: 3.46"x2.76"
- Weight: 100g

6. Pinouts

The Arduino pinouts are shown below:



Pinouts in area U1 are assigned to the X-Z8300 core. At the moment, no more information is available.

Pinouts in area U2 are assigned to the ATmega32u4 core. Each of the 20 digital pins (A0 - A5, D0 - D13) in area U2 can be used as an input or output, each operating at 5 volts. Each pin can output or input 40mA and each has an internal pull-up resistor (disconnected by default) of 20-50k ohm.

<u>Caution:</u> Exceeding 40mA on any I/O pin may cause permanent damage to the ATmega32u4.

Some pins have specialised functions:

6.1. Analog **Inputs**
A0 - A5, A6 - A11 (on D4, D6, D8, D9, D10, and D12). The LattePanda has 12 analogue inputs, labelled A0 through A11, all of which can also be used as digital I/O. Each pin has a 10bit resolution (i.e., 1024 different values). By default, they measure from ground to 5 volts.

6.2. RS232
D0 (RX) and D1 (TX). Used to receive (RX) and transmit (TX) TTL serial data.

6.3. External Interrupts
D3 (interrupt 0), D2 (interrupt 1), D0 (interrupt 2), D1 (interrupt 3) and D7 (interrupt 4). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

6.4. PWM
D3, D5, D6, D9, D10, and D13 provide 8-bit PWM output.

6.5. SPI
D16 (MOSI), D14 (MISO), D15 (SCK).

6.6. LED
D13 There is a built-in LED driven by digital pin 13. When the pin's value is HIGH or LOW

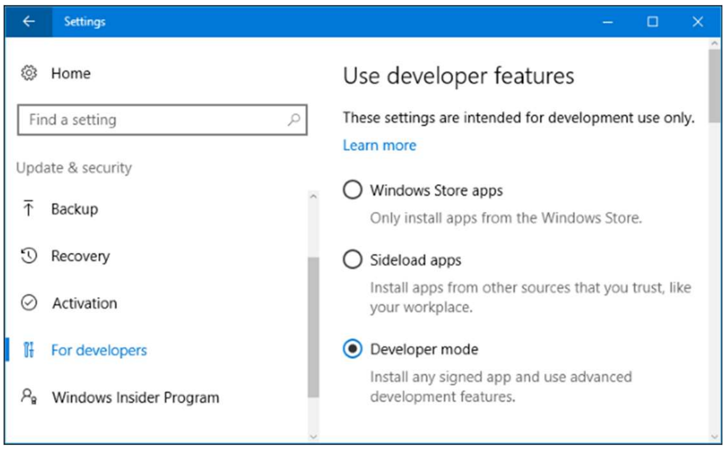
6.7. TWI
D2(SDA), D3(SCL).

6.8. Other pins on the board
6.8.1. Reset
Bring this line LOW to reset the micro-controller. It is typically used to add a reset button to shields which block the one on the board.

7. Setting up the Development Environment
This you only need do once and is very easy to do. Just follow the process below.

7.1. Putting Win10 Home into developer mode
As we will be compiling and debugging our applications on the LattePanda from Visual Studio we will need to put Win10 Home into developer mode. The home addition does not come with this switched on so you will need to carry out this small task first.

Go to the SETTINGS and then UPDATE & SECURITY | FOR DEVELOPERS and then you will find the following menu.



Use developer features

These settings are intended for development use only.

Learn more

Windows Store apps
Only install apps from the Windows Store.

Sideload apps
Install apps from other sources that you trust, like your workplace.

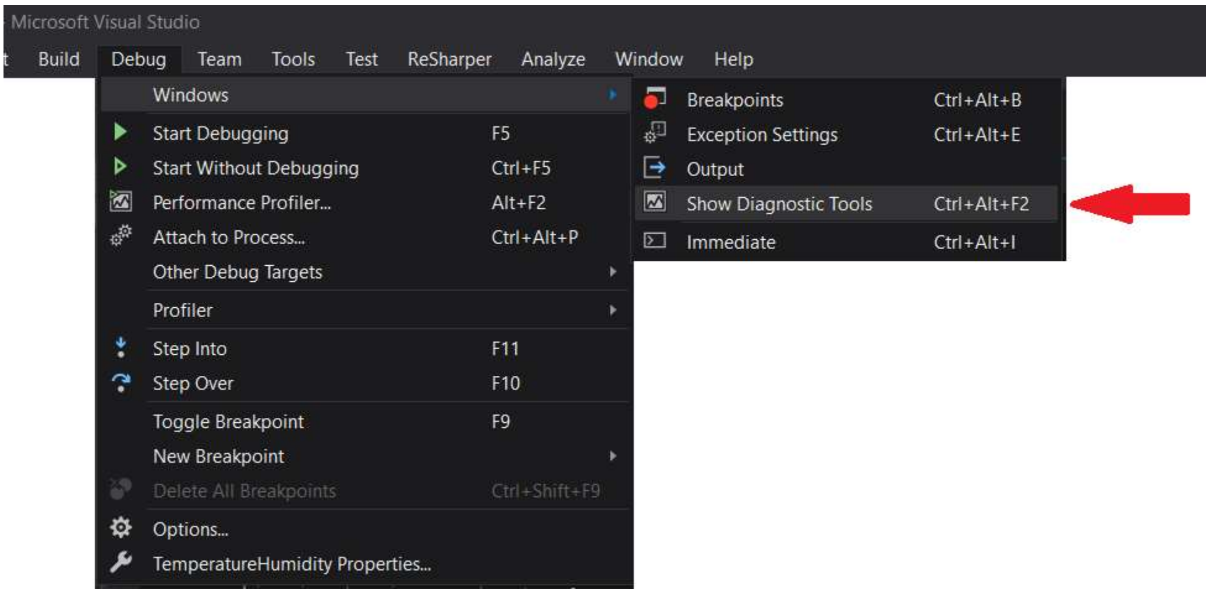
[✓ Developer mode]
Install any signed app and use advanced development features.

7.2. Visual Studio IDE

Next, we need to download Visual Studio onto the LattePanda so we can see if we can run development and deployment on the same device. You can of course run Visual studio on your PC and then transfer the App to the LattePanda, but we will want to show how to debug while on the LattePanda.

Visual Studio is a fully featured, extensible, free IDE for creating modern applications for Android, iOS, Windows, as well as web applications and cloud services. Gone are the days where you need to pay lots of money for a licence as now an Individual can have the community edition of Visual Studio for free, I suggest you read the Licence terms just to make sure you are not illegally using the IDE. I am downloading the 2019 edition from https://visualstudio.microsoft.com/vs/community/. I am keeping this separate from my work PC which has a developer licenced copy of Visual Studio 2017 with Jetbrains.

If you want to run the Diagnostics on your App you create in VS then you will need to make sure that the .Net profiling tool component is installed. By default in VS community edition 2019 this is not installed by default. To check if it is installed you can check by opening the following menu and if the item shows as indicated below then it is installed.



If it does not show then you will need to manually install it by going to TOOLS | GET TOOLS AND FEATURES and ticking it to be added.

7.3. Board Support Package (BSP)

To enable us to control the Arduino from C# .Net we need to download the BSP. Which we can find @ https://github.com/LattePandaTeam/LattePanda-Development-Support

We will find the BSP contains the following

- LattePanda C# Firmata library (Access GPIO from Visual Studio)
- LattePanda C# Firmata Demo
- FirmateWebClient: a hardware control platform that can be accessed on the web

We are interested in getting access to the GPIO from C# .Net so we will use LattePanda.Firmata which is an open-source Firmata library provided by LattePanda. This comes as a class in a file called Arduino.cs and it will allow you to control Arduino GPIO from Windows apps, with features including:

- Reading and writing to digital pins
- Reading analogue inputs
- Controlling servo motors
- Sending data to devices and receiving data form devices through the I2C Bus

You will need to import a copy of the file Arduino.cs into the root of your Visual Studio Solution.

7.4. Arduino IDE

We also need to get a copy of the Arduino IDE. You could use the web-based edition, but I use the windows installed version. Download this from the Arduino website @ https://www.arduino.cc/en/software.

The model of Arduino that we have on the LattePanda is the "Leonardo".

8. Firmata

This is a protocol that is used to communicate between a PC and a microcontroller. It is based on "Midi Message Format" where command bytes are 8 bits and data bytes are 7 bits. More data on the "Midi Message Format" can be found at https://www.midi.org/specifications/item/table-1-summary-of-midi-message

More information on the Firmata implementation of the "Midi Message Format" can be found at the original site which is alive but not updated at http://firmata.org/wiki/Main_Page and the site that is updated is found at https://github.com/firmata/protocol.

There is a specific site where you can get information on the Arduino implementation of the protocol and this is found at https://github.com/firmata/arduino

9. Windows Remote Arduino Experience

There is a Win10 client that you can use to check that Firmata is working on the Arduino. You will need to connect an Arduino that has the Standard Firmata library loaded in it. The

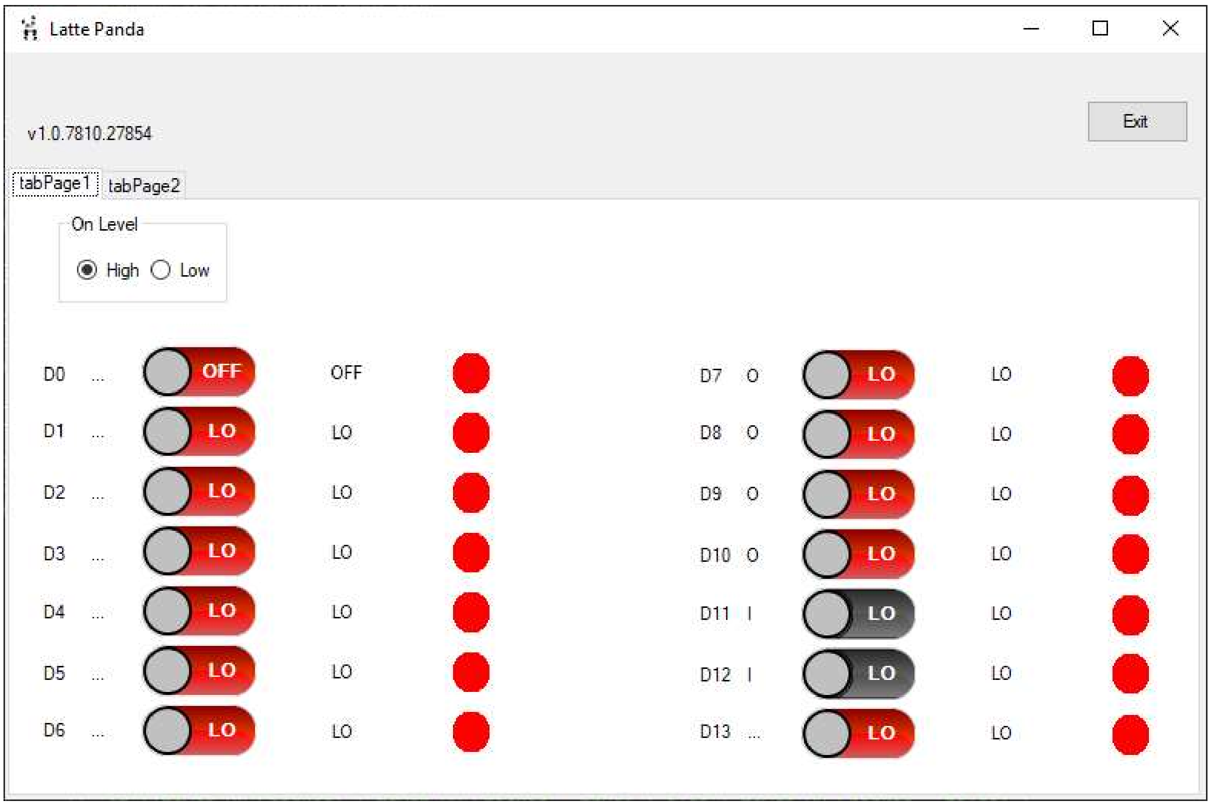
Windows app is available from: https://www.microsoft.com/en-gb/p/windows-remote-arduino-experience/9nblggh2041m?rtc=1#activetab=pivot:overviewtab

10. Win32 C# .Net App

We will create a small app that will allow us to show some of the features that we will explore.

At load time any I/O pins that have been allocated will be shown with
- "O" for output
- "I" for input
- "…" for unallocated

Inputs are Grey and are disabled.
Outputs are enabled.


Latte Panda

v1.0.7810.27854

Exit

tabPage1 | tabPage2

On Level: [✓ High] Low

D0	...	OFF	OFF	●
D1	...	LO	LO	●
D2	...	LO	LO	●
D3	...	LO	LO	●
D4	...	LO	LO	●
D5	...	LO	LO	●
D6	...	LO	LO	●
D7	O	LO	LO	●
D8	O	LO	LO	●
D9	O	LO	LO	●
D10	O	LO	LO	●
D11	I	LO	LO	●
D12	I	LO	LO	●
D13	...	LO	LO	●

11. Project 1 – Temperature and Humidity via Arduino

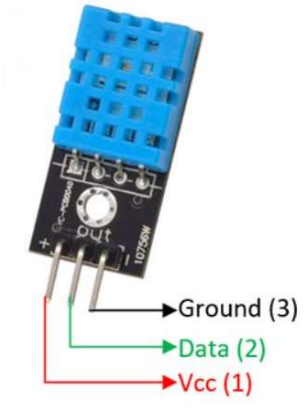
In this project we will just program the Arduino. As I have been working on a project where I have had to log various temperatures, I have a few DHT11 Temperature Sensors to hand.

11.1. DHT11 temperature & humidity sensor

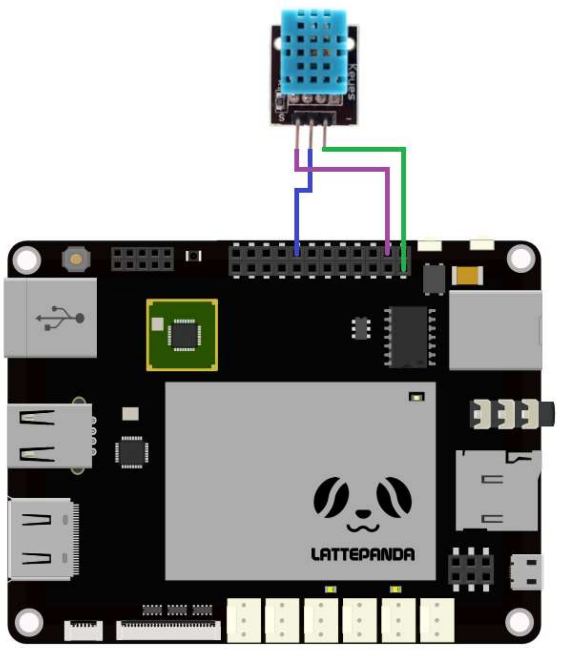
As mentioned, I am only using this model as I have many of them on my desk from a previous project's development. One thing to note is the sample rate and accuracy is as stated below but this is good enough for the first test project:

- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50°C temperature readings ±2°C accuracy
- No more than 1 Hz sampling rate (once every second)

There are many manufacturers of these sensors so please take care and note which type you have. The one I am using has three pins to connect to as shown below. I have seen 3 pin versions that have different pinouts, be careful if you are following this that you have one with the same pin outs as me.



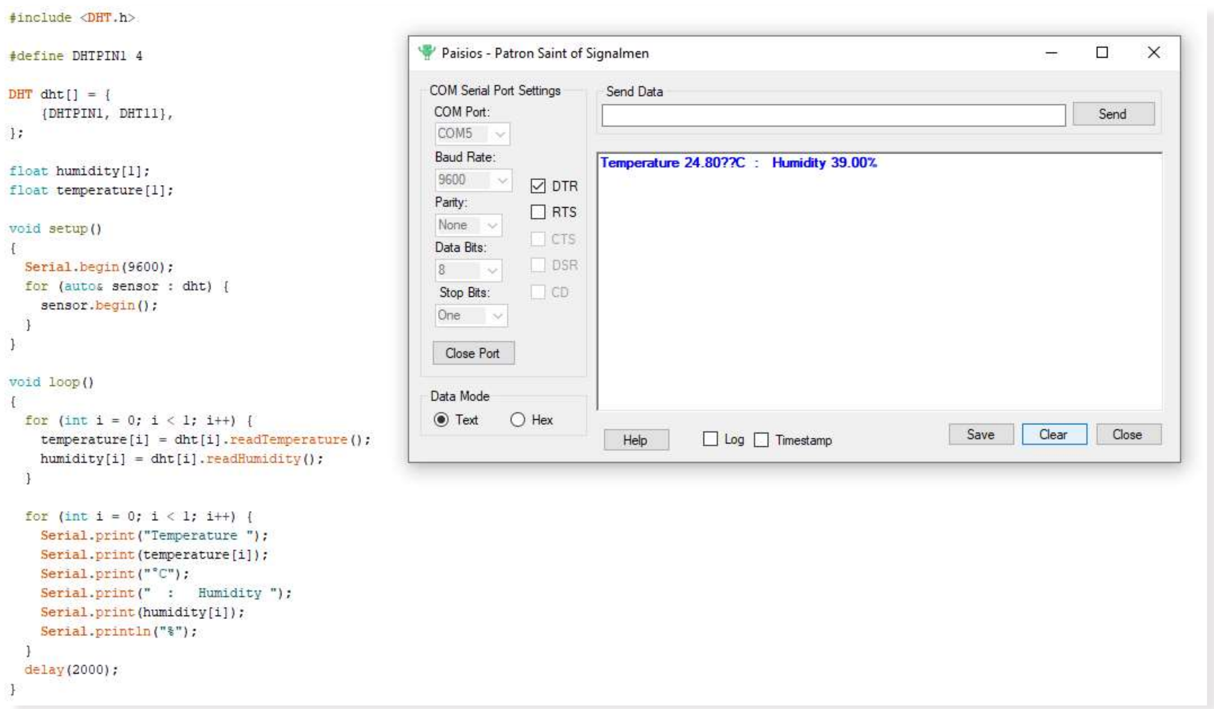
This I connect into the LattePanda as shown below:



11.2. Code

Below we see the code in the Arduino and we also see the return from the running Arduino. Note the missing ° sign. This is an issue with Paisios and nothing to do with the Arduino.

You may wonder why I have the data read back into arrays. This is because this code comes from the Gx project mentioned earlier which had several sensors on it. Here we are only using one.



```
#include <DHT.h>

#define DHTPIN1 4

DHT dht[] = {
    {DHTPIN1, DHT11},
};

float humidity[1];
float temperature[1];

void setup()
{
  Serial.begin(9600);
  for (auto& sensor : dht) {
    sensor.begin();
  }
}

void loop()
{
  for (int i = 0; i < 1; i++) {
    temperature[i] = dht[i].readTemperature();
    humidity[i] = dht[i].readHumidity();
  }

  for (int i = 0; i < 1; i++) {
    Serial.print("Temperature ");
    Serial.print(temperature[i]);
    Serial.print("°C");
    Serial.print("  :   Humidity ");
    Serial.print(humidity[i]);
    Serial.println("%");
  }
  delay(2000);
}
```

Paisios - Patron Saint of Signalmen

COM Serial Port Settings: COM Port: COM5; Baud Rate: 9600; Parity: None; Data Bits: 8; Stop Bits: One; DTR [✓]; RTS [ ]; CTS [ ]; DSR [ ]; CD [ ]

Data Mode: [✓ Text] Hex

```
Temperature 24.80??C  :  Humidity 39.00%
```

12. Project 2 – Temperature and Humidity via C# .Net

We will use the Latte Panda and connect and program everything via .Net. As we did in "Project 1" we will use the same temperature sensor in this project.

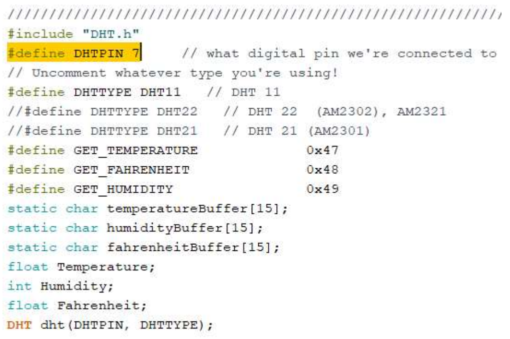
As I will be using the same sensor as I did in "Project 1" I will not repeat the data about the sensor here. Please refer to "Project 1" for that information.

The aim of this work is to create a C# app that will display the data from the sensor ever few seconds. This will just be a basic app as proof of principle. The app can be further developed for your individual needs if required.

12.1. Arduino Code

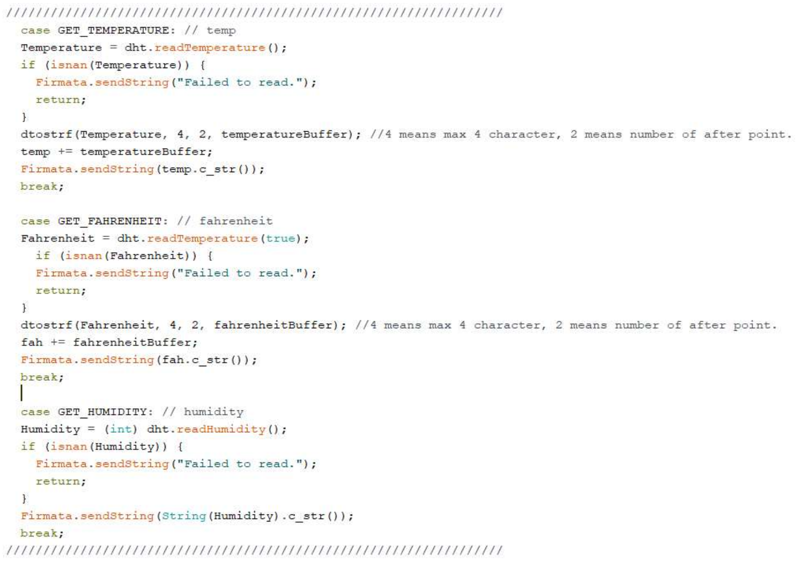
The first step here is to add code to the Arduino so that it will be able to read the sensor and return the data to the C# app when it requests it. As we will be using the Firmata protocol for the data transmission we could use the latest Standard Firmata library and adjust it to our needs if required but I found an older copy that had already been worked on for the sensor that I am using and as this is a proof of principle, I will use it. I am not sure who the person was that changed it but I give them full credit for doing it.

When I look in the Arduino code "DHT_Firmata" I see that the pin that the sensor will use is D7. You can change this if you want but I will go with this pin and the type of sensor that I will use is the DHT11 as shown below.



```
/////////////////////////////////////////////////////////////
#include "DHT.h"
#define DHTPIN 7     // what digital pin we're connected to
// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
#define GET_TEMPERATURE             0x47
#define GET_FAHRENHEIT              0x48
#define GET_HUMIDITY                0x49
static char temperatureBuffer[15];
static char humidityBuffer[15];
static char fahrenheitBuffer[15];
float Temperature;
int Humidity;
float Fahrenheit;
DHT dht(DHTPIN, DHTTYPE);
```

The other part of the file to note is where it gets the data from the sensor.

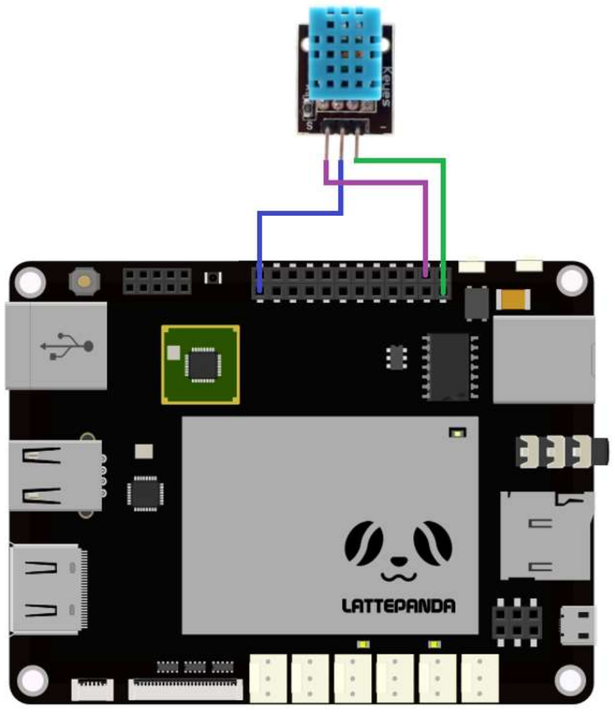


```
//////////////////////////////////////////////////////////////////
  case GET_TEMPERATURE: // temp
  Temperature = dht.readTemperature();
  if (isnan(Temperature)) {
    Firmata.sendString("Failed to read.");
    return;
  }
  dtostrf(Temperature, 4, 2, temperatureBuffer); //4 means max 4 character, 2 means number of after point.
  temp += temperatureBuffer;
  Firmata.sendString(temp.c_str());
  break;

  case GET_FAHRENHEIT: // fahrenheit
  Fahrenheit = dht.readTemperature(true);
    if (isnan(Fahrenheit)) {
    Firmata.sendString("Failed to read.");
    return;
  }
  dtostrf(Fahrenheit, 4, 2, fahrenheitBuffer); //4 means max 4 character, 2 means number of after point.
  fah += fahrenheitBuffer;
  Firmata.sendString(fah.c_str());
  break;

  case GET_HUMIDITY: // humidity
  Humidity = (int) dht.readHumidity();
  if (isnan(Humidity)) {
    Firmata.sendString("Failed to read.");
    return;
  }
  Firmata.sendString(String(Humidity).c_str());
  break;
//////////////////////////////////////////////////////////////////
```

You could merge these changes into the latest version of the StandardFirmata library and it should work with that as well.

The next thing to do is to load the DHT_Firmata into the Arduino. Once complete the Arduino is fully programmed.

As mentioned for more details especially the pinouts of the sensor please refer to "Project 1". The sensor is wired up as follows:



12.2. C# .Net App explanation

The next step is to create the C# app to allow us to request and capture the data from the sensor.

To start we go to VS IDE (I am using 2019 Community addition) and create a new C# .Net WinForm Solution. Once you have got the solution created, you need to add the Arduino class from GitHub: `https://github.com/LattePandaTeam/LattePanda-Development-Support/tree/master/LattePandaFirmata`

This is done by adding the file called Arduino.cs to your project.

In the solution we create a reference to the class

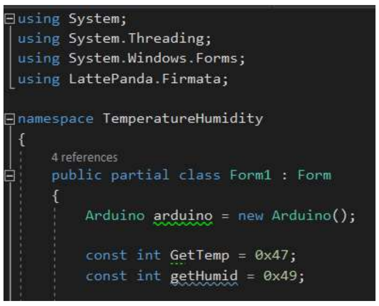
using LattePanda.Firmata;

and instantiate the class at the start of the C# code.

Arduino arduino = new Arduino();

Now you just use the instance of `arduino` to get the data we need from the sensor. We also need to add the addresses so Firmata knows how to get the data for the Temperature and Humidity from the sensor.

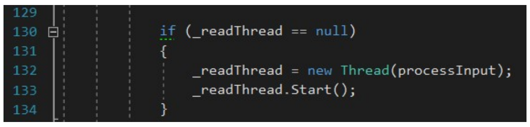
const int GetTemp = 0x47;
const int getHumid = 0x49;



```
using System;
using System.Threading;
using System.Windows.Forms;
using LattePanda.Firmata;

namespace TemperatureHumidity
{
    4 references
    public partial class Form1 : Form
    {
        Arduino arduino = new Arduino();

        const int GetTemp = 0x47;
        const int getHumid = 0x49;
```

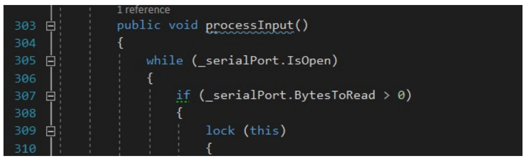
At this stage it is good to take a look at the Arduino.cs and understand what it is doing. The reason here is that we are making a WinForm app and this is event driven and if you have worked with WinForms there are conditions under which it will not be easy to refresh the UI.

In Arduino.cs we see that there is a main thread



```
                if (_readThread == null)
                {
                    _readThread = new Thread(processInput);
                    _readThread.Start();
                }
```

that gets started as function processInput().



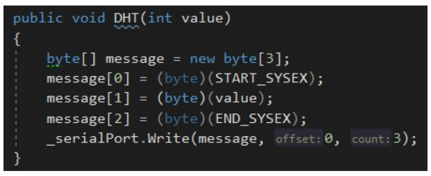
```
        1 reference
        public void processInput()
        {
            while (_serialPort.IsOpen)
            {
                if (_serialPort.BytesToRead > 0)
                {
                    lock (this)
                    {
```

We need to be careful here as this is the function is the one that gets the Temperature and Humidity. The first thing to watch is cross threading but the more important one is that this thread may be so busy that it prevents the UI from updating. Now we know this we will need to code the App to make sure that the UI will be able to update and also not cross thread. If this was more than just proof of principle, I may want to change the way the data is collected but for now I use what I have.

In my C# app I will first get send a call to the Arduino to get the Temparture data

arduino.DHT(GetTemp);

This will call the following function and pass in the address of the Temperature or Humidity.



```
public void DHT(int value)
{
    byte[] message = new byte[3];
    message[0] = (byte)(START_SYSEX);
    message[1] = (byte)(value);
    message[2] = (byte)(END_SYSEX);
    _serialPort.Write(message, offset: 0, count: 3);
}
```

This function will build the Firmata packet and send it to the Arduino where it will be resolved. For the temperature the Firmata packet will be (all data shown as Hex in base 16):

F0 47 F7

The return from this will come back in on the RS232 port and it will be captured in the Function

```
public void processInput()
```

where it will get written to a string

```
STRING_DHT = new string(CSEN);
```

I will now make another call

```
arduino.STRING_DHT
```

to get the data from this string so I can display it

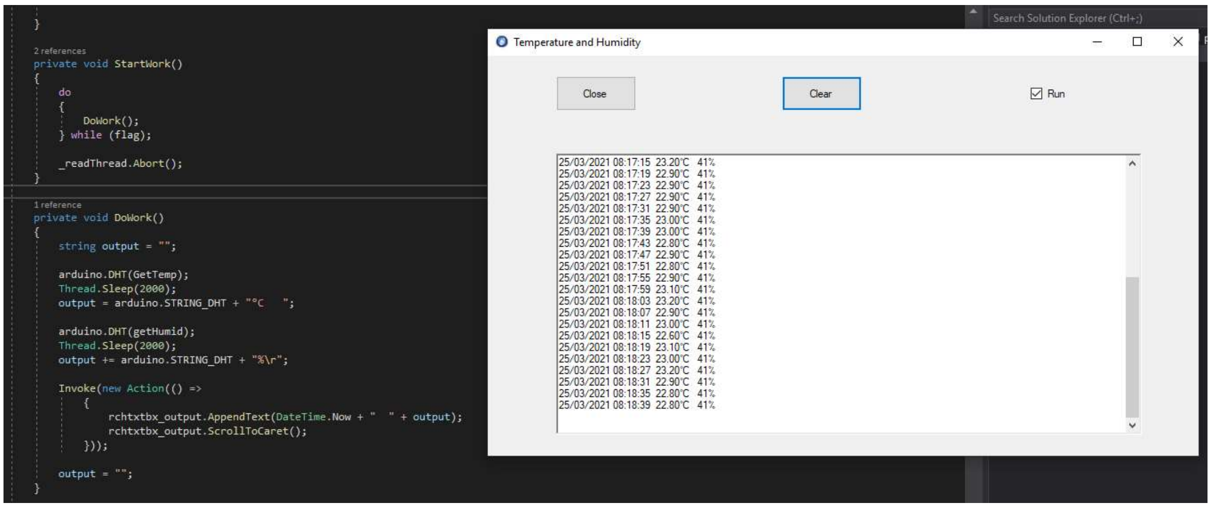
```
output = arduino.STRING_DHT + "°C   ";
```

Note that we are working synchronously here, and the data returned is the latest that is in the String. You may want to change this section if you want to run asynchronously.

As I am talking to the Arduino.cs and it has threads I need to make sure that when I write the data to my C# app UIs richtextbox that I do not cross thread. To do this I use Invoke to get it into the correct thread if it is already not there.

```
Invoke(new Action(() =>
            {
              rchtxtbx_output.AppendText(DateTime.Now + "  " + output);
              rchtxtbx_output.ScrollToCaret();
            }));
```

As we can see below the app is running in real time and every 4 seconds it will log the data into the UI for us to view.



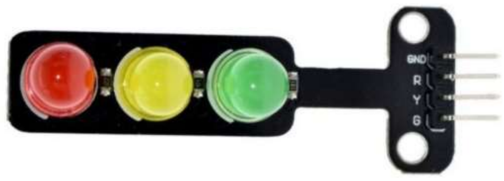
```
}

2 references
private void StartWork()
{
    do
    {
        DoWork();
    } while (flag);

    _readThread.Abort();
}

1 reference
private void DoWork()
{
    string output = "";

    arduino.DHT(GetTemp);
    Thread.Sleep(2000);
    output = arduino.STRING_DHT + "°C   ";

    arduino.DHT(getHumid);
    Thread.Sleep(2000);
    output += arduino.STRING_DHT + "%\r";

    Invoke(new Action(() =>
        {
            rchtxtbx_output.AppendText(DateTime.Now + "  " + output);
            rchtxtbx_output.ScrollToCaret();
        }));

    output = "";
}
```

```
25/03/2021 08:17:15  23.20°C   41%
25/03/2021 08:17:19  22.90°C   41%
25/03/2021 08:17:23  22.90°C   41%
25/03/2021 08:17:27  22.90°C   41%
25/03/2021 08:17:31  22.90°C   41%
25/03/2021 08:17:35  23.00°C   41%
25/03/2021 08:17:39  23.00°C   41%
25/03/2021 08:17:43  22.80°C   41%
25/03/2021 08:17:47  22.90°C   41%
25/03/2021 08:17:51  22.80°C   41%
25/03/2021 08:17:55  22.90°C   41%
25/03/2021 08:17:59  23.10°C   41%
25/03/2021 08:18:03  23.20°C   41%
25/03/2021 08:18:07  22.90°C   41%
25/03/2021 08:18:11  23.00°C   41%
25/03/2021 08:18:15  22.60°C   41%
25/03/2021 08:18:19  23.10°C   41%
25/03/2021 08:18:23  23.00°C   41%
25/03/2021 08:18:27  23.20°C   41%
25/03/2021 08:18:31  22.90°C   41%
25/03/2021 08:18:35  22.80°C   41%
25/03/2021 08:18:39  22.80°C   41%
```

There may be better ways to achieve this with better code but as a proof of principle, I think this shows that we can interact with the Arduino I/O direct from the C# app.

13. Project 3 – Controlling I/O to set signals

This project will see us switching on and off some I/O pins on the Arduino from a C# .Net app. To show that it is working a LED is attached to 3 pins D7, D8 and D9. This just happens to be a traffic light module I have to hand. This is ideal as the module has the resistors and the LEDs all built in so we only need to connect to its pins. If you are following the diagrams on this page please make sure your module has the same pinouts as that shown below. Some are different so just adjust accordingly.

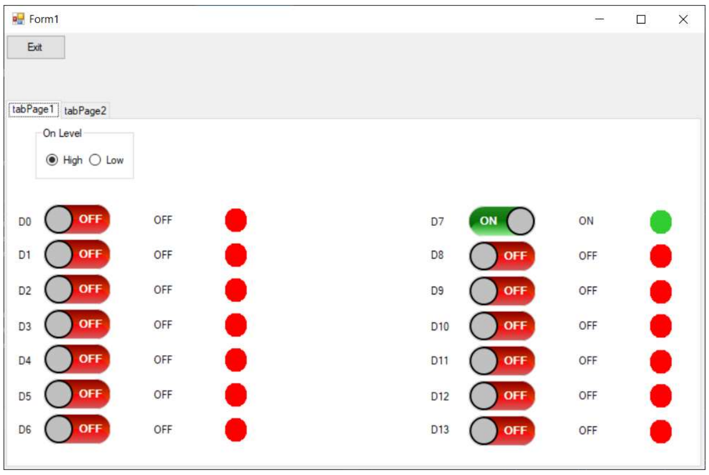


13.1. Windows App

For this we will be using the I/O as PNP and NPN. That means at times the lines will be High when Off and Low when On and at other times it will be Low when On and High when Off. This is all controlled from the Windows App

The app as shown below is just an Alpha build to get something working. It will be evolved in time and be prettified. For the initial test we will just use D7 and this will operate the RED LED.

The switches that you see are just checkboxes that have been enhanced to look like Switches. They are termed “Checked” when on and “Not Checked” when Off just as standard checkboxes.



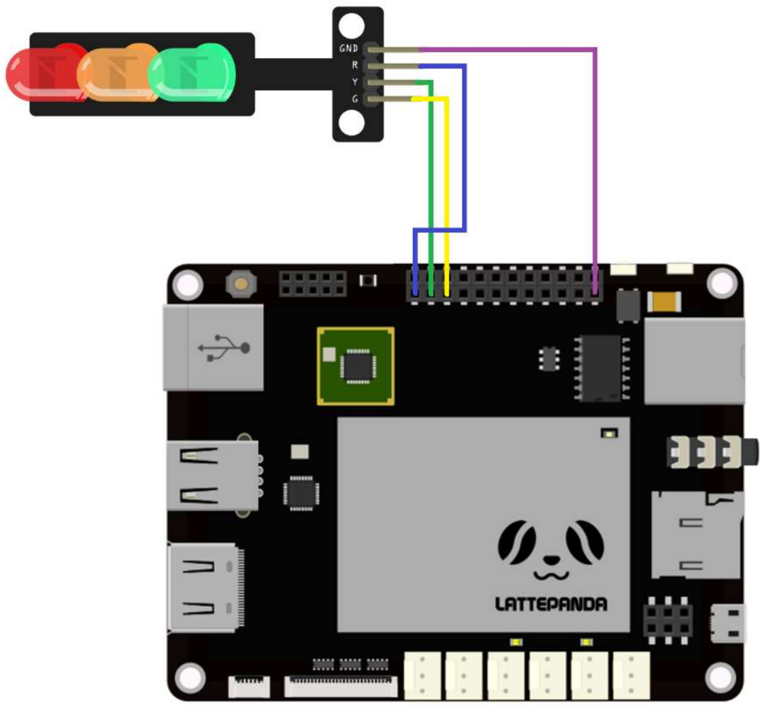
On Level: [✓ High] Low

D0	OFF	OFF	●	D7	ON	ON	●
D1	OFF	OFF	●	D8	OFF	OFF	●
D2	OFF	OFF	●	D9	OFF	OFF	●
D3	OFF	OFF	●	D10	OFF	OFF	●
D4	OFF	OFF	●	D11	OFF	OFF	●
D5	OFF	OFF	●	D12	OFF	OFF	●
D6	OFF	OFF	●	D13	OFF	OFF	●

If you swap the On Level from High to Low then the Switch D7 will automatically toggle to maintain the same signal level. As shown above the Red LED will be on but change to Low level and the switch will toggle off but the LED will remain on. This shows the switching to High or switching to Low for PNP and NPN signals.

13.2. Hardware Circuit

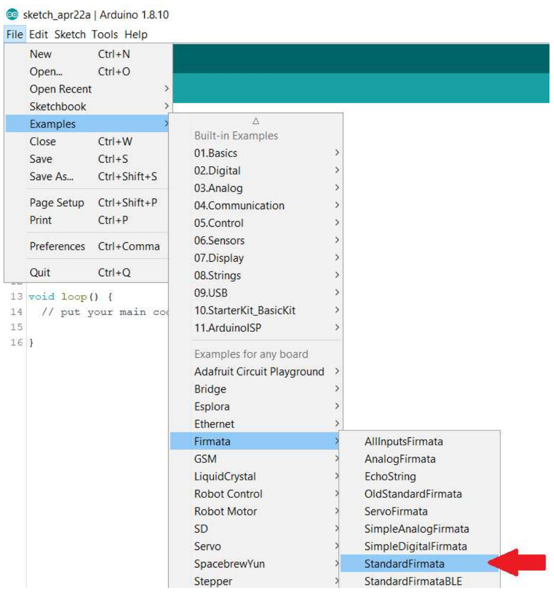
Using the Traffic Light Module makes the connections simple as we can see below. This module contains the full circuit including the resistors so works out the box. The circuit diagram is show below.



As an aside in some countries Traffic Lights are called Robots see https://en.wikipedia.org/wiki/Traffic_light.

13.3. Arduino code

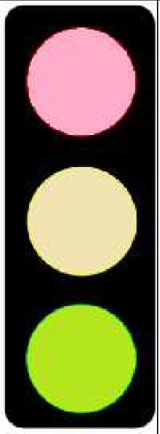
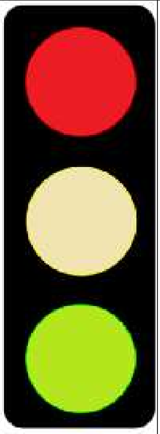
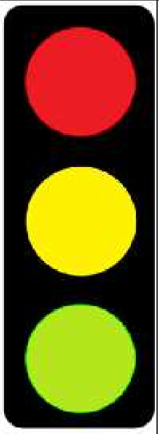
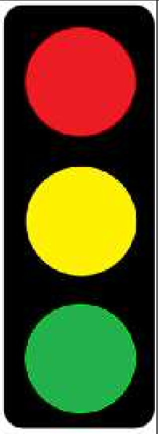
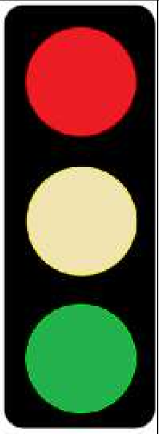
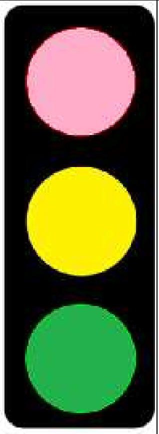
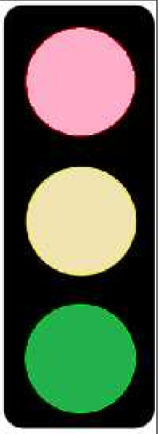
This is simple as we will use the standard Firmata example which is included in the Arduino IDE. Go to examples and open up the file and send it down to the Panda Latte Arduino. That is all that needs to happen to the Arduino to allow our windows app to directly address the I/O pins.



```
void loop() {
  // put your main co
}
```

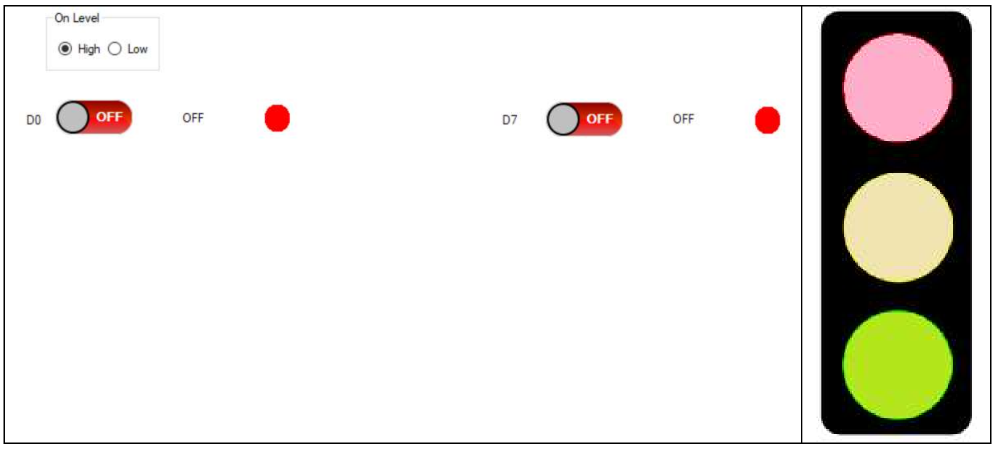
13.4. Running the Windows App.

Traffic light states are as follows

						
Red = off	Red = on	Red = on	Red = on	Red = on	Red = off	Red = off
Amber = off	Amber = off	Amber = on	Amber = on	Amber = off	Amber = on	Amber = off
Green = off	Green = off	Green = off	Green = on	Green = on	Green = on	Green = on

Remember that OFF = LOW and ON = HIGH

When we start the app the signal is set HIGH and D7 is LOW meaning RED LED is off



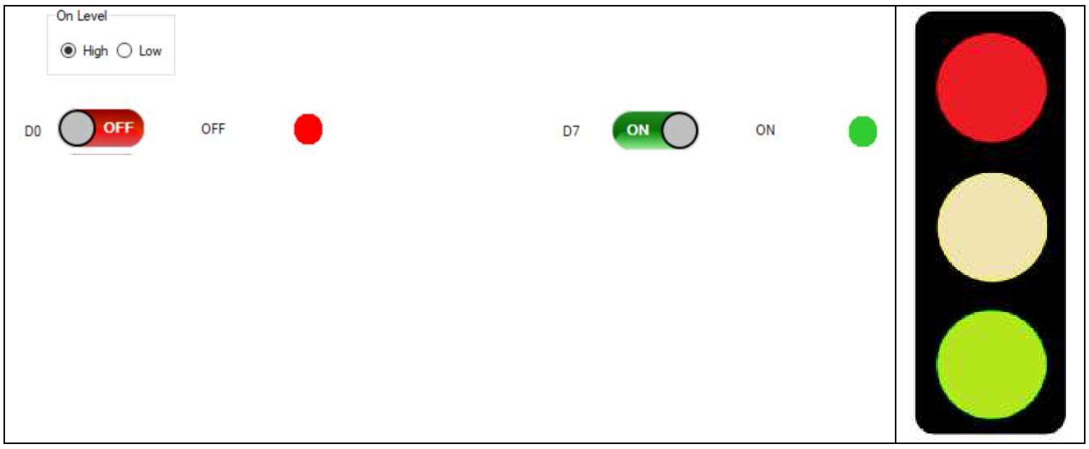
On Level

[✓ High] Low

D0 OFF OFF

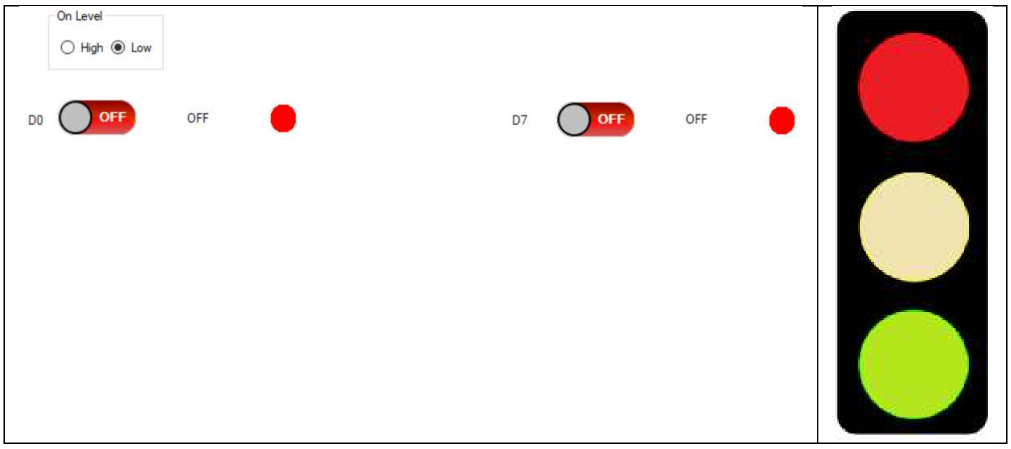
D7 OFF OFF

If we put D7 to HIGH then the RED LED will come on.



Turn the "On Level" to LOW and you will see that as the RED LED is on that D7 will go to the LOW state to maintain the RED LED being on.

Note that D7 is showing its icon as OFF (Red) when it should show as ON (Green) as mentioned in conclusion this needs changing in the future.



13.5. Conclusions

As already mentioned, the Windows app is very much Alpha build just to make things work and we can now see various items that we can change in it to make things work better/correctly. These items will be worked on in the future.

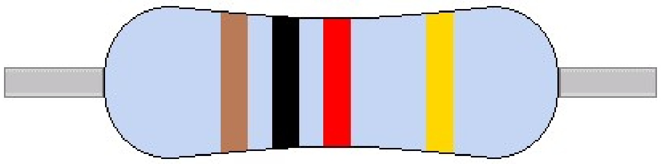
I think that instead of having the Windows App UI say ON and OFF that it should say HIGH and LOW. This makes more sense as I set the signal high or low. The icon in the UI to show if the signal is on or off also needs to be changed to show it real value.

14. Project 4 – Get and Set Digital I/O signals Via Windows app

This project will see us pressing a switch to turn on an I/O pin on the Latte Panda Arduino. The switch will be a press button switch and it will take its power from the Arduino. The change in the input voltage on the incoming I/O pin will be detected by a C# .Net app running on the Latte Panda. The App will indicate when the signal is active by showing a Green circle and when it is inactive a Red circle. Active means that it detects when it goes from 0v to 5v and inactive from 5v to 0v.

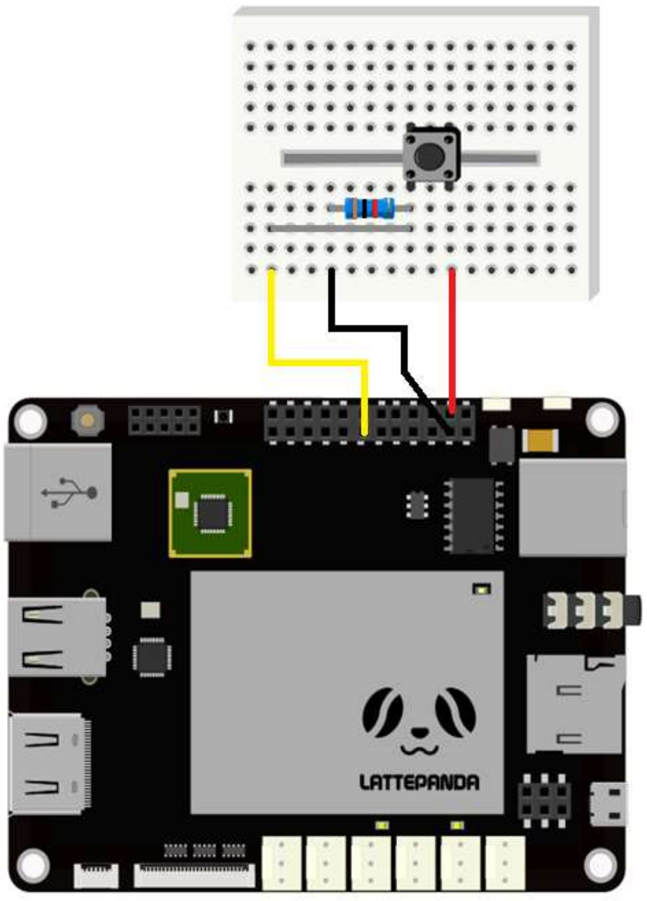
As we are bringing a voltage into the Arduino, we need to take precautions to make sure we do not bring in a signal that is likely to cause permanent damage to the Arduino. As we are not sure whether internal pullups are enabled, we will add a low-tech option of a min 1kΩ resistor. This will limit current too well within the capabilities of the drivers, even if the level deviated to say 12V due to a fault elsewhere.

A 1kΩ resistor has the following colour bands Brown : Black : Red : Gold.



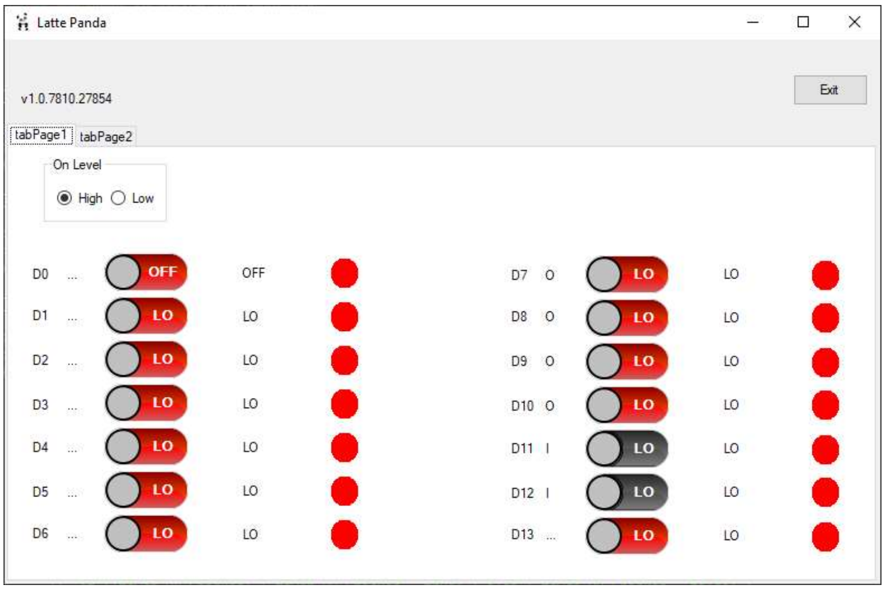
14.1. Circuit Diagram using Switch

The circuit diagram showing the connections that we will use is shown next:



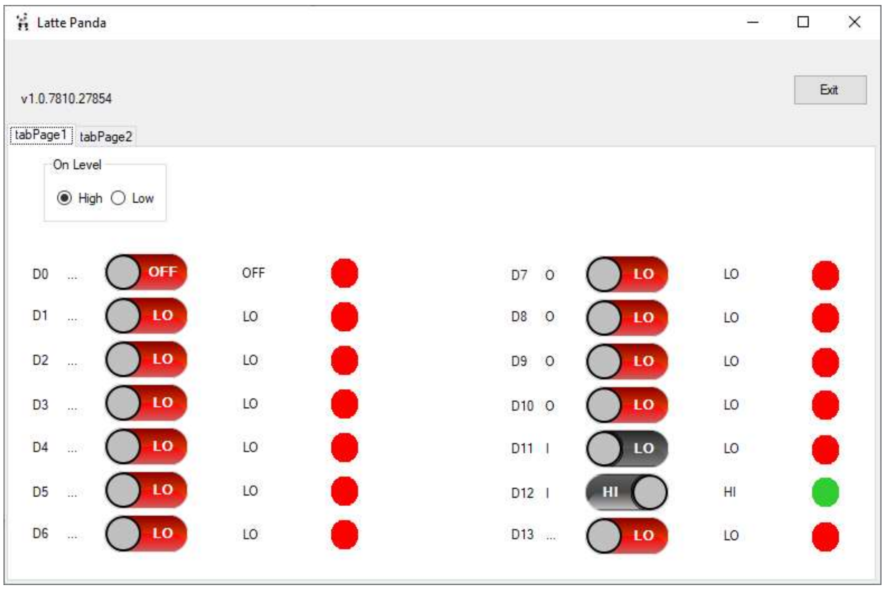
When we press the switch, the current will flow from the Red Wire (Vcc) to the Yellow wire (Load). As explained earlier we will add protection of a 1kΩ resistor from the load to Black Wire (Ground).

We modify the App so that we know which are inputs and which are outputs. If the switch is on an input, then we will disable it so that it only gets and cannot set as show in Grey below.



```
On Level
(●) High  ( ) Low

D0  ...  OFF  OFF      D7   O  LO  LO
D1  ...  LO   LO       D8   O  LO  LO
D2  ...  LO   LO       D9   O  LO  LO
D3  ...  LO   LO       D10  O  LO  LO
D4  ...  LO   LO       D11  I  LO  LO
D5  ...  LO   LO       D12  I  LO  LO
D6  ...  LO   LO       D13 ... LO  LO
```

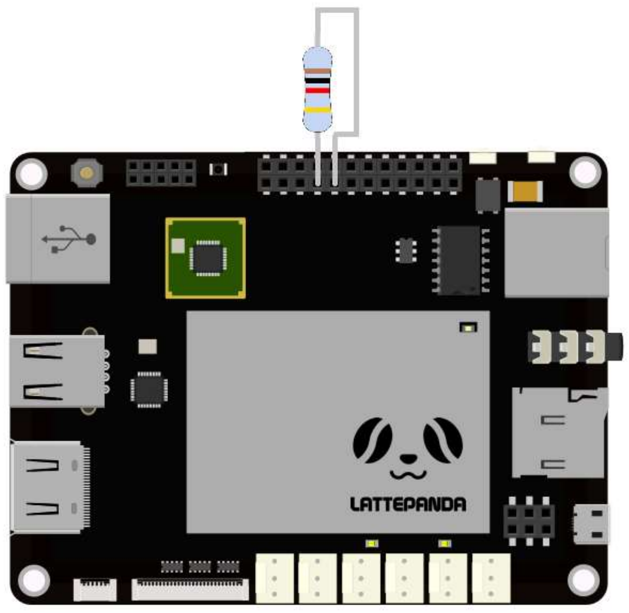
We have attached the Yellow wire to D12 so when we press the switch, we expect that D12 will change. We test this and as you see in the next picture this indeed what happens.



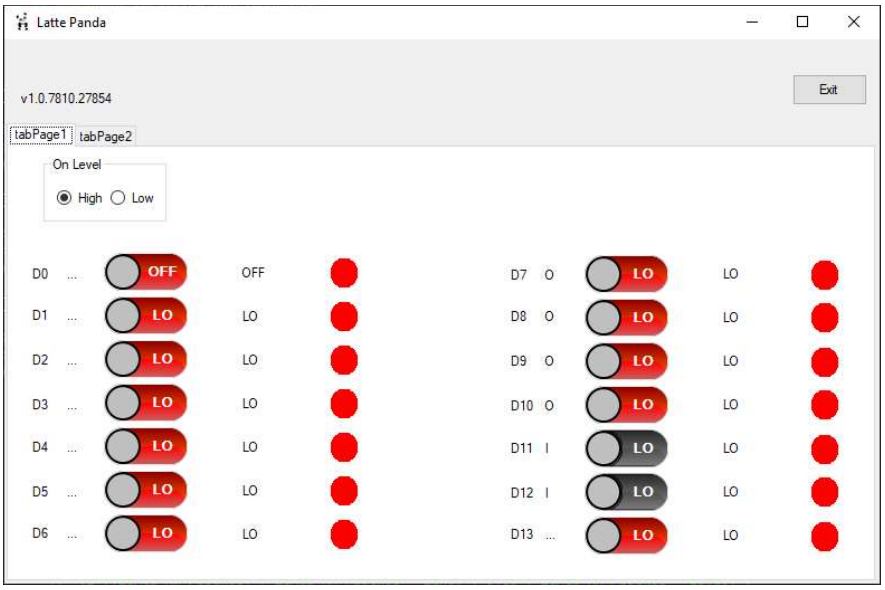
```
On Level
(●) High  ( ) Low

D0  ...  OFF  OFF      D7   O  LO  LO
D1  ...  LO   LO       D8   O  LO  LO
D2  ...  LO   LO       D9   O  LO  LO
D3  ...  LO   LO       D10  O  LO  LO
D4  ...  LO   LO       D11  I  LO  LO
D5  ...  LO   LO       D12  I  HI  HI
D6  ...  LO   LO       D13 ... LO  LO
```

14.2. Circuit Diagram using I/O

In this example we will send the signal out of one I/O pin and read it on another. We do this with all the safety precautions of a 1kΩ resistor as explained earlier.



We wire this so that a signal sent out of D10 will be read back through D11.



Latte Panda

v1.0.7810.27854

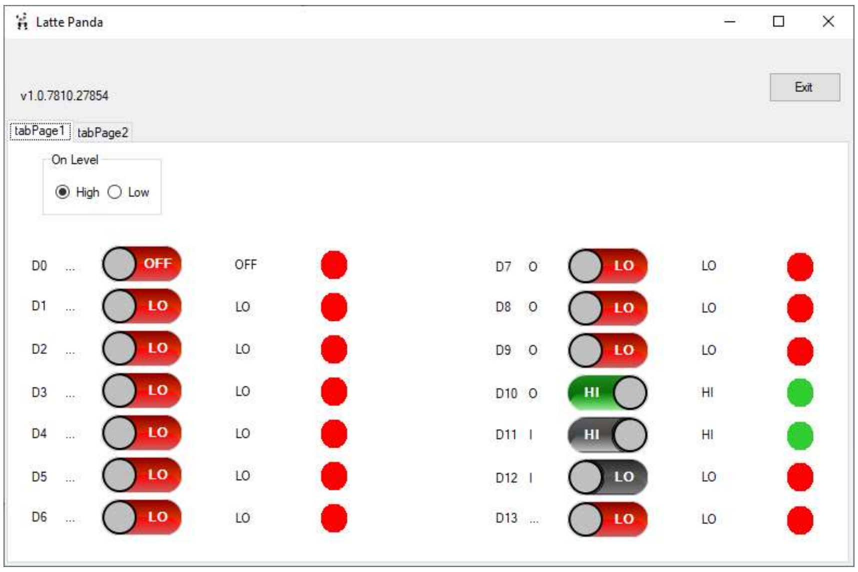
Exit

tabPage1 tabPage2

On Level: [✓ High] Low

```
D0  ...  OFF  OFF  ●        D7   O  LO  LO  ●
D1  ...  LO   LO   ●        D8   O  LO  LO  ●
D2  ...  LO   LO   ●        D9   O  LO  LO  ●
D3  ...  LO   LO   ●        D10  O  LO  LO  ●
D4  ...  LO   LO   ●        D11  I  LO  LO  ●
D5  ...  LO   LO   ●        D12  I  LO  LO  ●
D6  ...  LO   LO   ●        D13 ... LO  LO  ●
```

We run this test and when we switch D10 we then see D11 receives the signal, and it changes as shown below.



Latte Panda

v1.0.7810.27854

Exit

tabPage1 | tabPage2

On Level: [✓ High] Low

D0	...	OFF	OFF	🔴	D7	O	LO	LO	🔴
D1	...	LO	LO	🔴	D8	O	LO	LO	🔴
D2	...	LO	LO	🔴	D9	O	LO	LO	🔴
D3	...	LO	LO	🔴	D10	O	HI	HI	🟢
D4	...	LO	LO	🔴	D11	I	HI	HI	🟢
D5	...	LO	LO	🔴	D12	I	LO	LO	🔴
D6	...	LO	LO	🔴	D13	...	LO	LO	🔴

14.3. Conclusions

From the work above we can see that we can get or set a signal level from the Latte Panda app.