



SensorDuino

ARDUINO UNO & ESP8266 (NODEMCU)

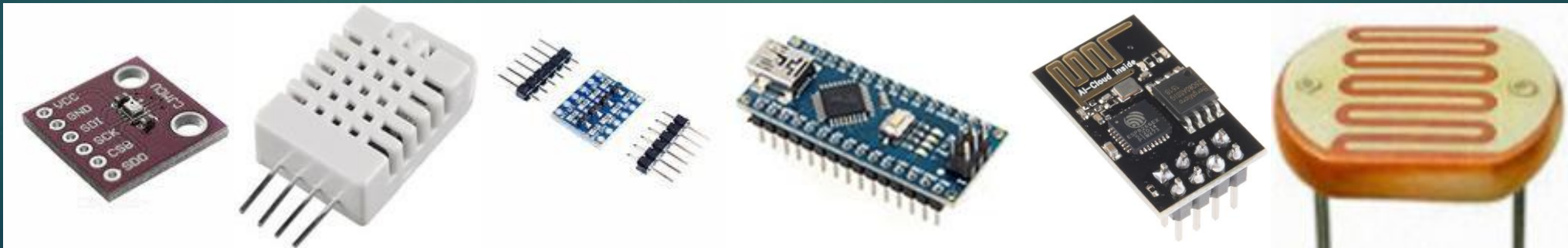
BY: GERARD VAN SEVENTER - 2017

Functionality & Requirements

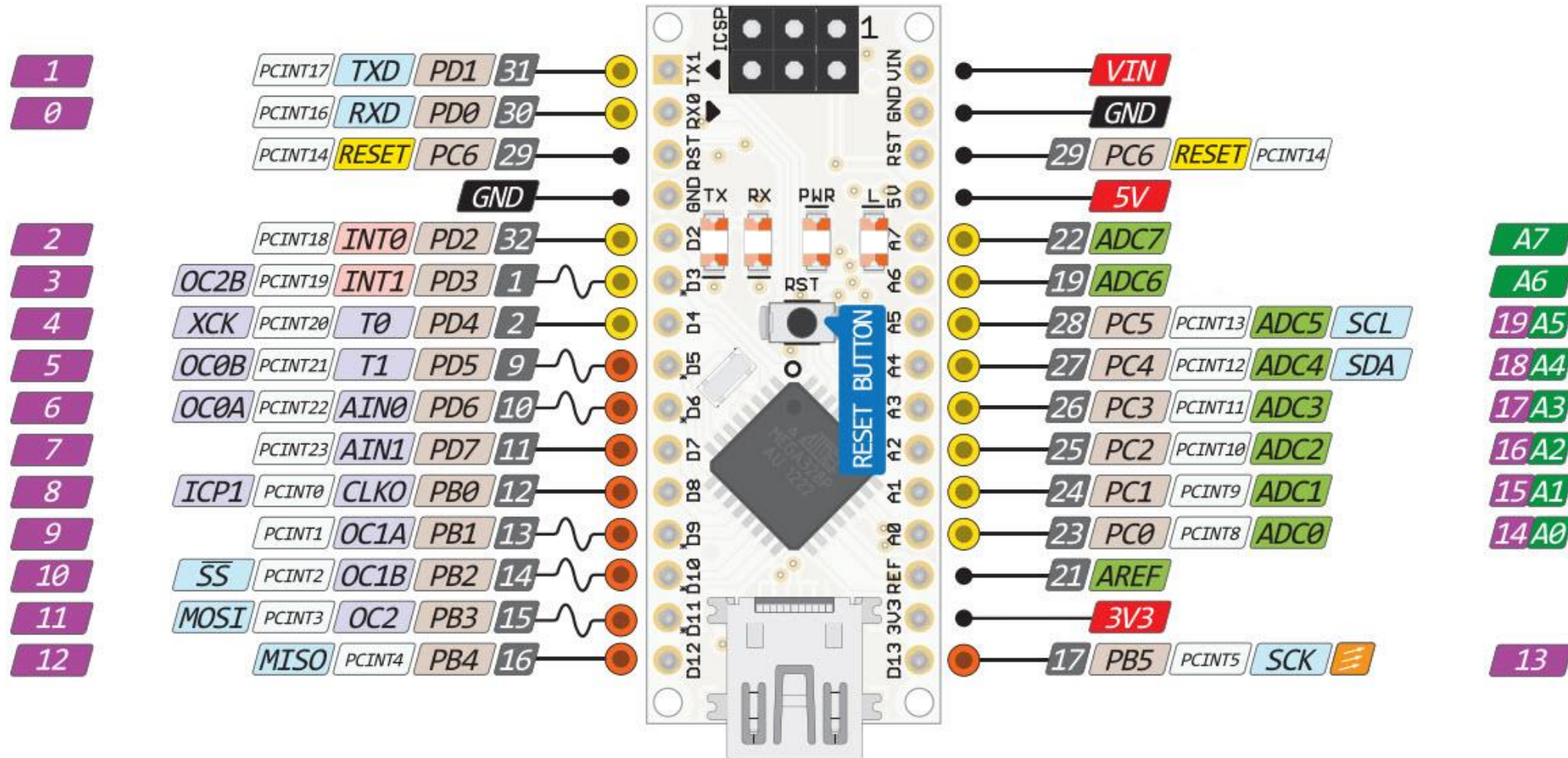
- ▶ Low cost
- ▶ monitor water level in two tanks via magnetic switches (3 levels)
- ▶ measures:
 - ▶ temperature
 - ▶ barometric pressure
 - ▶ air humidity
 - ▶ Light
- ▶ optional:
 - ▶ magnetic switches (door/alarm/etc.)
 - ▶ relay output (drives waterpump when level is too high)
- ▶ reports directly to Domoticz by calling JSON uri

Components

- ▶ Arduino Nano
- ▶ ESP-01 (esp8266) as Wifi shield
- ▶ ESP is running NodeMCU
- ▶ Logic level converter 3.3v <-> 5v
- ▶ 2 x 3 Switches (Water level in 2 tanks)
- ▶ BMP280 barometric pressure and temperature sensor
- ▶ DHT22 moisture and temperature sensor
- ▶ LDR Light dependent resistor



Arduino NANO Pinout



ESP-01

ESP8266 MODULE ESP-01 CHEAT SHEET

ESP8266 FEATURES:

- 32 BIT CPU @ 80MHZ
- 64 KB COMMAND RAM
- 96 KB DATA RAM
- EXTERNAL QPI FLASH
USUALLY 512KB
UP TO 4MBIT
- IEEE 802.11 B/G/N WIFI
2.4GHz, WEP/WPA/WPA2
- UP TO 16 GPIO PINS
- SPI, I²C, I²S, UART
- 10-BIT ADC



PINOUT



ESP01 FEATURES:

- 2x4 DIL HEADER
- INTEGRATED ANTENNA
- INTEGRATED LED (V_{CC}, TXD)
- USUALLY 512KB OR
1MB FLASH

OPERATING

V_{CC}: 3.3V (UP TO 200MA)

IO AND UART ARE NOT 5V TOLERANT

CH_PD (AKA ENABLE) MUST BE PULLED HIGH
TO OPERATE

MODULE VARIANTS

ESP01V1:

ONLY V_{CC}, GND, RXD AND TXD CONNECTED
CAN NOT BE FLASHED WITHOUT MODIFICATION

ESP01V2:

CONNECTIONS AS SHOWN HERE

BOOT MODES

PINS MUST BE PULLED TO THE
APPROPRIATE LEVEL DURING POWERUP

GPI015	GPI00	GPI02	Mode
LOW	LOW	HIGH	Serial Programming
LOW	HIGH	HIGH	Boot from Flash
HIGH	ANY	ANY	Boot from SD-Card

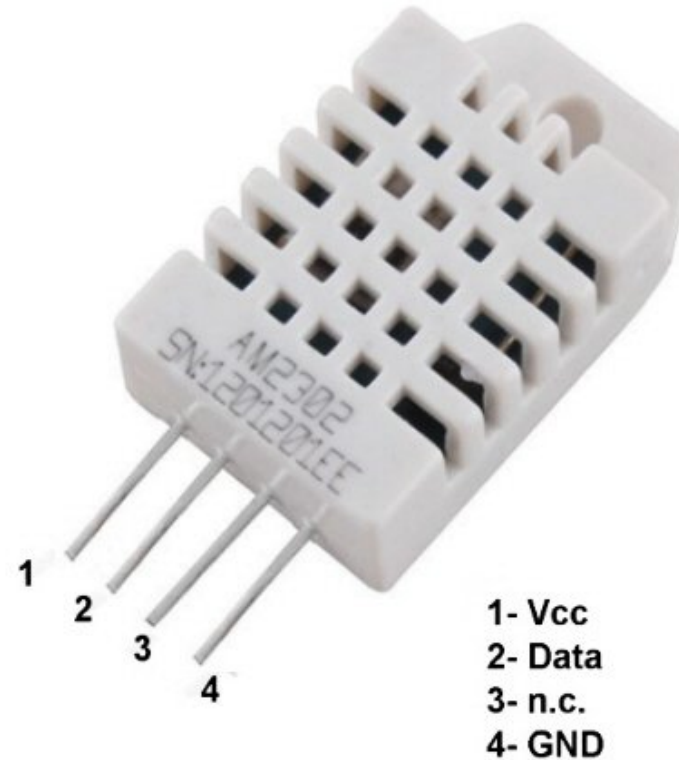
GPI015 is connected to GND on ESP-01

DHT22

DHT22 Temperature-Humidity Sensor

- 3.3 to 6V power and I/O
- 1.5mA max current use during conversion
- 0-100% humidity readings with 2-5% accuracy
- -40 to 80°C temperature readings $\pm 0.5^{\circ}\text{C}$ accuracy
- Up to 0.5 Hz sampling rate (once every 2 seconds)
- 4 pins, 0.1" spacing

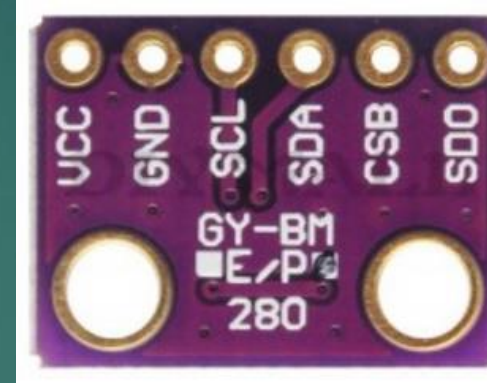
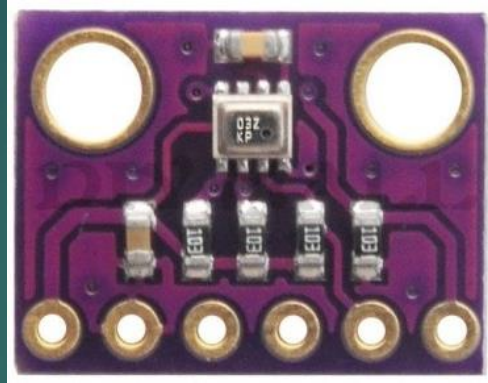
- 1) VCC
- 2) DATA (digital I/O)
- 3) Not Connected (N.C)
- 4) GND



Note: Connect a 4.7K or 10K resistor between VCC and the DATA pin

Barometric pressure & temperature

► BMP280 via I2C

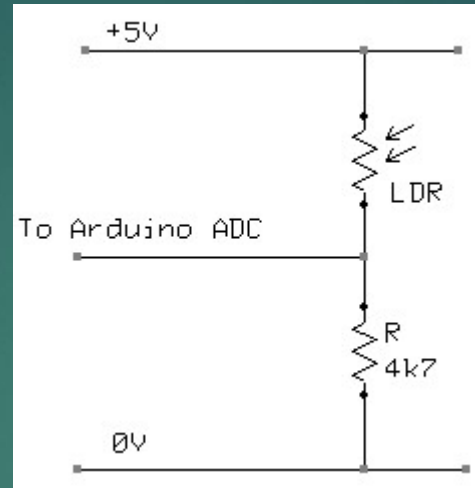


Newest high precision Bosch BMP280 barometric pressure & temperature sensor. Offers pressure measuring range of 300 to 1100 hPa with an accuracy of 0.01 hPa (higher accuracy than older BMP180 at 0.02 hPa resolution). Excellent for barometric pressure, elevation & vertical velocity speed measurements. It's based on piezo-resistive technology for high accuracy, ruggedness and long term stability. Comes with breadboard compatible 6-PIN header.

- Includes barometric & temperature sensor
- Connects to a microcontroller or Arduino via I2C bus (SCL->A5, SDA->A4) or SPI (SCL->SCK, CSB->MISO, SDA->MOSI, CSB->CS)
- Two wire interface, SCL & SDA are 5V tolerant
- Factory calibrated with the calibration coefficients stored in ROM
- Voltage supply (VIN): 1.8V to 3.6V (3.3V typical)
- Ultra low power consumption

Other Sensors

► LDR

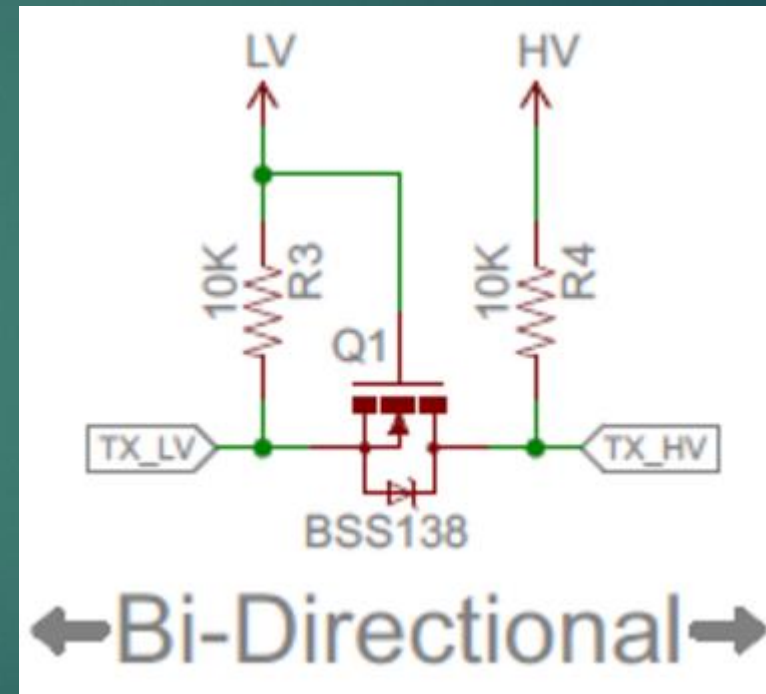
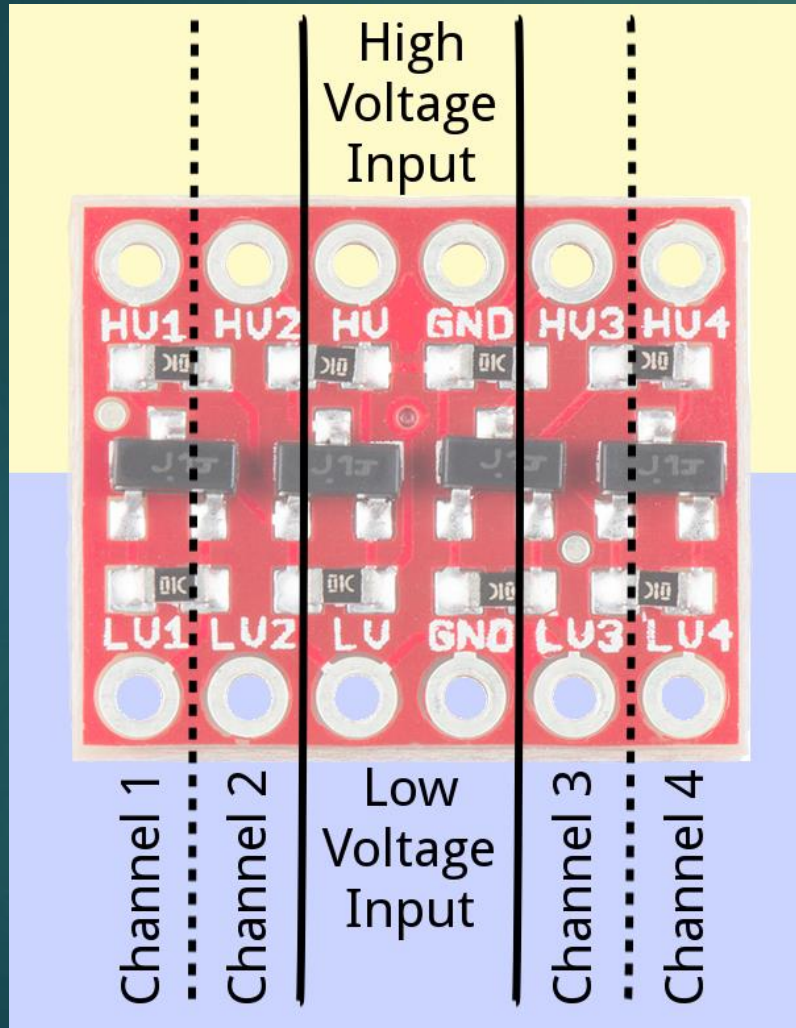


► Magnetic Switches

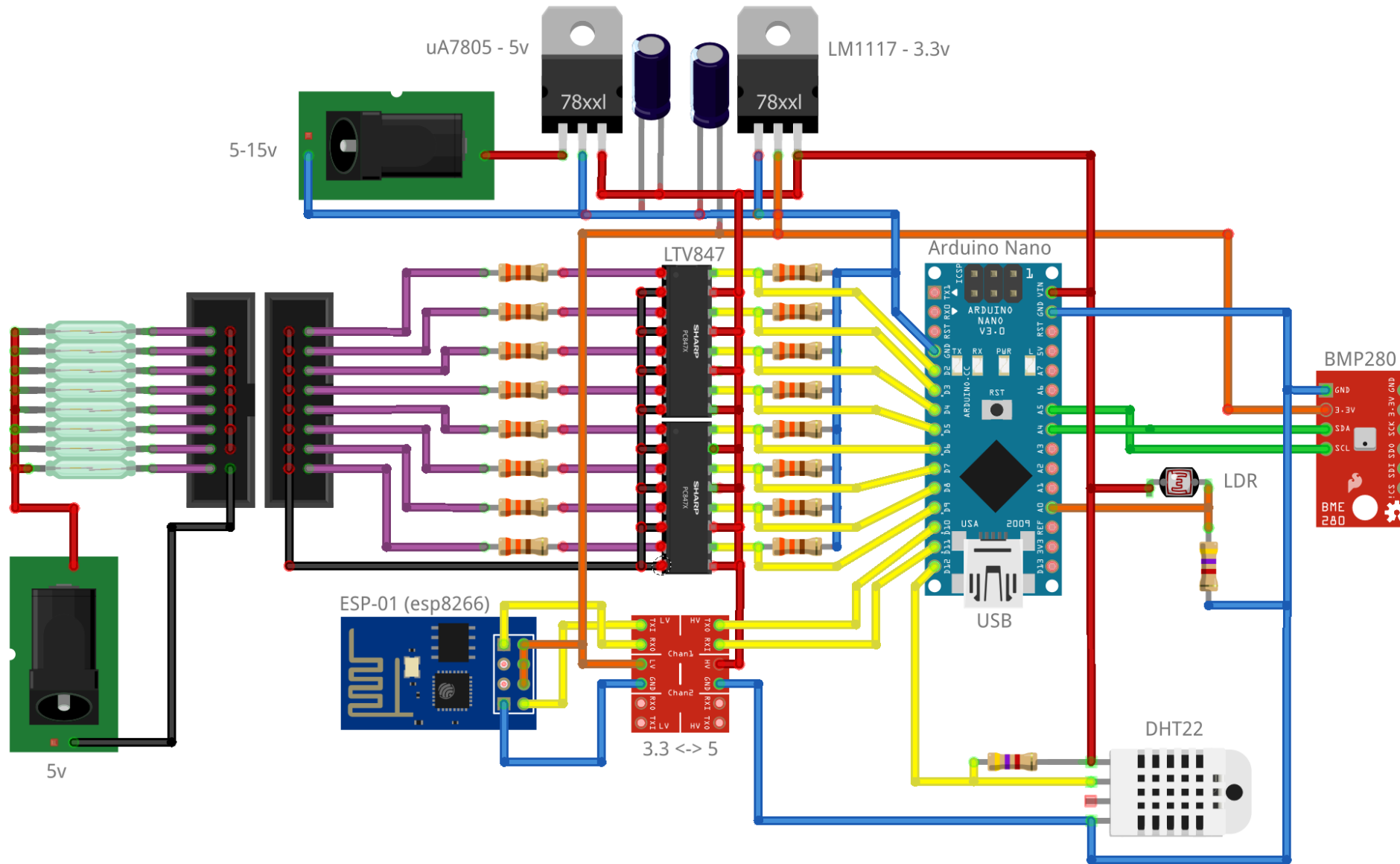


Opto isolated

Logic Level Converter 3.3V \leftrightarrow 5V



Schematic



Software

- ▶ Arduino IDE for programming Arduino NANO

```
if (Serial.available()) {  
    digitalWrite(13, HIGH);  
    ESP8266.write(Serial.read());  
}
```

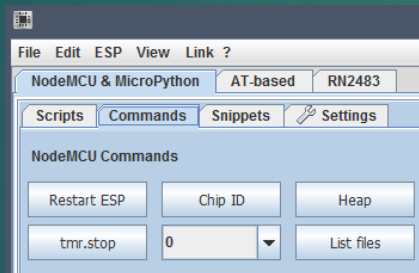


- ▶ NodeMCU on ESP8266

```
function getIP()  
    ip = wifi.sta.getip()  
    print(ip)  
end
```



- ▶ ESPlorer



ESP8266 – NodeMCU API

- ▶ UART is set to 9600 with no echo (`uart.setup(0,9600,8,0,1,0)`)
- ▶ Let ESP do some tasks by servicing some API's like:
 - ▶ `connectAP()` – Connect to access point
 - ▶ `sendData()` – Connect to host and send data

```
function connectAP(SSID,PASW,TIMEO)
  retry=0
  wifi.setmode(wifi.STATION)
  wifi.sta.config(SSID,PASW)
  wifi.sta.connect()
  tmr.alarm(0, 1000, 1, function()
    if (wifi.sta.getip() == nil) then
      --print("Busy...")
    else
      print('Connected')
      tmr.stop(0)
    end
    retry = retry + 1
    if (retry>TIMEO) then
      print('Failed')
      tmr.stop(0)
    end
  end)
end
```

```
function sendData(sHost,iPort,sUri)
  conn = nil
  conn = net.createConnection(net.TCP, 0)
  conn:on("receive", function(sck, c) print(c) end)
  conn:on("connection",
    function(conn, payload)
      --print("Connected")
      conn:send("GET " .. sUri .. " HTTP/1.1\r\n"
        .. "Host: " .. sHost .. ":" .. iPort .. "\r\n"
        .. "Connection: close\r\n"
        .. "User-Agent: Mozilla/4.0 (compatible; esp8266 GvS; Windows NT 5.1)\r\n"
        .. "Accept: */*\r\n\r\n")
    end)
  -- conn:on("disconnection", function(conn, payload) print('Disconnected') end)
  conn:connect(iPort,sHost)
end
```


Arduino code

```
boolean setup_ESP(){
  boolean bContinue=true;
  delay(500);
  ESP8266.print("\n\n\r\n");
  serial_flush_ESP();
  ESP8266.print("\nOK1\n\r\n");
  if(read_until_ESP("OK", sizeof("OK"), 2000, 1)){
    Serial.println("Talking to ESP");
  }
  delay(500);
  serial_flush_ESP();
  if (restartESP()==true)
    Serial.println("ESP Successfully restarted");

  delay(500);
  if (isESP_OK()==true) {
    Serial.println("ESP is responding ok");
  }
  else {
    bContinue=false;
  }

  if (bContinue)
    if (connectAP()==true) {
      Serial.println("ConnectAP=OK");
    }
    else {
      Serial.println("ConnectAP=NOK");
      bContinue=false;
    }

  if (bContinue) {
    delay(500);
    if (getESP_Status()=='5') {
      Serial.println("ESP up and running");
    }
    else {
      bContinue=false;
    }
  }
  serial_flush_ESP();
  return bContinue;
}
```

```
// Get wifi status
// 0: STA_IDLE,
// 1: STA_CONNECTING,
// 2: STA_WRONGPWD,
// 3: STA_APNOTFOUND,
// 4: STA_FAIL,
// 5: STA_GOTIP.
//
int getESP_Status() {
  serial_flush_ESP();
  int iRetval=0;
  ESP8266.print("\nwifi.sta.status()\r\n");
  if(read_until_ESP(keyword_rn, sizeof(keyword_rn), 1000, 1)) {
    //regel gevonden
    if (scratch_data_from_ESP[0]>0) {
      iRetval=scratch_data_from_ESP[1];
    }
  }
  else {
    Serial.println("ERR: wifi.sta.status");
  }
  return iRetval;
}
```

```
if (bBMP280) {
  temp280=bme.readTemperature();
  pressure280=bme.readPressure()/100;

  Serial.print(F("Temperature = "));
  Serial.print(temp280);
  Serial.println(" *C");

  Serial.print(F("Pressure = "));
  Serial.print(pressure280);
  Serial.println(" Pa");

  Serial.println();
}
```

```
// Sensor2
//json.htm?type=command&param=udevice&idx=275&nvalue=0&svalue=22.2;44.4;0
//
sUri = "/json.htm?type=command&param=udevice&idx=";
sUri = sUri + idSensor2 + "&nvalue=0&svalue=" + t + ";" + h + ";" + "0";
Serial.println(sUri);
delay(500);
if (sendData(csHost, ciPort, sUri)) {
  Serial.println("SendData2 ok");
}
```

Flow

Power-up → Check ESP Ready → SetupESP() → connectAP() → wait for connect

```
connectAP("MySSID","AP-Password",10)           getIP() 192.168.14.19
                                              =wifi.sta.status() 5
```


checkSensors → sendData()


```
=sendData("192.168.14.100",8084,"/json.htm?type=command&param=udevice&idx=274&nvalue=0&svalue=1023")
                                           /json.htm?type=command&param=udevice&idx=275&nvalue=0&svalue=24.30;31.50;0
```

```
HTTP/1.1 200 OK
Content-Length: 53
Content-Type:
application/json;charset=UTF-8
Cache-Control: no-cache
Pragma: no-cache
Access-Control-Allow-Origin: *
```


```
{
  "status" : "OK",
  "title" : "Update Device"
}
```


DHT22 **23.7° C / 31%**

 **Normal, Dew Point: 5.57° C**
Last Seen: 2017-01-19 17:57:24


 [Log](#) [Edit](#) [Notifications](#)


devBaro **1038.36 hPa**

 **1038.36 hPa, Prediction: Stable**
Last Seen: 2017-01-19 21:34:47

 [Log](#) [Edit](#) [Notifications](#)

devSens1 **321 units**

 **321 units**
Last Seen: 2017-01-19 17:59:45
Type: General, Custom Sensor

 [Log](#) [Edit](#) [Notifications](#)