INSTITUTO POLITECNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Instrumentación Virtual Aplicada

Proyecto final. Interfaz gráfica de energía con PyQT5 y Arduino.

Profesor: Dr. Jorge Fonseca Campos
Grupo: 4MV19

Alumno:

- Escárcega Corona Luis

Fecha de entrega: 8 enero de 2023

TABLA DE CONTENIDO

Requerimientos

- Interfaz gráfica en PyQt5.
- Medición de corriente y tensión en AC o DC.
- Gráficas de:
    - Tensión.
    - Corriente.
    - Gráfica de energía.
- Activación de pin digital al alcanzar un umbral.

Funcionamiento

La interfaz gráfica requiere la implementación de dos sensores, controlados por un microcontrolador ATMEGA328P en una placa de Arduino Uno, que monitorean la tensión y corriente eléctrica en la carga de DC de un circuito alimentado a 5V.

El procesamiento de las señales de tensión y corriente eléctrica permiten calcular y graficar la potencia instantánea y consumo de energía de dicho circuito. La interfaz solo permite que el usuario ingrese como único parámetro un valor "N" de consumo energético. Una vez el microcontrolador compute que se ha logrado el consumo "N" elegido; una terminal de salida digital del microcontrolador deberá encender un LED verde.

Los sensores ACS712 (corriente eléctrica) y FZ0430 (tensión) están conectados a los terminales analógicos A0 y A2 respectivamente; el LED verde indicador del umbral de energía está conectado a la terminal digital D7 del Arduino Uno.

El módulo del sensor ACS712 posee dos terminales para la alimentación a 5V y una salida de señal. Una bornera para su conexión en serie con la carga a medir. Internamente posee un circuito integrado de ocho terminales. Internamente contine un sensor de efecto Hall con cancelación de Offset dinámico, dos filtros activos pasa altas y pasa bajas en serie.

Ilustración 1. Módulo ACS712.

El circuito integrado requiere de un capacitor de filtrado externo para el acoplamiento de los filtros (C1) y un capacitor de filtrado para la alimentación (C2). Además, posee un LED testigo de alimentación (D1) y su respectiva resistencia limitadora de corriente (R1).

El modelo de 30A proporciona 66mV por cada Amper de medida o en otras palabras 0.066 v/A. La relación entre el voltaje de entrada a la terminal A0 y la corriente de la carga a medir está de la siguiente manera según la Ley de Corriente de Kirchhoff; donde S es la sensibilidad en $[\frac{V}{A}]$

$$V_{A0} = I * S + 2.5V = I * \left(\frac{0.066V}{A}\right) + 2.5V$$

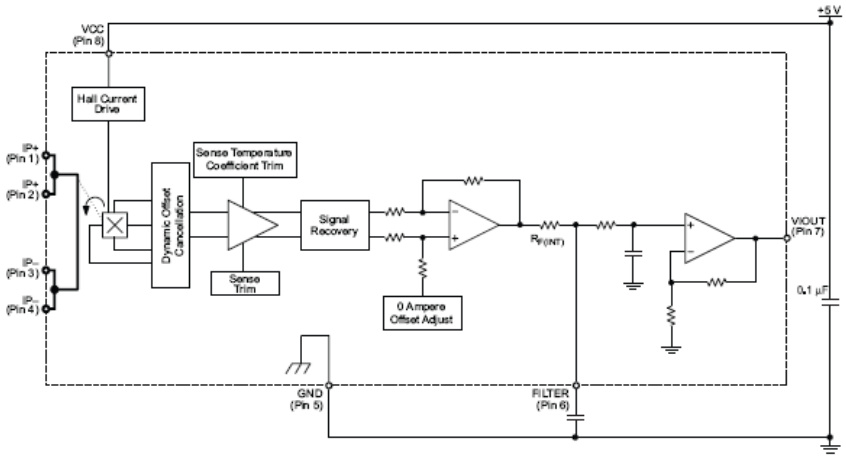

Ilustración 2. Diagrama eléctrico del IC ACS712.

El módulo del sensor FZ0430 es un arreglo de resistencias en divisor de voltaje donde $R_1 = 7.5k\Omega$ y $R_2 = 30k\Omega$. Aplicando ley de Kirchhoff de voltaje y despejando se tiene la siguiente relación entre el voltaje del Pin analógico A2 y el voltaje del nodo a medir.

$$V_{A2} = V_{MEDIR} * \left(\frac{R_2}{R_1 + R_2}\right) = V_{MEDIR} * \left(\frac{7.5k\Omega}{30k\Omega + 7.5k\Omega}\right) = V_{MEDIR} * \left(\frac{7.5k\Omega}{37.5k\Omega}\right) = V_{MEDIR} * 0.2$$

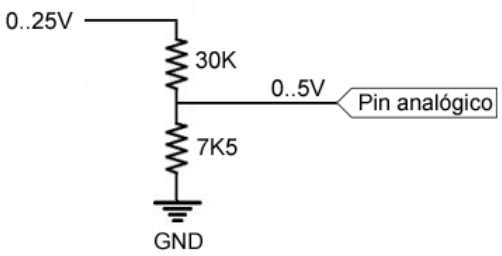

Ilustración 3. Diagrama eléctrico FZ0430.

Ilustración 4. Módulo de sensor FZ0430.

La carga para implementar es un LED brillante de 5mm blanco en serie con una resistencia de 1kΩ alimentados por una fuente conmutada de 5V externa a la alimentación del Arduino Uno. Las conexiones se muestran a continuación:

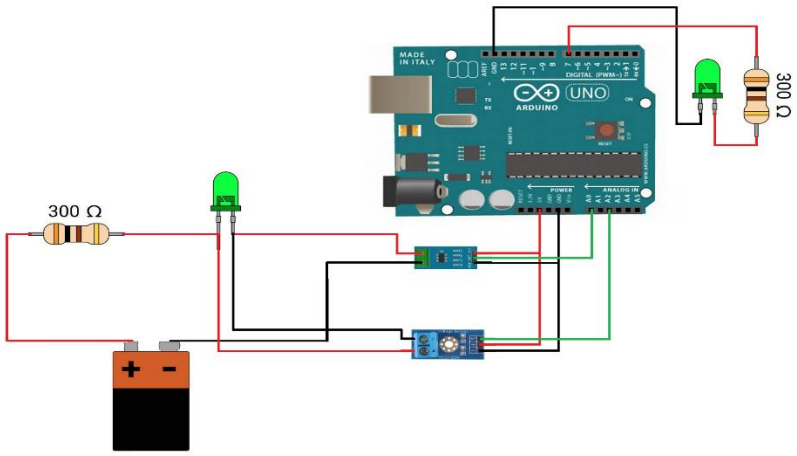

Ilustración 3. Diagrama eléctrico.

Front-End con PyQT5

El front-end de la interfaz gráfica con extensión .ui fue programada mediante el software Qt Designer de Anaconda para facilitar la distribución de gráficas.

La interfaz se compone de una ventana que cuenta con cuatro Graphics View y tres LCD Numbers para visualizar el voltaje, corriente, potencia y energía. Cuenta con un elemento virtual box o lista desplegable para establecer el valor de umbral de energía en W/h donde el led verde encenderá. Un botón de “Iniciar” que funcionará una vez enlazada la comunicación entre el Arduino y la aplicación y establecido el valor de umbral.

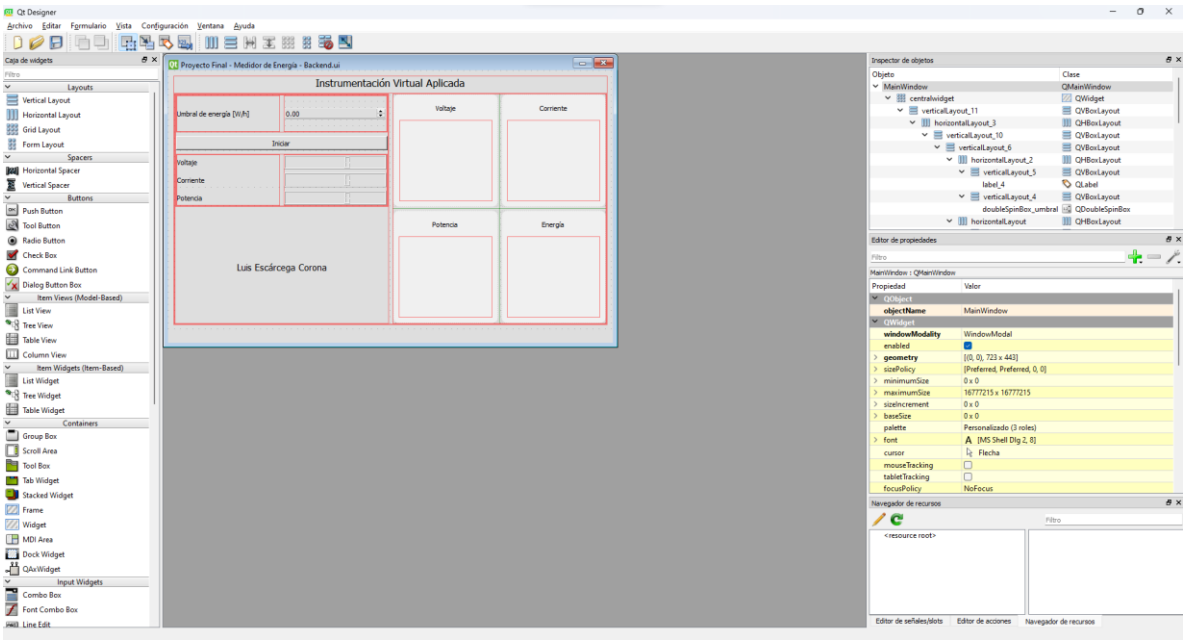

Ilustración 4. Entorno de desarrollo del Front-End de la interfaz gráfica.

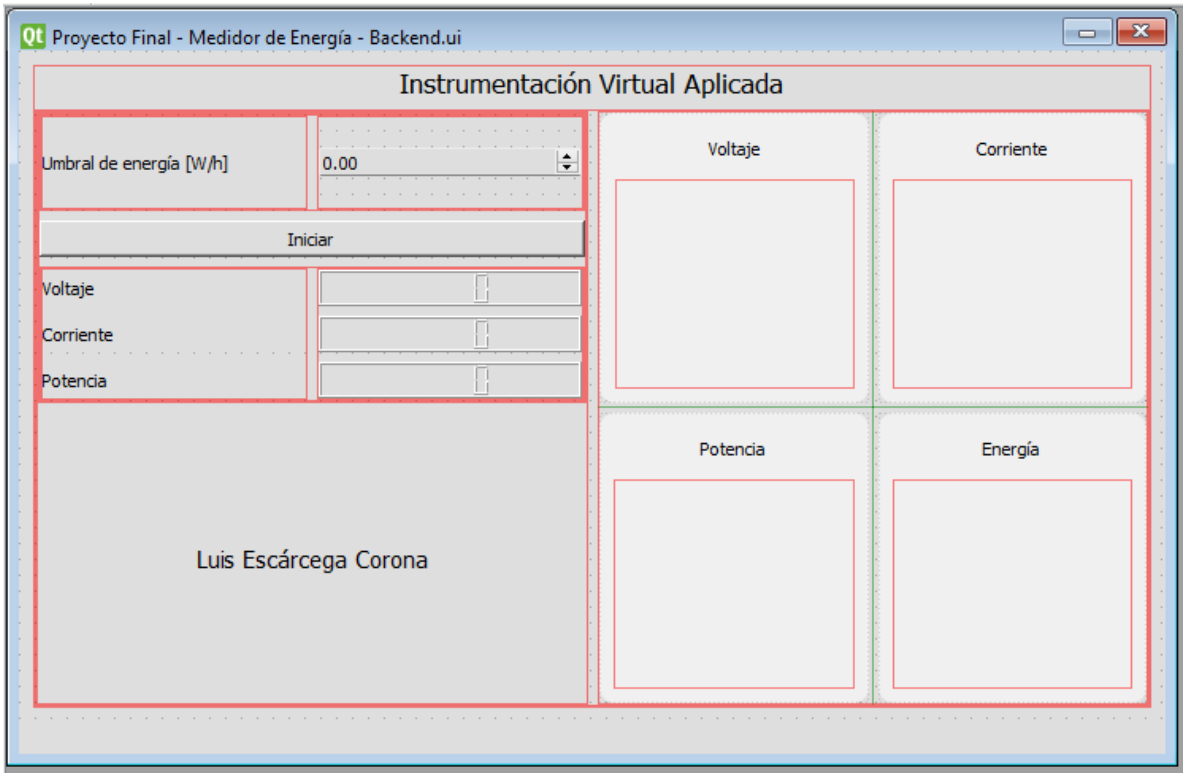

Ilustración 5. Vista de detalle de la interfaz gráfica.

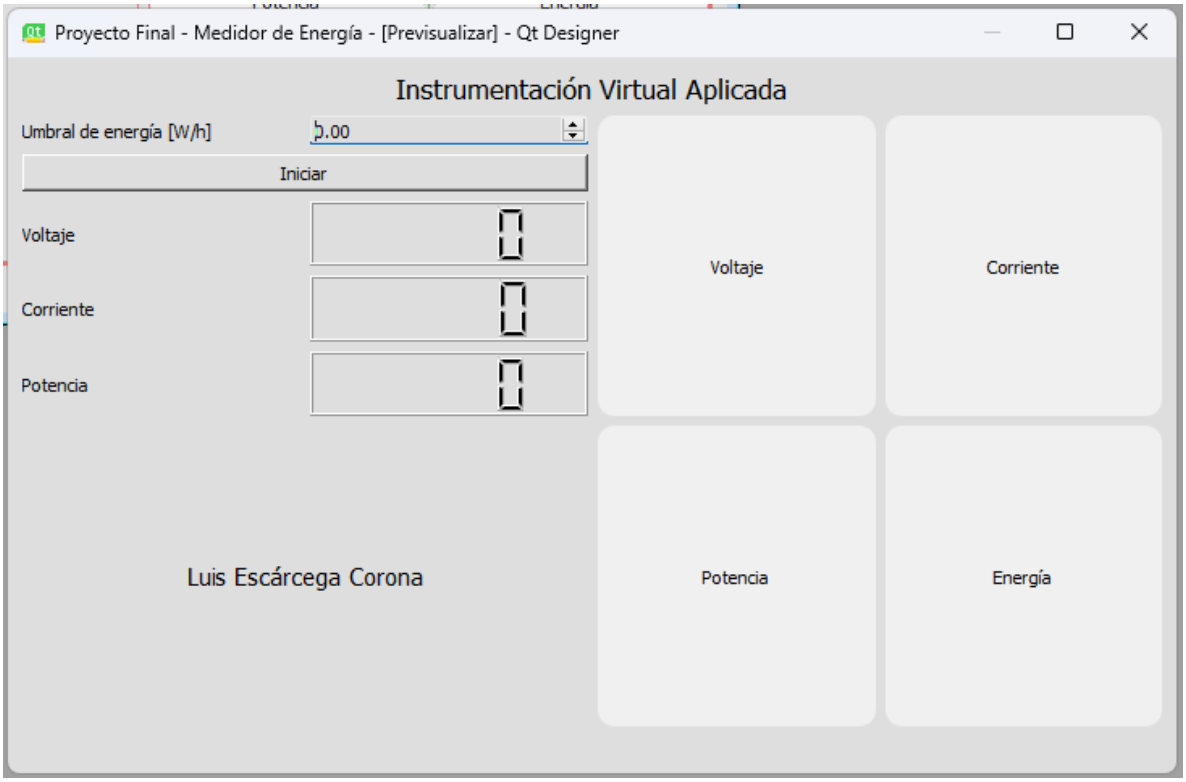

Ilustración 6. Previsualización de la interfaz gráfica.

Programación Back-End con Visual Studio Code

El back-end se compone de dos códigos en lenguajes C y Python que controlan al Arduino y el ejecutable, respectivamente.

La programación en C en el IDE de Arduino es un código de protocolo de comunicación de la librería Firmata que simplifica la comunicación entre Arduino y el computador. Este se muestra en el Anexo 1.

La programación del código, en Python, para la interfaz gráfica se realizó en el entorno de Visual Studio Code. (Debido a una confusión aparecen los nombres de los archivos “Backend” y “Frontend” invertidos, es decir el Backend en realidad es la interfaz multimedia y viceversa).

En una misma carpeta se guardó el archivo con extensión “.ui” del front-end realizado en QtDesigner, la conversión del front-end en extensión “.py” y el archivo Main con las clases que calculan voltaje y corriente, además de cuatro clases para cada gráfica.

El programa inicia conectando la computadora con la tarjeta Arduino a través del protocolo Firmata, definiendo los pines de entrada y salida.

```
board = pyfirmata.Arduino('COM3')
iter8 = pyfirmata.util.Iterator(board)
iter8.start()
led1 = board.get_pin('d:7:o')
sensvoltage = board.get_pin('a:0:i')
senscurrent = board.get_pin('a:2:i')
print("CONEXION CON ARDUINO")
```

El programa inicializa la interfaz gráfica en espera del evento del SpinBox para el valor del umbral y la conexión del botón Inicio. El botón inicio inicializa el método “Inicio” que genera un vector de tiempo indefinido y obtiene el valor del umbral; dentro del mismo método “Inicio” se inicializa el método getvoltage y getcurrent para crear el vector de valores de voltaje y corriente. Haciendo uso del comendo np.multiply() se obtiene la potencia. Mediante las funciones sum() y len() se obtiene un valor promedio de voltaje, corriente y potencia para desplegar en los LCD. Finalmente se calcula la energía instantánea y se crean cuatro objetos con las clases Canvas_voltage, Canvas_current, Canvas_power y Canvas_energy dentro de un ciclo while que compara el valor del umbral con la suma de la energía.

```
def inicio(self):
        print("Inicio")
        umbral = self.ui.doubleSpinBox_umbral.value()
        np.t = []
        # 1 Obtener Voltaje
        self.yv = self.getvoltage(self, sensvoltage, 0, 1023, 0.0, 25.0)
        self.yvpromedio = sum(self.getvoltage)/len(self.getvoltage)
        print("Voltaje obtenido")
        # 2 Obtener Corriente
```

```
        self.yc = self.getcurrent(self, senscurrent)
        self.ycpromedio = sum(self.yc)/len(self.yc)
        print("Corriente obtenido")
        # 3 Obtener Potencia
        self.potencia = np.multiply(self.yc,self.yv)
        self.potenciapromedio = sum(self.potencia)/len(self.potencia)
        print("Potencia obtenida")
        # 4 Desplegar en LCD
        self.ui.lcd_voltaje.display(self.yvpromedio)
        self.ui.lcd_current.display(self.ycpromedio)
        self.ui.lcd_power.display(self.potenciapromedio)
        print("Valores en LCD")
        # 5 Obtener Energía yp*np.t
        self.ye = self.yp*np.t
        print("Energía obtenida")
        # 6 Graficar
        self.grafica1 = Canvas_voltage()
        self.grafica2 = Canvas_corriente()
        self.grafica3 = Canvas_potencia()
        self.grafica4 = Canvas_energia()
        while sum(self.ye) < umbral:
            self.ui.grafica_voltaje.addWidget(self.grafica1(self,self.yv))
            self.ui.grafica_corriente.addWidget(self.grafica2(self,self.yc))
            self.ui.grafica_potencia.addWidget(self.grafica3(self,self.yp))
            self.ui.grafica_energia.addWidget(self.grafica4(self,self.ye))
            if (sum(self.ye) < umbral):
                led1.write(1)
                time.sleep(5)
                break
        print("Gráficas obtenidas")
```

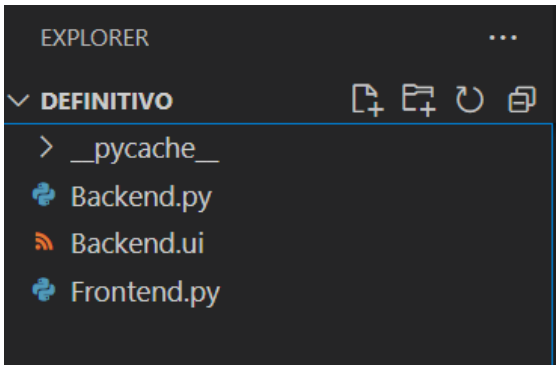

Ilustración 7. Archivos de Backend y Frontend.

Los códigos completos del Backend.ui, Backend.py y Frontend.py se encuentran en los Anexos 2, 3 y 4 respectivamente.

Pruebas

El código de Arduino se carga a través del Arduino IDE. Posterior se compila el programa Main que genera la interfaz de usuario e imprime en la terminal el estado de conexión de Arduino en espera de ingresar el valor de umbral y la activación del botón “Iniciar”.

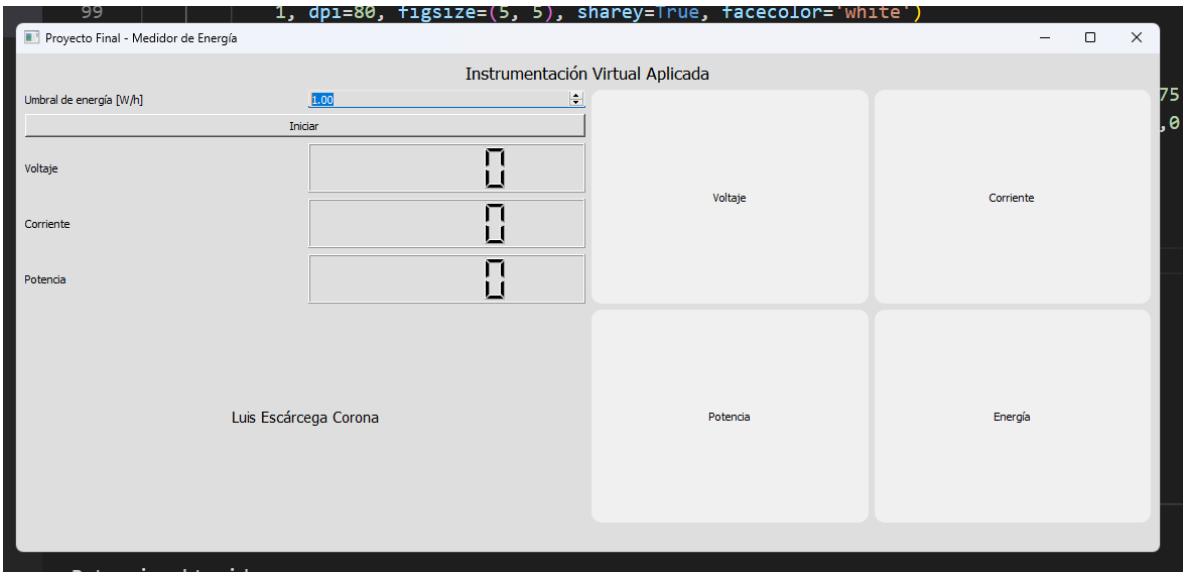

Ilustración 8. Interfaz gráfica en espera de valores e inicio.

Una vez iniciado el programa comenzará a graficar hasta alcanzar el umbral, dejará de graficar y activará el LED del pin digital 7.

Ilustración 9. Programa que ha llegado al Watt /Hora después de 15 minutos.

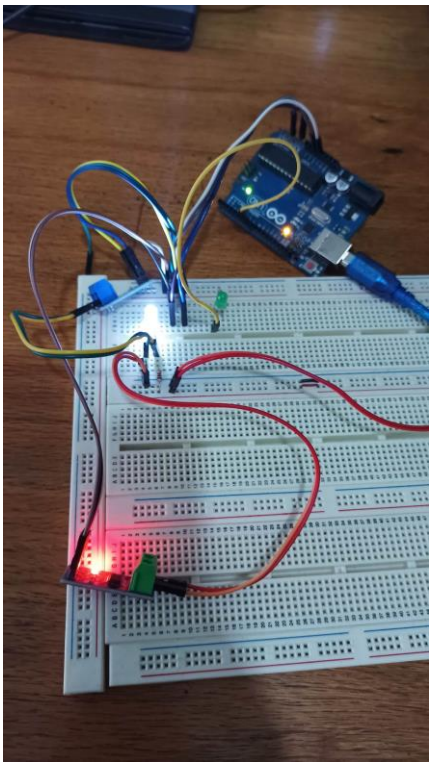

Ilustración 10. Circuito electrónico antes de lograr el umbral.

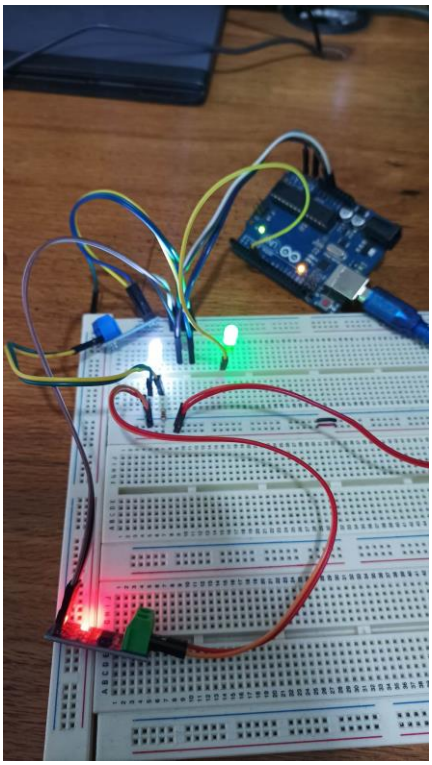

Ilustración 11. Circuito electrónico después de alcanzar el umbral.

Anexo 1. StandardFirmata.ino

```
/*
  Firmata is a generic protocol for communicating with microcontrollers
  from software on a host computer. It is intended to work with
  any host computer software package.

  To download a host software package, please click on the following link
  to open the list of Firmata client libraries in your default browser.

  https://github.com/firmata/arduino#firmata-client-libraries

  Copyright (C) 2006-2008 Hans-Christoph Steiner.  All rights reserved.
  Copyright (C) 2010-2011 Paul Stoffregen.  All rights reserved.
  Copyright (C) 2009 Shigeru Kobayashi.  All rights reserved.
  Copyright (C) 2009-2016 Jeff Hoefs.  All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  See file LICENSE.txt for further informations on licensing terms.

  Last updated August 17th, 2017
*/

#include <Servo.h>
#include <Wire.h>
#include <Firmata.h>

#define I2C_WRITE                   B00000000
#define I2C_READ                    B00001000
#define I2C_READ_CONTINUOUSLY       B00010000
#define I2C_STOP_READING            B00011000
#define I2C_READ_WRITE_MODE_MASK    B00011000
#define I2C_10BIT_ADDRESS_MODE_MASK B00100000
#define I2C_END_TX_MASK             B01000000
#define I2C_STOP_TX                 1
#define I2C_RESTART_TX              0
#define I2C_MAX_QUERIES             8
#define I2C_REGISTER_NOT_SPECIFIED  -1

// the minimum interval for sampling analog input
#define MINIMUM_SAMPLING_INTERVAL   1



/*==============================================================================
 * GLOBAL VARIABLES
 *============================================================================*/

#ifdef FIRMATA_SERIAL_FEATURE
SerialFirmata serialFeature;
#endif

/* analog inputs */
int analogInputsToReport = 0; // bitwise array to store pin reporting

/* digital input ports */
byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence
byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent

/* pins configuration */
byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else

/* timer variables */
unsigned long currentMillis;        // store the current value from millis()
```

```
unsigned long previousMillis;       // for comparison with currentMillis
unsigned int samplingInterval = 19; // how often to run the main loop (in ms)

/* i2c data */
struct i2c_device_info {
  byte addr;
  int reg;
  byte bytes;
  byte stopTX;
};

/* for i2c read continuous more */
i2c_device_info query[I2C_MAX_QUERIES];

byte i2cRxData[64];
boolean isI2CEnabled = false;
signed char queryIndex = -1;
// default delay time between i2c read request and Wire.requestFrom()
unsigned int i2cReadDelayTime = 0;

Servo servos[MAX_SERVOS];
byte servoPinMap[TOTAL_PINS];
byte detachedServos[MAX_SERVOS];
byte detachedServoCount = 0;
byte servoCount = 0;

boolean isResetting = false;

// Forward declare a few functions to avoid compiler errors with older versions
// of the Arduino IDE.
void setPinModeCallback(byte, int);
void reportAnalogCallback(byte analogPin, int value);
void sysexCallback(byte, byte, byte*);

/* utility functions */
void wireWrite(byte data)
{
#if ARDUINO >= 100
  Wire.write((byte)data);
#else
  Wire.send(data);
#endif
}

byte wireRead(void)
{
#if ARDUINO >= 100
  return Wire.read();
#else
  return Wire.receive();
#endif
}

/*==============================================================================
 * FUNCTIONS
 *============================================================================*/

void attachServo(byte pin, int minPulse, int maxPulse)
{
  if (servoCount < MAX_SERVOS) {
    // reuse indexes of detached servos until all have been reallocated
    if (detachedServoCount > 0) {
      servoPinMap[pin] = detachedServos[detachedServoCount - 1];
      if (detachedServoCount > 0) detachedServoCount--;
    } else {
      servoPinMap[pin] = servoCount;
      servoCount++;
    }
    if (minPulse > 0 && maxPulse > 0) {
```

```
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
    } else {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
    }
  } else {
    Firmata.sendString("Max servos attached");
  }
}

void detachServo(byte pin)
{
  servos[servoPinMap[pin]].detach();
  // if we're detaching the last servo, decrement the count
  // otherwise store the index of the detached servo
  if (servoPinMap[pin] == servoCount && servoCount > 0) {
    servoCount--;
  } else if (servoCount > 0) {
    // keep track of detached servos because we want to reuse their indexes
    // before incrementing the count of attached servos
    detachedServoCount++;
    detachedServos[detachedServoCount - 1] = servoPinMap[pin];
  }

  servoPinMap[pin] = 255;
}

void enableI2CPins()
{
  byte i;
  // is there a faster way to do this? would probaby require importing
  // Arduino.h to get SCL and SDA pins
  for (i = 0; i < TOTAL_PINS; i++) {
    if (IS_PIN_I2C(i)) {
      // mark pins as i2c so they are ignore in non i2c data requests
      setPinModeCallback(i, PIN_MODE_I2C);
    }
  }

  isI2CEnabled = true;

  Wire.begin();
}

/* disable the i2c pins so they can be used for other functions */
void disableI2CPins() {
  isI2CEnabled = false;
  // disable read continuous mode for all devices
  queryIndex = -1;
}

void readAndReportData(byte address, int theRegister, byte numBytes, byte stopTX) {
  // allow I2C requests that don't require a register read
  // for example, some devices using an interrupt pin to signify new data available
  // do not always require the register read so upon interrupt you call Wire.requestFrom()
  if (theRegister != I2C_REGISTER_NOT_SPECIFIED) {
    Wire.beginTransmission(address);
    wireWrite((byte)theRegister);
    Wire.endTransmission(stopTX); // default = true
    // do not set a value of 0
    if (i2cReadDelayTime > 0) {
      // delay is necessary for some devices such as WiiNunchuck
      delayMicroseconds(i2cReadDelayTime);
    }
  } else {
    theRegister = 0;  // fill the register with a dummy value
  }

  Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom
```

```
  // check to be sure correct number of bytes were returned by slave
  if (numBytes < Wire.available()) {
    Firmata.sendString("I2C: Too many bytes received");
  } else if (numBytes > Wire.available()) {
    Firmata.sendString("I2C: Too few bytes received");
    numBytes = Wire.available();
  }

  i2cRxData[0] = address;
  i2cRxData[1] = theRegister;

  for (int i = 0; i < numBytes && Wire.available(); i++) {
    i2cRxData[2 + i] = wireRead();
  }

  // send slave address, register and received bytes
  Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
}

void outputPort(byte portNumber, byte portValue, byte forceSend)
{
  // pins not configured as INPUT are cleared to zeros
  portValue = portValue & portConfigInputs[portNumber];
  // only send if the value is different than previously sent
  if (forceSend || previousPINs[portNumber] != portValue) {
    Firmata.sendDigitalPort(portNumber, portValue);
    previousPINs[portNumber] = portValue;
  }
}

/* -----------------------------------------------------------------------------
 * check all the active digital inputs for change of state, then add any events
 * to the Serial output queue using Serial.print() */
void checkDigitalInputs(void)
{
  /* Using non-looping code allows constants to be given to readPort().
   * The compiler will apply substantial optimizations if the inputs
   * to readPort() are compile-time constants. */
  if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]), false);
  if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]), false);
  if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]), false);
  if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]), false);
  if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]), false);
  if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]), false);
  if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]), false);
  if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]), false);
  if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]), false);
  if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]), false);
  if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10, portConfigInputs[10]), false);
  if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11, portConfigInputs[11]), false);
  if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12, portConfigInputs[12]), false);
  if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13, portConfigInputs[13]), false);
  if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14, portConfigInputs[14]), false);
  if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15, portConfigInputs[15]), false);
}

// -----------------------------------------------------------------------------
/* sets the pin mode to the correct state and sets the relevant bits in the
 * two bit-arrays that track Digital I/O and PWM status
 */
void setPinModeCallback(byte pin, int mode)
{
  if (Firmata.getPinMode(pin) == PIN_MODE_IGNORE)
    return;

  if (Firmata.getPinMode(pin) == PIN_MODE_I2C && isI2CEnabled && mode != PIN_MODE_I2C) {
    // disable i2c so pins can be used for other functions
    // the following if statements should reconfigure the pins properly
    disableI2CPins();
```

```
  }
  if (IS_PIN_DIGITAL(pin) && mode != PIN_MODE_SERVO) {
    if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
      detachServo(pin);
    }
  }
  if (IS_PIN_ANALOG(pin)) {
    reportAnalogCallback(PIN_TO_ANALOG(pin), mode == PIN_MODE_ANALOG ? 1 : 0); // turn on/off
reporting
  }
  if (IS_PIN_DIGITAL(pin)) {
    if (mode == INPUT || mode == PIN_MODE_PULLUP) {
      portConfigInputs[pin / 8] |= (1 << (pin & 7));
    } else {
      portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
    }
  }
  Firmata.setPinState(pin, 0);
  switch (mode) {
    case PIN_MODE_ANALOG:
      if (IS_PIN_ANALOG(pin)) {
        if (IS_PIN_DIGITAL(pin)) {
          pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
          // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
          digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        }
        Firmata.setPinMode(pin, PIN_MODE_ANALOG);
      }
      break;
    case INPUT:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
        // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        Firmata.setPinMode(pin, INPUT);
      }
      break;
    case PIN_MODE_PULLUP:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT_PULLUP);
        Firmata.setPinMode(pin, PIN_MODE_PULLUP);
        Firmata.setPinState(pin, 1);
      }
      break;
    case OUTPUT:
      if (IS_PIN_DIGITAL(pin)) {
        if (Firmata.getPinMode(pin) == PIN_MODE_PWM) {
          // Disable PWM if pin mode was previously set to PWM.
          digitalWrite(PIN_TO_DIGITAL(pin), LOW);
        }
        pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
        Firmata.setPinMode(pin, OUTPUT);
      }
      break;
    case PIN_MODE_PWM:
      if (IS_PIN_PWM(pin)) {
        pinMode(PIN_TO_PWM(pin), OUTPUT);
        analogWrite(PIN_TO_PWM(pin), 0);
        Firmata.setPinMode(pin, PIN_MODE_PWM);
      }
      break;
    case PIN_MODE_SERVO:
      if (IS_PIN_DIGITAL(pin)) {
        Firmata.setPinMode(pin, PIN_MODE_SERVO);
        if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
```

```
          // pass -1 for min and max pulse values to use default values set
          // by Servo library
          attachServo(pin, -1, -1);
        }
      }
      break;
    case PIN_MODE_I2C:
      if (IS_PIN_I2C(pin)) {
        // mark the pin as i2c
        // the user must call I2C_CONFIG to enable I2C for a device
        Firmata.setPinMode(pin, PIN_MODE_I2C);
      }
      break;
    case PIN_MODE_SERIAL:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handlePinMode(pin, PIN_MODE_SERIAL);
#endif
      break;
    default:
      Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
  }
  // TODO: save status to EEPROM here, if changed
}

/*
 * Sets the value of an individual pin. Useful if you want to set a pin value but
 * are not tracking the digital port state.
 * Can only be used on pins configured as OUTPUT.
 * Cannot be used to enable pull-ups on Digital INPUT pins.
 */
void setPinValueCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS && IS_PIN_DIGITAL(pin)) {
    if (Firmata.getPinMode(pin) == OUTPUT) {
      Firmata.setPinState(pin, value);
      digitalWrite(PIN_TO_DIGITAL(pin), value);
    }
  }
}

void analogWriteCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS) {
    switch (Firmata.getPinMode(pin)) {
      case PIN_MODE_SERVO:
        if (IS_PIN_DIGITAL(pin))
          servos[servoPinMap[pin]].write(value);
        Firmata.setPinState(pin, value);
        break;
      case PIN_MODE_PWM:
        if (IS_PIN_PWM(pin))
          analogWrite(PIN_TO_PWM(pin), value);
        Firmata.setPinState(pin, value);
        break;
    }
  }
}

void digitalWriteCallback(byte port, int value)
{
  byte pin, lastPin, pinValue, mask = 1, pinWriteMask = 0;

  if (port < TOTAL_PORTS) {
    // create a mask of the pins on this port that are writable.
    lastPin = port * 8 + 8;
    if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
    for (pin = port * 8; pin < lastPin; pin++) {
      // do not disturb non-digital pins (eg, Rx & Tx)
      if (IS_PIN_DIGITAL(pin)) {
```

```
        // do not touch pins in PWM, ANALOG, SERVO or other modes
        if (Firmata.getPinMode(pin) == OUTPUT || Firmata.getPinMode(pin) == INPUT) {
          pinValue = ((byte)value & mask) ? 1 : 0;
          if (Firmata.getPinMode(pin) == OUTPUT) {
            pinWriteMask |= mask;
          } else if (Firmata.getPinMode(pin) == INPUT && pinValue == 1 && Firmata.getPinState(pin)
!= 1) {
            // only handle INPUT here for backwards compatibility
#if ARDUINO > 100
            pinMode(pin, INPUT_PULLUP);
#else
            // only write to the INPUT pin to enable pullups if Arduino v1.0.0 or earlier
            pinWriteMask |= mask;
#endif
          }
          Firmata.setPinState(pin, pinValue);
        }
      }
      mask = mask << 1;
    }
    writePort(port, (byte)value, pinWriteMask);
  }
}



// -----------------------------------------------------------------------------
/* sets bits in a bit array (int) to toggle the reporting of the analogIns
 */
//void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
//}
void reportAnalogCallback(byte analogPin, int value)
{
  if (analogPin < TOTAL_ANALOG_PINS) {
    if (value == 0) {
      analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);
    } else {
      analogInputsToReport = analogInputsToReport | (1 << analogPin);
      // prevent during system reset or all analog pin values will be reported
      // which may report noise for unconnected analog pins
      if (!isResetting) {
        // Send pin value immediately. This is helpful when connected via
        // ethernet, wi-fi or bluetooth so pin states can be known upon
        // reconnecting.
        Firmata.sendAnalog(analogPin, analogRead(analogPin));
      }
    }
  }
  // TODO: save status to EEPROM here, if changed
}

void reportDigitalCallback(byte port, int value)
{
  if (port < TOTAL_PORTS) {
    reportPINs[port] = (byte)value;
    // Send port value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
  }
  // do not disable analog reporting on these 8 pins, to allow some
  // pins used for digital, others analog.  Instead, allow both types
  // of reporting to be enabled, but check if the pin is configured
  // as analog when sampling the analog inputs.  Likewise, while
  // scanning digital pins, portConfigInputs will mask off values from any
  // pins configured as analog
}

/*==============================================================================
```

```
 * SYSEX-BASED commands
 *============================================================================*/

void sysexCallback(byte command, byte argc, byte *argv)
{
  byte mode;
  byte stopTX;
  byte slaveAddress;
  byte data;
  int slaveRegister;
  unsigned int delayTime;

  switch (command) {
    case I2C_REQUEST:
      mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
      if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
        Firmata.sendString("10-bit addressing not supported");
        return;
      }
      else {
        slaveAddress = argv[0];
      }

      // need to invert the logic here since 0 will be default for client
      // libraries that have not updated to add support for restart tx
      if (argv[1] & I2C_END_TX_MASK) {
        stopTX = I2C_RESTART_TX;
      }
      else {
        stopTX = I2C_STOP_TX; // default
      }

      switch (mode) {
        case I2C_WRITE:
          Wire.beginTransmission(slaveAddress);
          for (byte i = 2; i < argc; i += 2) {
            data = argv[i] + (argv[i + 1] << 7);
            wireWrite(data);
          }
          Wire.endTransmission();
          delayMicroseconds(70);
          break;
        case I2C_READ:
          if (argc == 6) {
            // a slave register is specified
            slaveRegister = argv[2] + (argv[3] << 7);
            data = argv[4] + (argv[5] << 7);  // bytes to read
          }
          else {
            // a slave register is NOT specified
            slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
            data = argv[2] + (argv[3] << 7);  // bytes to read
          }
          readAndReportData(slaveAddress, (int)slaveRegister, data, stopTX);
          break;
        case I2C_READ_CONTINUOUSLY:
          if ((queryIndex + 1) >= I2C_MAX_QUERIES) {
            // too many queries, just ignore
            Firmata.sendString("too many queries");
            break;
          }
          if (argc == 6) {
            // a slave register is specified
            slaveRegister = argv[2] + (argv[3] << 7);
            data = argv[4] + (argv[5] << 7);  // bytes to read
          }
          else {
            // a slave register is NOT specified
            slaveRegister = (int)I2C_REGISTER_NOT_SPECIFIED;
```

```
          data = argv[2] + (argv[3] << 7);  // bytes to read
        }
        queryIndex++;
        query[queryIndex].addr = slaveAddress;
        query[queryIndex].reg = slaveRegister;
        query[queryIndex].bytes = data;
        query[queryIndex].stopTX = stopTX;
        break;
      case I2C_STOP_READING:
        byte queryIndexToSkip;
        // if read continuous mode is enabled for only 1 i2c device, disable
        // read continuous reporting for that device
        if (queryIndex <= 0) {
          queryIndex = -1;
        } else {
          queryIndexToSkip = 0;
          // if read continuous mode is enabled for multiple devices,
          // determine which device to stop reading and remove it's data from
          // the array, shifiting other array data to fill the space
          for (byte i = 0; i < queryIndex + 1; i++) {
            if (query[i].addr == slaveAddress) {
              queryIndexToSkip = i;
              break;
            }
          }

          for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
            if (i < I2C_MAX_QUERIES) {
              query[i].addr = query[i + 1].addr;
              query[i].reg = query[i + 1].reg;
              query[i].bytes = query[i + 1].bytes;
              query[i].stopTX = query[i + 1].stopTX;
            }
          }
          queryIndex--;
        }
        break;
      default:
        break;
    }
    break;
  case I2C_CONFIG:
    delayTime = (argv[0] + (argv[1] << 7));

    if (argc > 1 && delayTime > 0) {
      i2cReadDelayTime = delayTime;
    }

    if (!isI2CEnabled) {
      enableI2CPins();
    }

    break;
  case SERVO_CONFIG:
    if (argc > 4) {
      // these vars are here for clarity, they'll optimized away by the compiler
      byte pin = argv[0];
      int minPulse = argv[1] + (argv[2] << 7);
      int maxPulse = argv[3] + (argv[4] << 7);

      if (IS_PIN_DIGITAL(pin)) {
        if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
          detachServo(pin);
        }
        attachServo(pin, minPulse, maxPulse);
        setPinModeCallback(pin, PIN_MODE_SERVO);
      }
    }
    break;
```

```
    case SAMPLING_INTERVAL:
      if (argc > 1) {
        samplingInterval = argv[0] + (argv[1] << 7);
        if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
          samplingInterval = MINIMUM_SAMPLING_INTERVAL;
        }
      } else {
        //Firmata.sendString("Not enough data");
      }
      break;
    case EXTENDED_ANALOG:
      if (argc > 1) {
        int val = argv[1];
        if (argc > 2) val |= (argv[2] << 7);
        if (argc > 3) val |= (argv[3] << 14);
        analogWriteCallback(argv[0], val);
      }
      break;
    case CAPABILITY_QUERY:
      Firmata.write(START_SYSEX);
      Firmata.write(CAPABILITY_RESPONSE);
      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
        if (IS_PIN_DIGITAL(pin)) {
          Firmata.write((byte)INPUT);
          Firmata.write(1);
          Firmata.write((byte)PIN_MODE_PULLUP);
          Firmata.write(1);
          Firmata.write((byte)OUTPUT);
          Firmata.write(1);
        }
        if (IS_PIN_ANALOG(pin)) {
          Firmata.write(PIN_MODE_ANALOG);
          Firmata.write(10); // 10 = 10-bit resolution
        }
        if (IS_PIN_PWM(pin)) {
          Firmata.write(PIN_MODE_PWM);
          Firmata.write(DEFAULT_PWM_RESOLUTION);
        }
        if (IS_PIN_DIGITAL(pin)) {
          Firmata.write(PIN_MODE_SERVO);
          Firmata.write(14);
        }
        if (IS_PIN_I2C(pin)) {
          Firmata.write(PIN_MODE_I2C);
          Firmata.write(1);  // TODO: could assign a number to map to SCL or SDA
        }
#ifdef FIRMATA_SERIAL_FEATURE
        serialFeature.handleCapability(pin);
#endif
        Firmata.write(127);
      }
      Firmata.write(END_SYSEX);
      break;
    case PIN_STATE_QUERY:
      if (argc > 0) {
        byte pin = argv[0];
        Firmata.write(START_SYSEX);
        Firmata.write(PIN_STATE_RESPONSE);
        Firmata.write(pin);
        if (pin < TOTAL_PINS) {
          Firmata.write(Firmata.getPinMode(pin));
          Firmata.write((byte)Firmata.getPinState(pin) & 0x7F);
          if (Firmata.getPinState(pin) & 0xFF80) Firmata.write((byte)(Firmata.getPinState(pin) >> 7)
& 0x7F);
          if (Firmata.getPinState(pin) & 0xC000) Firmata.write((byte)(Firmata.getPinState(pin) >>
14) & 0x7F);
        }
        Firmata.write(END_SYSEX);
      }
```

```
      break;
    case ANALOG_MAPPING_QUERY:
      Firmata.write(START_SYSEX);
      Firmata.write(ANALOG_MAPPING_RESPONSE);
      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
        Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
      }
      Firmata.write(END_SYSEX);
      break;

    case SERIAL_MESSAGE:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handleSysex(command, argc, argv);
#endif
      break;
  }
}

/*==============================================================================
 * SETUP()
 *============================================================================*/

void systemResetCallback()
{
  isResetting = true;

  // initialize a defalt state
  // TODO: option to load config from EEPROM instead of default

#ifdef FIRMATA_SERIAL_FEATURE
  serialFeature.reset();
#endif

  if (isI2CEnabled) {
    disableI2CPins();
  }

  for (byte i = 0; i < TOTAL_PORTS; i++) {
    reportPINs[i] = false;    // by default, reporting off
    portConfigInputs[i] = 0;  // until activated
    previousPINs[i] = 0;
  }

  for (byte i = 0; i < TOTAL_PINS; i++) {
    // pins with analog capability default to analog input
    // otherwise, pins default to digital output
    if (IS_PIN_ANALOG(i)) {
      // turns off pullup, configures everything
      setPinModeCallback(i, PIN_MODE_ANALOG);
    } else if (IS_PIN_DIGITAL(i)) {
      // sets the output to 0, configures portConfigInputs
      setPinModeCallback(i, OUTPUT);
    }

    servoPinMap[i] = 255;
  }
  // by default, do not report any analog inputs
  analogInputsToReport = 0;

  detachedServoCount = 0;
  servoCount = 0;

  /* send digital inputs to set the initial state on the host computer,
   * since once in the loop(), this firmware will only send on change */
  /*
  TODO: this can never execute, since no pins default to digital input
        but it will be needed when/if we support EEPROM stored config
  for (byte i=0; i < TOTAL_PORTS; i++) {
    outputPort(i, readPort(i, portConfigInputs[i]), true);
```

```
  }
  */
  isResetting = false;
}

void setup()
{
  Firmata.setFirmwareVersion(FIRMATA_FIRMWARE_MAJOR_VERSION, FIRMATA_FIRMWARE_MINOR_VERSION);

  Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
  Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
  Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
  Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
  Firmata.attach(SET_PIN_MODE, setPinModeCallback);
  Firmata.attach(SET_DIGITAL_PIN_VALUE, setPinValueCallback);
  Firmata.attach(START_SYSEX, sysexCallback);
  Firmata.attach(SYSTEM_RESET, systemResetCallback);

  // to use a port other than Serial, such as Serial1 on an Arduino Leonardo or Mega,
  // Call begin(baud) on the alternate serial port and pass it to Firmata to begin like this:
  // Serial1.begin(57600);
  // Firmata.begin(Serial1);
  // However do not do this if you are using SERIAL_MESSAGE

  Firmata.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for ATmega32u4-based boards and Arduino 101
  }

  systemResetCallback();  // reset to default config
}

/*==============================================================================
 * LOOP()
 *============================================================================*/
void loop()
{
  byte pin, analogPin;

  /* DIGITALREAD - as fast as possible, check for changes and output them to the
   * FTDI buffer using Serial.print()  */
  checkDigitalInputs();

  /* STREAMREAD - processing incoming messagse as soon as possible, while still
   * checking digital inputs.  */
  while (Firmata.available())
    Firmata.processInput();

  // TODO - ensure that Stream buffer doesn't go over 60 bytes

  currentMillis = millis();
  if (currentMillis - previousMillis > samplingInterval) {
    previousMillis += samplingInterval;
    /* ANALOGREAD - do all analogReads() at the configured sampling interval */
    for (pin = 0; pin < TOTAL_PINS; pin++) {
      if (IS_PIN_ANALOG(pin) && Firmata.getPinMode(pin) == PIN_MODE_ANALOG) {
        analogPin = PIN_TO_ANALOG(pin);
        if (analogInputsToReport & (1 << analogPin)) {
          Firmata.sendAnalog(analogPin, analogRead(analogPin));
        }
      }
    }
    // report i2c data for all device with read continuous mode enabled
    if (queryIndex > -1) {
      for (byte i = 0; i < queryIndex + 1; i++) {
        readAndReportData(query[i].addr, query[i].reg, query[i].bytes, query[i].stopTX);
      }
    }
  }
```

```
#ifdef FIRMATA_SERIAL_FEATURE
  serialFeature.update();
#endif
}
```

Anexo 2. Backend.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="windowModality">
   <enum>Qt::WindowModal</enum>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>723</width>
    <height>443</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Proyecto Final - Medidor de Energía</string>
  </property>
  <property name="styleSheet">
   <string notr="true">background-color: rgb(222, 222, 222);</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout_2">
    <item row="0" column="0">
     <layout class="QVBoxLayout" name="verticalLayout_11">
      <item>
       <widget class="QLabel" name="label">
        <property name="font">
         <font>
          <pointsize>13</pointsize>
         </font>
        </property>
        <property name="accessibleName">
         <string>0</string>
        </property>
        <property name="layoutDirection">
         <enum>Qt::LeftToRight</enum>
        </property>
        <property name="text">
```

```
         <string>Instrumentación Virtual Aplicada</string>
        </property>
        <property name="textFormat">
         <enum>Qt::PlainText</enum>
        </property>
        <property name="alignment">
         <set>Qt::AlignCenter</set>
        </property>
       </widget>
      </item>
      <item>
       <layout class="QHBoxLayout" name="horizontalLayout_3">
        <item>
         <layout class="QVBoxLayout" name="verticalLayout_10">
          <item>
           <layout class="QVBoxLayout" name="verticalLayout_6">
            <item>
             <layout class="QHBoxLayout" name="horizontalLayout_2">
              <item>
               <layout class="QVBoxLayout" name="verticalLayout_5">
                <item>
                 <widget class="QLabel" name="label_4">
                  <property name="text">
                   <string>Umbral de energía [W/h]</string>
                  </property>
                 </widget>
                </item>
               </layout>
              </item>
              <item>
               <layout class="QVBoxLayout" name="verticalLayout_4">
                <item>
                 <widget class="QDoubleSpinBox"
name="doubleSpinBox_umbral"/>
                </item>
               </layout>
              </item>
             </layout>
            </item>
            <item>
             <widget class="QPushButton" name="start">
              <property name="text">
               <string>Iniciar</string>
              </property>
             </widget>
```

```
          </item>
          <item>
           <layout class="QHBoxLayout" name="horizontalLayout">
            <item>
             <layout class="QVBoxLayout" name="verticalLayout_3">
              <item>
               <widget class="QLabel" name="label_3">
                <property name="text">
                 <string>Voltaje</string>
                </property>
               </widget>
              </item>
              <item>
               <widget class="QLabel" name="label_5">
                <property name="text">
                 <string>Corriente</string>
                </property>
               </widget>
              </item>
              <item>
               <widget class="QLabel" name="label_6">
                <property name="text">
                 <string>Potencia</string>
                </property>
               </widget>
              </item>
             </layout>
            </item>
            <item>
             <layout class="QVBoxLayout" name="verticalLayout_2">
              <item>
               <widget class="QLCDNumber" name="lcd_voltaje"/>
              </item>
              <item>
               <widget class="QLCDNumber" name="lcd_current"/>
              </item>
              <item>
               <widget class="QLCDNumber" name="lcd_power"/>
              </item>
             </layout>
            </item>
           </layout>
          </item>
         </layout>
        </item>
```

```
          <item>
           <widget class="QLabel" name="label_7">
            <property name="font">
             <font>
              <pointsize>11</pointsize>
             </font>
            </property>
            <property name="text">
             <string>Luis Escárcega Corona</string>
            </property>
            <property name="alignment">
             <set>Qt::AlignCenter</set>
            </property>
           </widget>
          </item>
         </layout>
        </item>
        <item>
         <layout class="QGridLayout" name="gridLayout">
          <item row="0" column="0">
           <widget class="QFrame" name="frame">
            <property name="styleSheet">
             <string notr="true">background-color:rgb(240, 240, 240);
border-radius:10px;
</string>
            </property>
            <property name="frameShape">
             <enum>QFrame::StyledPanel</enum>
            </property>
            <property name="frameShadow">
             <enum>QFrame::Raised</enum>
            </property>
            <layout class="QVBoxLayout" name="verticalLayout" stretch="1,5">
             <property name="leftMargin">
              <number>10</number>
             </property>
             <item>
              <widget class="QLabel" name="label_8">
               <property name="text">
                <string>Voltaje</string>
               </property>
               <property name="alignment">
                <set>Qt::AlignCenter</set>
               </property>
              </widget>
```

```
             </item>
             <item>
              <layout class="QVBoxLayout" name="grafica_voltaje"/>
             </item>
            </layout>
           </widget>
          </item>
          <item row="0" column="1">
           <widget class="QFrame" name="frame_2">
            <property name="styleSheet">
             <string notr="true">background-color:rgb(240, 240, 240);
border-radius:10px;
</string>
            </property>
            <property name="frameShape">
             <enum>QFrame::StyledPanel</enum>
            </property>
            <property name="frameShadow">
             <enum>QFrame::Raised</enum>
            </property>
            <layout class="QVBoxLayout" name="verticalLayout_7"
stretch="1,5">
             <item>
              <widget class="QLabel" name="label_12">
               <property name="text">
                <string>Corriente</string>
               </property>
               <property name="alignment">
                <set>Qt::AlignCenter</set>
               </property>
              </widget>
             </item>
             <item>
              <layout class="QVBoxLayout" name="grafica_corriente"/>
             </item>
            </layout>
           </widget>
          </item>
          <item row="1" column="0">
           <widget class="QFrame" name="frame_3">
            <property name="styleSheet">
             <string notr="true">background-color:rgb(240, 240, 240);
border-radius:10px;

</string>
```

```
            </property>
            <property name="frameShape">
             <enum>QFrame::StyledPanel</enum>
            </property>
            <property name="frameShadow">
             <enum>QFrame::Raised</enum>
            </property>
            <layout class="QVBoxLayout" name="verticalLayout_8"
stretch="1,5">
             <item>
              <widget class="QLabel" name="label_9">
               <property name="text">
                <string>Potencia</string>
               </property>
               <property name="alignment">
                <set>Qt::AlignCenter</set>
               </property>
              </widget>
             </item>
             <item>
              <layout class="QVBoxLayout" name="grafica_potencia"/>
             </item>
            </layout>
           </widget>
          </item>
          <item row="1" column="1">
           <widget class="QFrame" name="frame_4">
            <property name="styleSheet">
             <string notr="true">background-color:rgb(240, 240, 240);
border-radius:10px;
</string>
            </property>
            <property name="frameShape">
             <enum>QFrame::StyledPanel</enum>
            </property>
            <property name="frameShadow">
             <enum>QFrame::Raised</enum>
            </property>
            <layout class="QVBoxLayout" name="verticalLayout_9"
stretch="1,5">
             <item>
              <widget class="QLabel" name="label_10">
               <property name="text">
                <string>Energía</string>
               </property>
```

```
              <property name="alignment">
               <set>Qt::AlignCenter</set>
              </property>
             </widget>
            </item>
            <item>
             <layout class="QVBoxLayout" name="grafica_energia"/>
            </item>
           </layout>
          </widget>
         </item>
        </layout>
       </item>
      </layout>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

Anexo 3. Backend.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'Backend.ui'
#
# Created by: PyQt5 UI code generator 5.15.7
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.setWindowModality(QtCore.Qt.WindowModal)
        MainWindow.resize(723, 443)
        MainWindow.setStyleSheet("background-color: rgb(222, 222, 222);")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout_2 = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout_2.setObjectName("gridLayout_2")
        self.verticalLayout_11 = QtWidgets.QVBoxLayout()
        self.verticalLayout_11.setObjectName("verticalLayout_11")
        self.label = QtWidgets.QLabel(self.centralwidget)
        font = QtGui.QFont()
        font.setPointSize(13)
        self.label.setFont(font)
        self.label.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label.setTextFormat(QtCore.Qt.PlainText)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName("label")
        self.verticalLayout_11.addWidget(self.label)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        self.verticalLayout_10 = QtWidgets.QVBoxLayout()
        self.verticalLayout_10.setObjectName("verticalLayout_10")
        self.verticalLayout_6 = QtWidgets.QVBoxLayout()
        self.verticalLayout_6.setObjectName("verticalLayout_6")
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.verticalLayout_5 = QtWidgets.QVBoxLayout()
```

```
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setObjectName("label_4")
        self.verticalLayout_5.addWidget(self.label_4)
        self.horizontalLayout_2.addLayout(self.verticalLayout_5)
        self.verticalLayout_4 = QtWidgets.QVBoxLayout()
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.doubleSpinBox_umbral =
QtWidgets.QDoubleSpinBox(self.centralwidget)
        self.doubleSpinBox_umbral.setObjectName("doubleSpinBox_umbral")
        self.verticalLayout_4.addWidget(self.doubleSpinBox_umbral)
        self.horizontalLayout_2.addLayout(self.verticalLayout_4)
        self.verticalLayout_6.addLayout(self.horizontalLayout_2)
        self.start = QtWidgets.QPushButton(self.centralwidget)
        self.start.setObjectName("start")
        self.verticalLayout_6.addWidget(self.start)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout()
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setObjectName("label_3")
        self.verticalLayout_3.addWidget(self.label_3)
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setObjectName("label_5")
        self.verticalLayout_3.addWidget(self.label_5)
        self.label_6 = QtWidgets.QLabel(self.centralwidget)
        self.label_6.setObjectName("label_6")
        self.verticalLayout_3.addWidget(self.label_6)
        self.horizontalLayout.addLayout(self.verticalLayout_3)
        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.lcd_voltaje = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcd_voltaje.setObjectName("lcd_voltaje")
        self.verticalLayout_2.addWidget(self.lcd_voltaje)
        self.lcd_current = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcd_current.setObjectName("lcd_current")
        self.verticalLayout_2.addWidget(self.lcd_current)
        self.lcd_power = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcd_power.setObjectName("lcd_power")
        self.verticalLayout_2.addWidget(self.lcd_power)
        self.horizontalLayout.addLayout(self.verticalLayout_2)
        self.verticalLayout_6.addLayout(self.horizontalLayout)
        self.verticalLayout_10.addLayout(self.verticalLayout_6)
        self.label_7 = QtWidgets.QLabel(self.centralwidget)
```

```
        font = QtGui.QFont()
        font.setPointSize(11)
        self.label_7.setFont(font)
        self.label_7.setAlignment(QtCore.Qt.AlignCenter)
        self.label_7.setObjectName("label_7")
        self.verticalLayout_10.addWidget(self.label_7)
        self.horizontalLayout_3.addLayout(self.verticalLayout_10)
        self.gridLayout = QtWidgets.QGridLayout()
        self.gridLayout.setObjectName("gridLayout")
        self.frame = QtWidgets.QFrame(self.centralwidget)
        self.frame.setStyleSheet("background-color:rgb(240, 240, 240);\n"
"border-radius:10px;\n"
"")
        self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame.setObjectName("frame")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.frame)
        self.verticalLayout.setContentsMargins(10, -1, -1, -1)
        self.verticalLayout.setObjectName("verticalLayout")
        self.label_8 = QtWidgets.QLabel(self.frame)
        self.label_8.setAlignment(QtCore.Qt.AlignCenter)
        self.label_8.setObjectName("label_8")
        self.verticalLayout.addWidget(self.label_8)
        self.grafica_voltaje = QtWidgets.QVBoxLayout()
        self.grafica_voltaje.setObjectName("grafica_voltaje")
        self.verticalLayout.addLayout(self.grafica_voltaje)
        self.verticalLayout.setStretch(0, 1)
        self.verticalLayout.setStretch(1, 5)
        self.gridLayout.addWidget(self.frame, 0, 0, 1, 1)
        self.frame_2 = QtWidgets.QFrame(self.centralwidget)
        self.frame_2.setStyleSheet("background-color:rgb(240, 240, 240);\n"
"border-radius:10px;\n"
"")
        self.frame_2.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame_2.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame_2.setObjectName("frame_2")
        self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.frame_2)
        self.verticalLayout_7.setObjectName("verticalLayout_7")
        self.label_12 = QtWidgets.QLabel(self.frame_2)
        self.label_12.setAlignment(QtCore.Qt.AlignCenter)
        self.label_12.setObjectName("label_12")
        self.verticalLayout_7.addWidget(self.label_12)
        self.grafica_corriente = QtWidgets.QVBoxLayout()
        self.grafica_corriente.setObjectName("grafica_corriente")
        self.verticalLayout_7.addLayout(self.grafica_corriente)
```

```
        self.verticalLayout_7.setStretch(0, 1)
        self.verticalLayout_7.setStretch(1, 5)
        self.gridLayout.addWidget(self.frame_2, 0, 1, 1, 1)
        self.frame_3 = QtWidgets.QFrame(self.centralwidget)
        self.frame_3.setStyleSheet("background-color:rgb(240, 240, 240);\n"
"border-radius:10px;\n"
"\n"
"")
        self.frame_3.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame_3.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame_3.setObjectName("frame_3")
        self.verticalLayout_8 = QtWidgets.QVBoxLayout(self.frame_3)
        self.verticalLayout_8.setObjectName("verticalLayout_8")
        self.label_9 = QtWidgets.QLabel(self.frame_3)
        self.label_9.setAlignment(QtCore.Qt.AlignCenter)
        self.label_9.setObjectName("label_9")
        self.verticalLayout_8.addWidget(self.label_9)
        self.grafica_potencia = QtWidgets.QVBoxLayout()
        self.grafica_potencia.setObjectName("grafica_potencia")
        self.verticalLayout_8.addLayout(self.grafica_potencia)
        self.verticalLayout_8.setStretch(0, 1)
        self.verticalLayout_8.setStretch(1, 5)
        self.gridLayout.addWidget(self.frame_3, 1, 0, 1, 1)
        self.frame_4 = QtWidgets.QFrame(self.centralwidget)
        self.frame_4.setStyleSheet("background-color:rgb(240, 240, 240);\n"
"border-radius:10px;\n"
"")
        self.frame_4.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame_4.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame_4.setObjectName("frame_4")
        self.verticalLayout_9 = QtWidgets.QVBoxLayout(self.frame_4)
        self.verticalLayout_9.setObjectName("verticalLayout_9")
        self.label_10 = QtWidgets.QLabel(self.frame_4)
        self.label_10.setAlignment(QtCore.Qt.AlignCenter)
        self.label_10.setObjectName("label_10")
        self.verticalLayout_9.addWidget(self.label_10)
        self.grafica_energia = QtWidgets.QVBoxLayout()
        self.grafica_energia.setObjectName("grafica_energia")
        self.verticalLayout_9.addLayout(self.grafica_energia)
        self.verticalLayout_9.setStretch(0, 1)
        self.verticalLayout_9.setStretch(1, 5)
        self.gridLayout.addWidget(self.frame_4, 1, 1, 1, 1)
        self.horizontalLayout_3.addLayout(self.gridLayout)
        self.verticalLayout_11.addLayout(self.horizontalLayout_3)
        self.gridLayout_2.addLayout(self.verticalLayout_11, 0, 0, 1, 1)
```

```
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Proyecto Final -
Medidor de Energía"))
        self.label.setAccessibleName(_translate("MainWindow", "0"))
        self.label.setText(_translate("MainWindow", "Instrumentación Virtual
Aplicada"))
        self.label_4.setText(_translate("MainWindow", "Umbral de energía
[W/h]"))
        self.start.setText(_translate("MainWindow", "Iniciar"))
        self.label_3.setText(_translate("MainWindow", "Voltaje"))
        self.label_5.setText(_translate("MainWindow", "Corriente"))
        self.label_6.setText(_translate("MainWindow", "Potencia"))
        self.label_7.setText(_translate("MainWindow", "Luis Escárcega
Corona"))
        self.label_8.setText(_translate("MainWindow", "Voltaje"))
        self.label_12.setText(_translate("MainWindow", "Corriente"))
        self.label_9.setText(_translate("MainWindow", "Potencia"))
        self.label_10.setText(_translate("MainWindow", "Energía"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Anexo 4. Frontend.py

```
import sys
import numpy as np
from Backend import*
from PyQt5 import QtCore, QtGui, QtWidgets, uic
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
import matplotlib.pyplot as plt
# Conexión con Arduino
import pyfirmata
import time
board = pyfirmata.Arduino('COM3')
iter8 = pyfirmata.util.Iterator(board)
iter8.start()
led1 = board.get_pin('d:7:o')
sensvoltage = board.get_pin('a:0:i')
senscurrent = board.get_pin('a:2:i')
print("CONEXION CON ARDUINO")

class MiApp(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        print("Fin de MiApp")
        self.ui.start.clicked.connect(self.inicio)

    def inicio(self):
        print("Inicio")
        umbral = self.ui.doubleSpinBox_umbral.value()
        np.t = []
        # 1 Obtener Voltaje
        self.yv = self.getvoltage(self, sensvoltage, 0, 1023, 0.0, 25.0)
        self.yvpromedio = sum(self.getvoltage)/len(self.getvoltage)
        print("Voltaje obtenido")
        # 2 Obtener Corriente
        self.yc = self.getcurrent(self, senscurrent)
        self.ycpromedio = sum(self.yc)/len(self.yc)
        print("Corriente obtenido")
        # 3 Obtener Potencia
        self.potencia = np.multiply(self.yc,self.yv)
        self.potenciapromedio = sum(self.potencia)/len(self.potencia)
        print("Potencia obtenida")
        # 4 Desplegar en LCD
```

```
        self.ui.lcd_voltaje.display(self.yvpromedio)
        self.ui.lcd_current.display(self.ycpromedio)
        self.ui.lcd_power.display(self.potenciapromedio)
        print("Valores en LCD")
        # 5 Obtener Energía yp*np.t
        self.ye = self.yp*np.t
        print("Energía obtenida")
        # 6 Graficar
        self.grafica1 = Canvas_voltage()
        self.grafica2 = Canvas_corriente()
        self.grafica3 = Canvas_potencia()
        self.grafica4 = Canvas_energia()
        while self.ye < umbral:
            self.ui.grafica_voltaje.addWidget(self.grafica1(self, self.yv))
            self.ui.grafica_corriente.addWidget(self.grafica2(self,
self.yc))
            self.ui.grafica_potencia.addWidget(self.grafica3(self, self.yp))
            self.ui.grafica_energia.addWidget(self.grafica4(self, self.ye))
            if (self.ye < umbral):
                led1.write(1)
                time.sleep(5)
                break
        print("Gráficas obtenidas")
    def getvoltage(self, x, in_min, in_max, out_min, out_max):     # Funcion
para obtener el voltaje
        return (x - in_min)*(out_max - out_min) / (in_max - in_min) +
out_min

    def getcurrent(self, cur):                                     # Funcion
para obtener la corriente
        for i in range (200):
            self.corriente = (cur*(5.0/1023)-2.5)/0.066
            self.corriente = self.corriente/200
        return self.corriente

class Canvas_voltage(FigureCanvas):  # Clase para graficar voltaje
    def __init__(self, x=[]):
        self.fig, self.ax = plt.subplots(
            1, dpi=80, figsize=(5, 5), sharey=True, facecolor='white')
        super().__init__(self.fig)
        yv = x
        self.ax = plt.axes()
        plt.plot(yv, color='orange')

class Canvas_corriente(FigureCanvas):  # Clase para graficar corriente
```

```
    def __init__(self, x=[]):
        self.fig, self.ax = plt.subplots(
            1, dpi=80, figsize=(5, 5), sharey=True, facecolor='white')
        super().__init__(self.fig)
        yc = x
        self.ax = plt.axes()
        plt.plot(yc, color='blue')

class Canvas_potencia(FigureCanvas):  # Clase para graficar potencia
    def __init__(self, x=[]):
        self.fig, self.ax = plt.subplots(
            1, dpi=80, figsize=(5, 5), sharey=True, facecolor='white')
        super().__init__(self.fig)
        yp = x
        self.ax = plt.axes()
        plt.plot(yp, color='green')

class Canvas_energia(FigureCanvas):  # Clase para graficar energia
    def __init__(self, x=[]):
        self.fig, self.ax = plt.subplots(
            1, dpi=80, figsize=(5, 5), sharey=True, facecolor='white')
        super().__init__(self.fig)
        ye = x
        self.ax = plt.axes()
        plt.plot(ye, color='magenta')

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mi_app = MiApp()
    mi_app.show()
    sys.exit(app.exec_())
```