Embedded Systems Semester 5 - Sensor Technology

Lecturer Peter Kamphuis

Solar Irradiance Meter

—

Stoyan Bonev, Dimitriy Georgiev, Richard Kaiva

Group 1

Student № 356626, 381912, 388499

Date 13-01-2021

Table of Contents

Analysis

For this assignment we were asked to create a solar irradiance meter by means of a small 5W PV panel and an embedded device based on the PIC18F25k80 microcontroller. The purpose of the meter is to test the efficiency of bigger solar panels, by comparing their power output to the meter's. The meter works by measuring the power of the sunlight at a given point in time and applies temperature correction for the final power calculation. From that value we can estimate the power that the main panels should be outputting, and if they aren't then there might be a problem with them.

#	Requirement
1	Solar panel is connected and produces voltage
2	Power is calculated from panel voltage
3	Output value of the power is corrected for the temperature of the panel using the temperature coefficient.
4	Final power value is given.
5	All data is displayed in an interface

Design

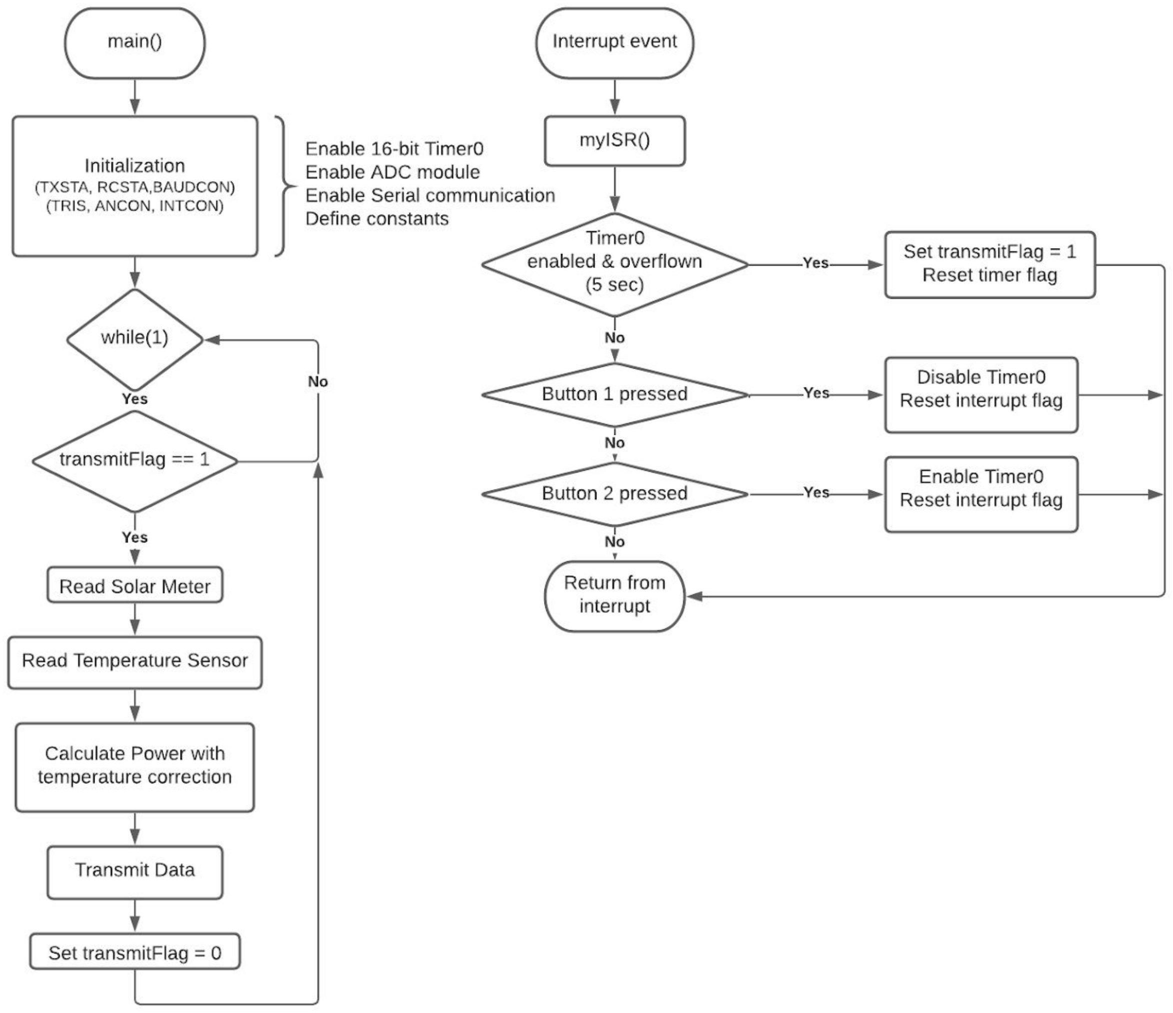


Figure 1. Flowchart of Solara irradiance meter

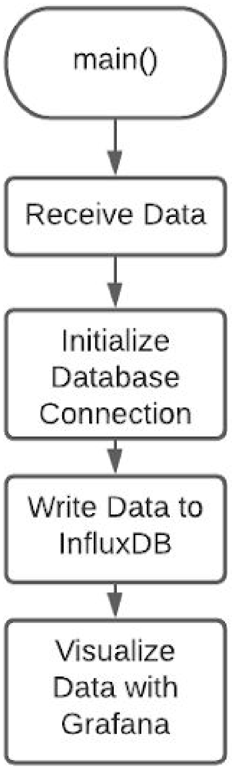


Figure 2. Flowchart of Server-side program

Data Acquisition

After linking all the necessary libraries, the program begins by calling the main() function. In the initialization phase, configuration of the TRIS, PIE, TXSTA, RCSTA, BAUDCON and SPBRG registers is made to enable serial communication of the MCU. Furthermore, configuration of the ADCON and ANCON registers is made to enable the ADC module that will allow conversion of an analog signal to 12-bit digital number. The ANCON register is used to enable the analog mode of the necessary pins. The ADCON0 register controls the operation of the analog-to-digital module. The ADCON1 register sets the voltage reference and special trigger selection for the module. While the ADCON2 register defines the clock source, data acquisition time, and bit justification for the ADC. Additionally, Timer0 is configured as a 16-bit timer with a prescaler of 1:256 and the INTCON registers are configured to trigger an interrupt event on timer overflow. The timer is set to overflow every 5 seconds. When an interrupt occurs, a designated flag variable is set to indicate that data should be read and transmitted. If the transmit flag is set, then, in the main loop of the program, a function call to the ADC is made and data from AN1 and AN9 channels is read and sent out. The buttons can be used to enable/disable Timer0.

The assignment implies reading the output signals of the solar panel and the temperature sensor attached to it. To be able to measure the output signal of the solar panel, a shunt resistor needs to be connected between the two terminals of the panel. By measuring the voltage drop across the resistor we can calculate the current and hence the output power of the panel. The ADC module of the microcontroller has a 12-bit data register (ADC values: 0-4096) and it was configured to use an internal reference of 4.1V. The ADC resolution can be calculated with the following formula:

$$ADC\ resolution = \frac{V_{ref}}{2^{12}} = \frac{4.1}{4096} = 1\ mA$$

The voltage drop across a 1 ohm resistor would not be high enough for such ADC resolution. That is why, a shunt resistor of >100 ohms is needed. The resistance, however, should not be too big either, because then the voltage drop may become bigger than what the ADC can handle and damage it. The output power of the panel can be calculated based on the following formula:

$$P = V^2/R,$$

where the voltage is measured across the shunt resistor. Furthermore, the power should be corrected for the temperature of the PV cells. The temperature has influence on the efficiency of the cells with a coefficient of 0.4%/°C, with 25°C as a reference. The final power can be calculated according to the following formula:

$$P_{corrected} = P - [P * (Temperature - 25.0) * 0.4\%]$$

Finally, the solar irradiance can be estimated by dividing the temperature corrected power by the area of the solar panel, according to the following formula:

$$Solar\ Irradiance\ (W/m^2) = \frac{P_{corrected}}{Area\ of\ solar\ panel}$$

Data Storage and Visualization:

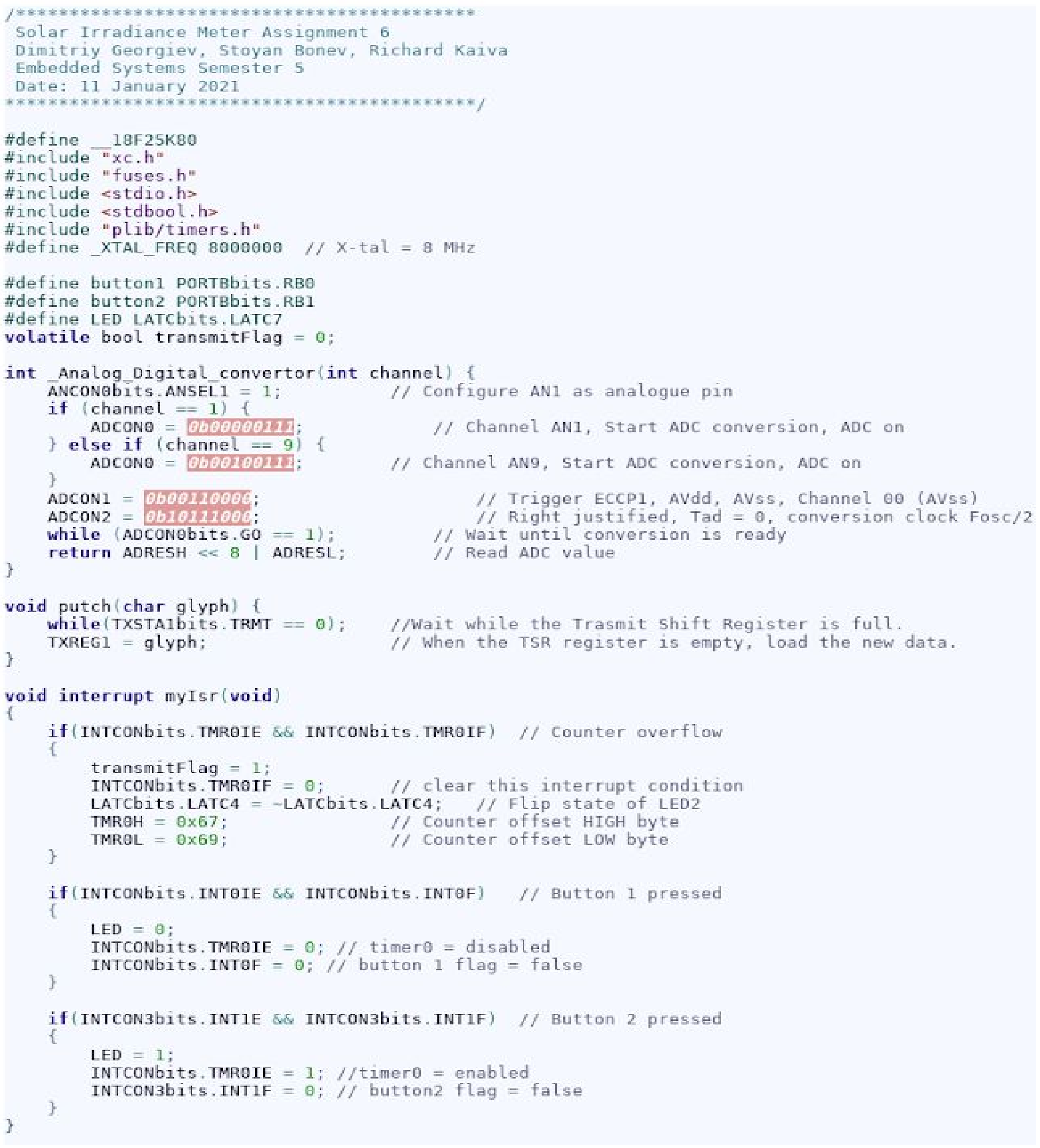
Storing the data is managed by InfluxDB which is a high-performance database for time series data. The user interface is then created using Grafana - an open source tool for data visualization. Grafana is set up to query the database which is running locally on the computer. Data of four different parameters is being stored in addition to the time stamp: sun power from Buienradar, weather description in Groningen region, measured sun power, and the temperature of the solar panel. Finally, a Python code has been created to read the data sent by the PIC18F66k80 microcontroller from the serial port, to request weather data from Buienradar, and to save the information to the database.

At first the Python code creates a new instance of the InfluxDBClient to connect to the database at localhost on port 8086. The client is then set to use the database called "solar_irradiance_db". The correct port ("COM4") is then opened with a specified baudrate of 2400. Every time the microcontroller sends the data, it is read from the serial port and saved to a variable for further processing. The temperature and solar irradiance values are sent together separated by a comma; therefore, it is necessary to split the received data to obtain two separate values.

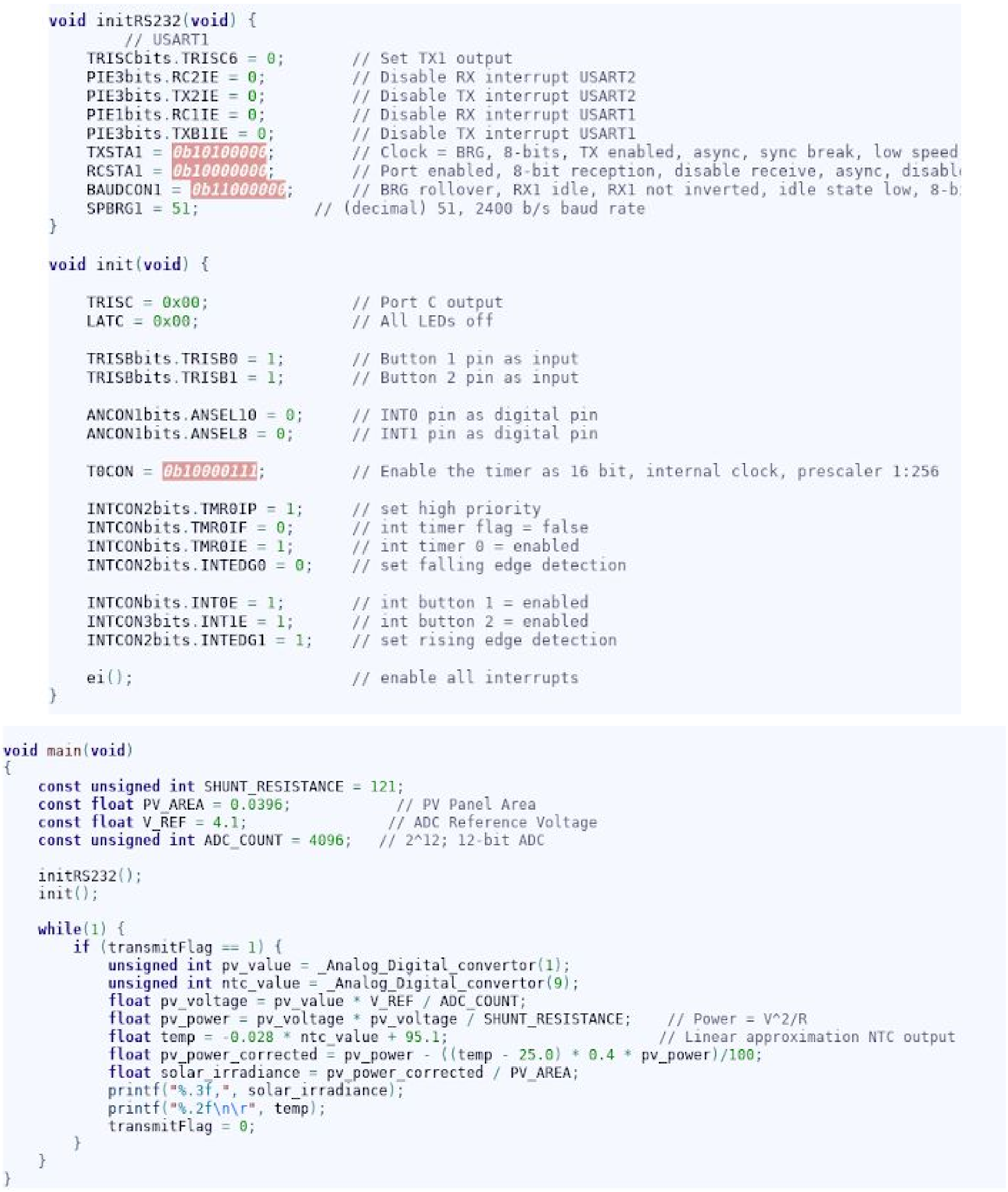
Subsequently, a request is made to data.buienradar.nl to receive the weather data in *json* format which is then filtered to acquire the sun power and weather information from the weather station in Groningen (closest available one to our location). These four parameters are converted into a format that allows to save the data to the database; information about the timestamp is added automatically by the system. The information processing, requesting, and saving procedure is repeated every time new data is received from the microcontroller. Grafana is then set up to request data from the database and create graphs for the solar panel temperature, measured solar irradiance, and solar power from Buienradar separately. The weather information, a short sentence describing the weather in a region, is simply displayed as text.

Implementation

Microcontroller Side



```c
/**********************************************
 Solar Irradiance Meter Assignment 6
 Dimitriy Georgiev, Stoyan Bonev, Richard Kaiva
 Embedded Systems Semester 5
 Date: 11 January 2021
**********************************************/

#define __18F25K80
#include "xc.h"
#include "fuses.h"
#include <stdio.h>
#include <stdbool.h>
#include "plib/timers.h"
#define _XTAL_FREQ 8000000  // X-tal = 8 MHz

#define button1 PORTBbits.RB0
#define button2 PORTBbits.RB1
#define LED LATCbits.LATC7
volatile bool transmitFlag = 0;

int _Analog_Digital_convertor(int channel) {
    ANCON0bits.ANSEL1 = 1;          // Configure AN1 as analogue pin
    if (channel == 1) {
        ADCON0 = 0b00000111;            // Channel AN1, Start ADC conversion, ADC on
    } else if (channel == 9) {
        ADCON0 = 0b00100111;        // Channel AN9, Start ADC conversion, ADC on
    }
    ADCON1 = 0b00110000;                // Trigger ECCP1, AVdd, AVss, Channel 00 (AVss)
    ADCON2 = 0b10111000;                // Right justified, Tad = 0, conversion clock Fosc/2
    while (ADCON0bits.GO == 1);         // Wait until conversion is ready
    return ADRESH << 8 | ADRESL;        // Read ADC value
}

void putch(char glyph) {
    while(TXSTA1bits.TRMT == 0);    //Wait while the Trasmit Shift Register is full.
    TXREG1 = glyph;                 // When the TSR register is empty, load the new data.
}

void interrupt myIsr(void)
{
    if(INTCONbits.TMR0IE && INTCONbits.TMR0IF)  // Counter overflow
    {
        transmitFlag = 1;
        INTCONbits.TMR0IF = 0;      // clear this interrupt condition
        LATCbits.LATC4 = ~LATCbits.LATC4;   // Flip state of LED2
        TMR0H = 0x67;               // Counter offset HIGH byte
        TMR0L = 0x69;               // Counter offset LOW byte
    }

    if(INTCONbits.INT0IE && INTCONbits.INT0F)   // Button 1 pressed
    {
        LED = 0;
        INTCONbits.TMR0IE = 0; // timer0 = disabled
        INTCONbits.INT0F = 0; // button 1 flag = false
    }

    if(INTCON3bits.INT1E && INTCON3bits.INT1F)  // Button 2 pressed
    {
        LED = 1;
        INTCONbits.TMR0IE = 1; //timer0 = enabled
        INTCON3bits.INT1F = 0; // button2 flag = false
    }
}
```

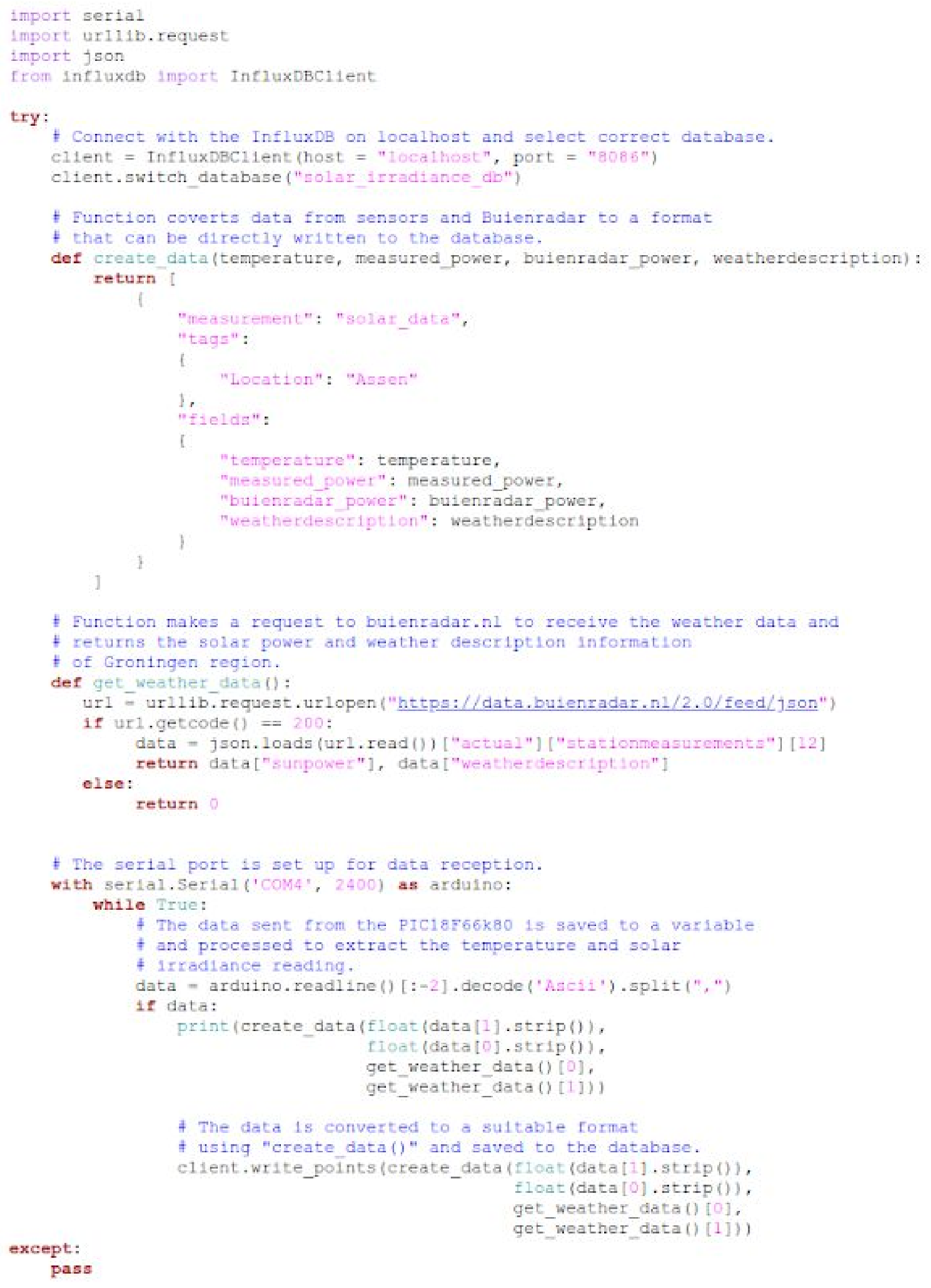


```c
void initRS232(void) {
        // USART1
    TRISCbits.TRISC6 = 0;         // Set TX1 output
    PIE3bits.RC2IE = 0;           // Disable RX interrupt USART2
    PIE3bits.TX2IE = 0;           // Disable TX interrupt USART2
    PIE1bits.RC1IE = 0;           // Disable RX interrupt USART1
    PIE3bits.TXB1IE = 0;          // Disable TX interrupt USART1
    TXSTA1 = 0b10100000;          // Clock = BRG, 8-bits, TX enabled, async, sync break, low speed
    RCSTA1 = 0b10000000;          // Port enabled, 8-bit reception, disable receive, async, disabl
    BAUDCON1 = 0b11000000;        // BRG rollover, RX1 idle, RX1 not inverted, idle state low, 8-b
    SPBRG1 = 51;              // (decimal) 51, 2400 b/s baud rate
}

void init(void) {

    TRISC = 0x00;                 // Port C output
    LATC = 0x00;                  // All LEDs off

    TRISBbits.TRISB0 = 1;         // Button 1 pin as input
    TRISBbits.TRISB1 = 1;         // Button 2 pin as input

    ANCON1bits.ANSEL10 = 0;       // INT0 pin as digital pin
    ANCON1bits.ANSEL8 = 0;        // INT1 pin as digital pin

    T0CON = 0b10000111;           // Enable the timer as 16 bit, internal clock, prescaler 1:256

    INTCON2bits.TMR0IP = 1;       // set high priority
    INTCONbits.TMR0IF = 0;        // int timer flag = false
    INTCONbits.TMR0IE = 1;        // int timer 0 = enabled
    INTCON2bits.INTEDG0 = 0;      // set falling edge detection

    INTCONbits.INT0E = 1;         // int button 1 = enabled
    INTCON3bits.INT1E = 1;        // int button 2 = enabled
    INTCON2bits.INTEDG1 = 1;      // set rising edge detection

    ei();                         // enable all interrupts
}
```

```c
void main(void)
{
    const unsigned int SHUNT_RESISTANCE = 121;
    const float PV_AREA = 0.0396;            // PV Panel Area
    const float V_REF = 4.1;                // ADC Reference Voltage
    const unsigned int ADC_COUNT = 4096;   // 2^12; 12-bit ADC

    initRS232();
    init();

    while(1) {
        if (transmitFlag == 1) {
            unsigned int pv_value = _Analog_Digital_convertor(1);
            unsigned int ntc_value = _Analog_Digital_convertor(9);
            float pv_voltage = pv_value * V_REF / ADC_COUNT;
            float pv_power = pv_voltage * pv_voltage / SHUNT_RESISTANCE;    // Power = V^2/R
            float temp = -0.028 * ntc_value + 95.1;                        // Linear approximation NTC output
            float pv_power_corrected = pv_power - ((temp - 25.0) * 0.4 * pv_power)/100;
            float solar_irradiance = pv_power_corrected / PV_AREA;
            printf("%.3f,", solar_irradiance);
            printf("%.2f\n\r", temp);
            transmitFlag = 0;
        }
    }
}
```

Figure 3. Microcontroller C code - Reading and transmission of PV panel data

Server Side



```python
import serial
import urllib.request
import json
from influxdb import InfluxDBClient

try:
    # Connect with the InfluxDB on localhost and select correct database.
    client = InfluxDBClient(host = "localhost", port = "8086")
    client.switch_database("solar_irradiance_db")

    # Function coverts data from sensors and Buienradar to a format
    # that can be directly written to the database.
    def create_data(temperature, measured_power, buienradar_power, weatherdescription):
        return [
            {
                "measurement": "solar_data",
                "tags":
                {
                    "Location": "Assen"
                },
                "fields":
                {
                    "temperature": temperature,
                    "measured_power": measured_power,
                    "buienradar_power": buienradar_power,
                    "weatherdescription": weatherdescription
                }
            }
        ]

    # Function makes a request to buienradar.nl to receive the weather data and
    # returns the solar power and weather description information
    # of Groningen region.
    def get_weather_data():
       url = urllib.request.urlopen("https://data.buienradar.nl/2.0/feed/json")
       if url.getcode() == 200:
            data = json.loads(url.read())["actual"]["stationmeasurements"][12]
            return data["sunpower"], data["weatherdescription"]
       else:
            return 0


    # The serial port is set up for data reception.
    with serial.Serial('COM4', 2400) as arduino:
        while True:
            # The data sent from the PIC18F66k80 is saved to a variable
            # and processed to extract the temperature and solar
            # irradiance reading.
            data = arduino.readline()[:-2].decode('Ascii').split(",")
            if data:
                print(create_data(float(data[1].strip()),
                                  float(data[0].strip()),
                                  get_weather_data()[0],
                                  get_weather_data()[1]))

                # The data is converted to a suitable format
                # using "create_data()" and saved to the database.
                client.write_points(create_data(float(data[1].strip()),
                                                float(data[0].strip()),
                                                get_weather_data()[0],
                                                get_weather_data()[1]))
except:
    pass
```

Figure 4. Server Python code - Saving data to the database

Evaluation

#	Requirement	Check
1	Solar panel is connected and produces voltage	Yes
2	Power is calculated from panel voltage	Yes
3	Output value of the power is corrected for the temperature of the panel using the temperature coefficient.	Yes
4	Final power value is given.	Yes
5	All data is displayed in an interface	Yes