Low Power Sensor Station Datalogger

Rui Carapinha
Witold Paluszynski
Wroclaw Univesity of Technology
Politechnika Wroclawska

22/01/2019

1 Introduction

My project proposal is a Low Power Sensor Station Datalogger, this project is based on an ATmega328 processor and the goal is to last the most time possible with a battery.

The program collects data every 10 minutes from two thermistor and from a real-time clock and saves it to a SD Card. The Arduino is sleeping the most part of the time, it only wakes up to collect the data then go backs to sleep, this way the program can lasts longer with a battery.

2 Hardware and Software

The hardware I used was the following:

- Arduino Uno
- Real Time Clock DS3231
- Micro SD Card Adapter
- 2 Thermistors
- Resistors
- Battery Adapter

The software I used was the following:

- C++
- Matlab
- Fritzing

3 Functionality

3.1 Real Time Clock DS3231

A Real-Time clock is a computer clock that keeps track of the current time. The DS3231 is a Real-Time Clock with high accuracy, this module is powered by its own 3V battery. I choosed this Real-Time Clock because it has the funcionality of alarms that can be used as interrupts. Those interrupts will ”wake up” the Arduino every 10 minutes to save the data.

Figure 1: DS3231 - Real-Time Clock

3.2 Micro SD Card Adapter

This module, called Micro SD Card Adapter allows for the Arduino to access the data in the SD Card. This adapter is the only hardware that can’t work natively with 3.3V. To make it work with 3.3V the only change it needs is to bypass the 5V to 3.3V voltage regulator, if we do that we can work with this module at 3.3V.

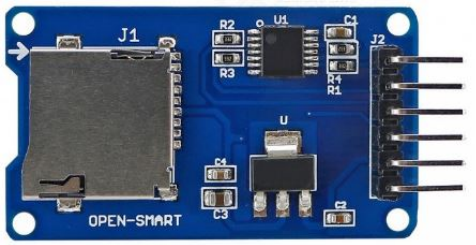

Figure 2: Micro SD Adapter

3.3 Thermistor

The thermistor (thermal + resistor) is a resistance that depends on the temperature. In my case I'm using a NTC Thermistor (Negative Temperature Coefficient) this means that the resistance of the thermistor decreases when the temperature rises.

Figure 3: Thermistor

4 Mounting and Coding

4.1 Mounting

The first working mounting I made has in a breadboard and has like the following:

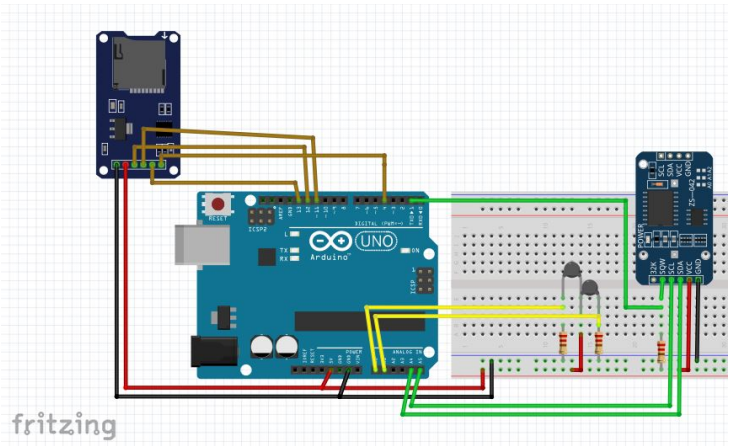

Figure 4: Breadboard Mounting

4.2 Coding

In this part of the report I'll explain, part by part, how the Arduino code works. First, I'll start with the initializations, which are the following:

Listing 1: Initializations

```
//Libraries
#include <Wire.h>
#include <RTClibExtended.h>
#include <LowPower.h>
#include <SPI.h>
#include <SD.h>

//Pins
File dataFile;
RTC_DS3231 rtc;

int SDPin = 4;
int T1Pin = 0;
int T2Pin = 1;
int V1, V2;
float R1 = 10000;
float R2 = 10000;
float logR2_T1, R2_T1, T1, Tc1;
float logR2_T2, R2_T2, T2, Tc2;
float c1 = 1.009249522e-03;
float c2 = 2.378405444e-04;
float c3 = 2.019202697e-07;

//Variables
int NowHour;                    //Interrupt
int NowMinute;                  //Interrupt
String dataString = "";         //SdCard

void setup() {
        SD.begin(4);
        Wire.begin();
        rtc.begin();
        rtc.armAlarm(1, false);
        rtc.clearAlarm(1);
        rtc.alarmInterrupt(1, false);
        rtc.armAlarm(2, false);
        rtc.clearAlarm(2);
        rtc.alarmInterrupt(2, false);
        rtc.writeSqwPinMode(DS3231_OFF);
        Interrupt();
}
```

This code starts all the libraries (Wire, RTCLibExtended, LowPower, SPI and SD) and all the libraries necessary for the program to work. The most important part of this code is the last part, the setup part. This part of the code initalizes the communication with the clock and clears any pending alarms the clock can have.

The next part of the code, is the functions. The first function I'll explain is the function that reads the values from the thermistors and from the real-time clock and saves it to the SD Card.

The function, with the name Datalog, is the following:

Listing 2: Datalog

```
void DataLog() {
    //Reading Thermistor 1
    V1 = analogRead(T1Pin);
    R2_T1 = R1 * (1023.0 / (float)V1 - 1.0);
    logR2_T1 = log(R2_T1);
    T1 = (1/(c1+c2*logR2_T1+c3*logR2_T1*logR2_T1*logR2_T1));
    Tc1 = T1 - 273.15;

    //Reading Thermistor 2
    V2 = analogRead(T2Pin);
    R2_T2 = R2 * (1023.0 / (float)V2 - 1.0);
    logR2_T2 = log(R2_T2);
    T2 = (1/(c1+c2*logR2_T2+c3*logR2_T2*logR2_T2*logR2_T2));
    Tc2 = T2 - 273.15;

    DateTime now = rtc.now();
    dataString += String(now.year(), DEC);
    dataString += ",";
    dataString += String(now.month(), DEC);
    dataString += ",";
    dataString += String(now.day(), DEC);
    dataString += ",";
    dataString += String(now.hour(), DEC);
    dataString += ",";
    dataString += String(now.minute(), DEC);
    dataString += ",";
    dataString += String(Tc1);
    dataString += ",";
    dataString += String(Tc2);

    File dataFile = SD.open("datalog.txt", FILE_WRITE);
    if (dataFile) {
      dataFile.println(dataString);
      dataFile.close();
      dataFile.println();
    }
    dataString = "";
}
```

This function reads the values from the thermistors by using the following formula:

$$R_2 = R_1 \cdot \frac{V_{in}}{V_{out}}$$

It works like a voltage divider between the resistance of the thermistor and a known resistance (10kΩ, in this case). Then we get the temperature in Kelvin using:

$$T[K] = \frac{1}{A + B \cdot ln(R) + C \cdot [ln(R)]^3}$$

Then we just need to make a simple conversion from Kelvin to Celsius.

This function retrieves the data from the thermistors and from the real-time clock and saves it to the SD Card. The data is saved in the following format:

$$Year, Month, Day, Hour, Minute, Temperature1, Temperature2$$

The next function handles the interrupt and the alarms. The code is the following:

Listing 3: Interrupt - Part 1

```
rtc.clearAlarm(1);
rtc.clearAlarm(2);
DateTime now = rtc.now();
NowHour = String(now.hour(), DEC).toInt();
NowMinute = String(now.minute(), DEC).toInt();
if (NowMinute <= 10) {
  rtc.setAlarm(ALM1_MATCH_HOURS, 10, NowHour, 0);
  rtc.alarmInterrupt(1, true);
  rtc.setAlarm(ALM2_MATCH_HOURS, 20, NowHour, 0);
  rtc.alarmInterrupt(2, true);
}
else if (NowMinute > 10 && NowMinute <= 20) {
  rtc.setAlarm(ALM1_MATCH_HOURS, 20, NowHour, 0);
  rtc.alarmInterrupt(1, true);
  rtc.setAlarm(ALM2_MATCH_HOURS, 30, NowHour, 0);
  rtc.alarmInterrupt(2, true);
}
else if (NowMinute > 20 && NowMinute <= 30) {
  rtc.setAlarm(ALM1_MATCH_HOURS, 30, NowHour, 0);
  rtc.alarmInterrupt(1, true);
  rtc.setAlarm(ALM2_MATCH_HOURS, 40, NowHour, 0);
  rtc.alarmInterrupt(2, true);
}
else if (NowMinute > 30 && NowMinute <= 40) {
  rtc.setAlarm(ALM1_MATCH_HOURS, 40, NowHour, 0);
  rtc.alarmInterrupt(1, true);
  rtc.setAlarm(ALM2_MATCH_HOURS, 50, NowHour, 0);
  rtc.alarmInterrupt(2, true);
}
else if (NowMinute > 40 && NowMinute <= 50) {
  if(NowHour < 23){
    rtc.setAlarm(ALM1_MATCH_HOURS, 50, NowHour, 0);
    rtc.alarmInterrupt(1, true);
    rtc.setAlarm(ALM2_MATCH_HOURS, 0, NowHour+1, 0);
    rtc.alarmInterrupt(2, true);
  }
  else if(NowHour >= 23){
    rtc.setAlarm(ALM1_MATCH_HOURS, 50, NowHour, 0);
    rtc.alarmInterrupt(1, true);
    rtc.setAlarm(ALM2_MATCH_HOURS, 0, 0, 0);
    rtc.alarmInterrupt(2, true);
  }
}
```

Listing 4: Interrupt - Part 2

```
  else {
    if(NowHour < 23){
      rtc.setAlarm(ALM1_MATCH_HOURS, 0, NowHour+1, 0);
      rtc.alarmInterrupt(1, true);
      rtc.setAlarm(ALM2_MATCH_HOURS, 10, NowHour+1, 0);
      rtc.alarmInterrupt(2, true);
    }
    else if(NowHour >= 23){
      rtc.setAlarm(ALM1_MATCH_HOURS, 0, 0, 0);
      rtc.alarmInterrupt(1, true);
      rtc.setAlarm(ALM2_MATCH_HOURS, 10, 0, 0);
      rtc.alarmInterrupt(2, true);
    }
  }
}
```

This code clears any pending alarm and implements an alarm every 10 minutes. The final function is the one that executes when an interrupt is triggered, the function is the following:

Listing 5: Time Function

```
void TimeHandler() {}

void TimeFunction(){
  DataLog();
  Interrupt();
}
```

This set of two functions are the ones that handle with the interrupt and call the other two functions: Interrupt and Datalog. The last piece of code that implements every thing is the following:

Listing 6: Time Function

```
void loop() {
  attachInterrupt(digitalPinToInterrupt(2),TimeHandler,LOW);
  LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);
  detachInterrupt(digitalPinToInterrupt(2));
  TimeFunction();
}
```

This code attaches the alarm to the interrupt and sets on the LowPower mode for the Arduino, this will make the Arduino go to ”sleep” and only executes the next lines of code when the interrupt is triggered.

5 Results

The data I collected after I was running the program a few hours was the following:

```
2019,1,2,22,0,15.92,15.23
2019,1,2,22,10,17.00,16.41
2019,1,2,22,20,17.59,16.71
2019,1,2,22,30,16.12,15.72
2019,1,2,22,40,16.12,15.03
2019,1,2,22,50,15.92,15.13
2019,1,2,23,0,16.81,16.31
2019,1,2,23,10,14.73,14.14
2019,1,2,23,20,14.53,13.24
2019,1,2,23,30,13.94,12.94
2019,1,2,23,40,17.99,16.51
2019,1,2,23,50,15.13,14.04
2019,1,3,0,0,15.13,14.04
2019,1,3,0,10,15.13,14.43
2019,1,3,0,20,17.40,15.82
```

Figure 5: Data Collected from the SD Card after running a few hours

After that we can easily analyze the data in the MATLAB script:

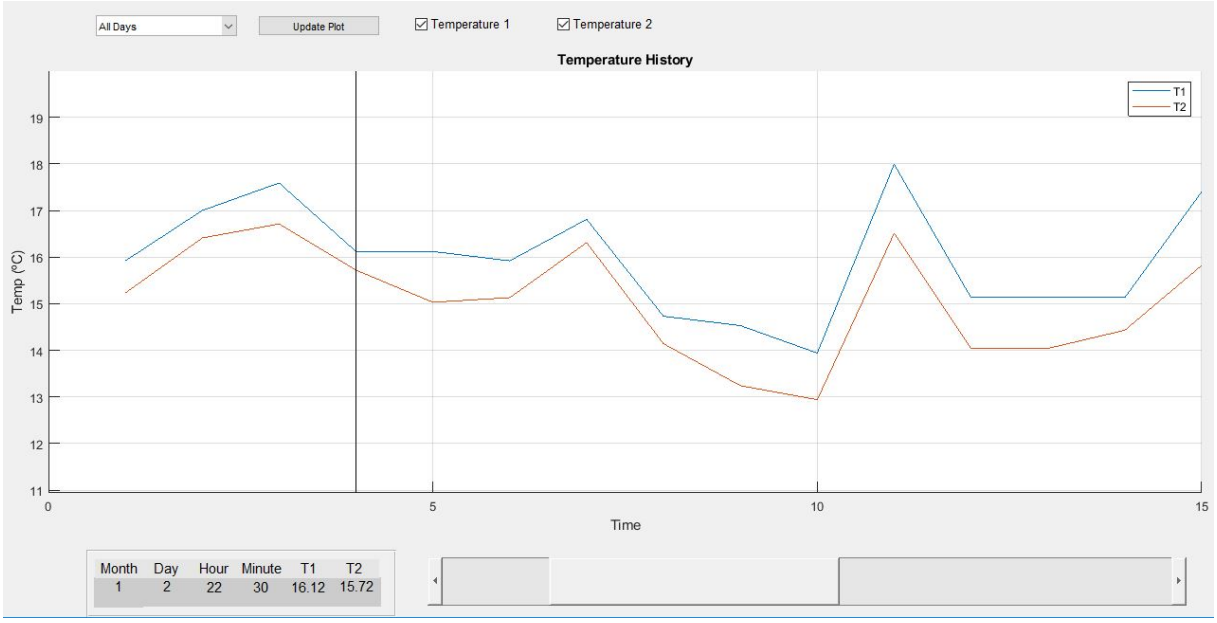

Figure 6: MATLAB analysis of the results

The results I had where pretty staisfactory and we can easily analyze them with the MATLAB program I developed.

6 Conclusion

In this project I was able to work with interrupts and low power modes of the processor, this was very interesting and with has a great experience. I learned that we can extend the batteries life without afecting the processor and the program capabilities. It was also very interesting working with the Real-Time Clock (a really useful module to keep track of the time and to generate alarms) and with the SD Card Adapter so we can move data to wherever we want.

The results I had were very interesting, we could easily get the results from the SD Card and analyze them in the MATLAB.

This project can still be upgraded a lot by, for example, by removing the Arduino and just using the processor because the Arduino has a lot of features (for example, USB connection) that aren't necessary in this project.

7 References

References

[1] Voltage Regulator,
http://www.advanced-monolithic.com/pdf/ds1117.pdf

[2] Real-Time Clock DS3231,
https://www.robotics.org.za/DS3231-MOD

[3] Low Power,
https://www.gammon.com.au/power

[4] Thermistors,
https://www.bcmsensor.com/ntc-thermistors-from-bcm-sensor/

[5] Low Power Library,
https://github.com/rocketscream/Low-Power