DEVELOPMENT OF DS18B20 TEMPERATURE SENSOR IOT DEVICE USING SECURE API CONNECTION

Noje Dan [1], Căraban Alina[2] , Moldovan Ovidiu Gheorghe[3] and Moldovan Octavian Alin[4] , Dan Crăciun[5]
[1] University of Oradea, Doctoral School of Engineering Sciences, dan.noje@gmail.com
[2] University of Oradea, Department of Chemistry acaraban@uoradea.ro
[3] University of Oradea, Department of Mechatronics, omoldovan@uoradea.ro
[4] University of Oradea, Doctoral School of Engineering Sciences, moldovan.o.alin@gmail.com
[5] University of Oradea, Department of Mechatronics, dan.craciun28@gmail.com

ABSTRACT: In this paper, a low-cost IoT device using the DS18B20 temperature sensor is proposed. This IoT device has been designed in such a way that it can be used in the development of a predictive maintenance system intended for fire prevention in production halls or other types of buildings. Thus, the sensor must be configurable to be able to acquire temperature values at certain time intervals, and in case of detecting that a certain pre-set value of the acquired temperature is exceeded, to connect via HTTPS protocol to an API endpoint of a software solution allowing an automatic recording of a maintenance task. This maintenance task will allow the maintenance team to be informed and manage the incident with traceability. In the practical experiment validating the functioning of the IoT device, it was connected to an existing task management software solution on the market.
KEYWORDS: IoT, temperature sensors, predictive maintenance, secure connection, software design

1. INTRODUCTION

Predictive maintenance (PrM) is used throughout the industry for several reasons, mainly to reduce costs, minimize downtime, improve process quality, and can also have a major impact on environmental protection [1].

Many industries adopt PrM and rely on the predictive capabilities of different systems to ensure adequate safety and reliability of all systems [2]. In today's industry, smart products and components (equipped with embedded smart devices) are wirelessly networked and remotely monitored in real-time [3]. Maintenance strategies are responsible for the performance and smooth operation of manufacturing systems, becoming increasingly relevant for the sustainability of manufacturing processes [4]. Recent developments in information technology, data transmission, and information processing, such as IoT (Internet of Things) technology, allow PrM systems to be developed to be efficient and cost-effective. In the development of PrM strategies and tools, IoT plays an increasingly important role [5], [6], [7]. A significant part of the literature focuses on methods for acquiring sensor data and interpreting these data to identify potential problems in the production process or in some production-related processes. In general, these techniques are generally referred to as condition-based maintenance techniques for industrial equipment and processes [8]. These techniques are divided into three categories, each category uses signals from existing process sensors, uses signals from test sensors, or generates a test signal that is sent to the equipment and the response is evaluated [8].

In this paper, we aim to develop a low-cost IoT device that can be used in the implementation of an automated predictive maintenance system for fire outbreak prevention in industrial halls or other types of buildings. The device will need to be low cost in order to be adopted in the market as easily as possible, but in addition to this it will need to fulfil several functional characteristics as follows:

- allow the acquisition of temperature values at configurable time intervals;
- be able to configure temperature thresholds that automatically trigger warnings if exceeded;
- allow connection to a wireless network;
- allow access using the secure HTTPS protocol to an API endpoint that automatically records a predictive maintenance task in an existing IT system for collaborative and traceable task management.

Given these requirements, virtually every IoT device can be considered independent, performing its role regardless of the state of other devices.

2. HARDWARE ARCHITECTURE OF THE IOT DEVICE

The IoT device was developed using a DS18B20 digital temperature sensor interfaced with Arduino [9]. The microchip used was ESP8266 [10], as shown in Figure 1.

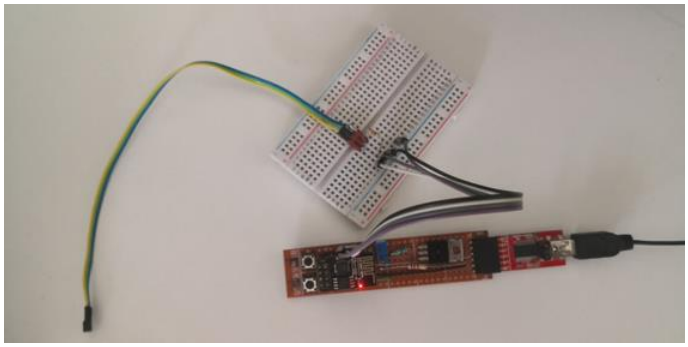

Figure 1. IoT device developed using the DS18B20 temperature sensor

The DS18B20 sensor is a temperature sensor with 1-Wire interface. The 1-Wire interface requires a single digital pin for bidirectional communication with a microcontroller.
The sensor usually comes in two forms. The first is in the form of a regular transistor, the other being in a waterproof probe style, which can be more useful when you need to measure something far away, underwater or underground, as can be seen in Fig.2.
The DS18B20 temperature sensor is quite accurate and needs no external components to operate. It can measure temperatures from -55°C to +125°C with an accuracy of ±0.5°C. This range of values that the sensor can read is wide enough, as the location of IoT devices using it will read ambient parameters in the production hall in the vicinity of possible sources of risk in terms of possible fires.
The resolution of the temperature sensor can be configured by the user to 9, 10, 11 or 12 bits. However, the default resolution at start-up is 12 bits.
The sensor can be powered with a 3V to 5.5V power supply and consumes only 1mA during active temperature conversions.

Figure 2. DS18B20 temperature sensor [9]

One of the biggest advantages of the DS18B20 sensor is that multiple DS18B20 sensors can coexist on the same 1-Wire communication channel. Because each DS18B20 sensor has a unique 64-bit serial code assigned at the factory, it is easy to differentiate them from each other. This feature can be a huge advantage when you want to control multiple DS18B20 sensors spread over a large area.

One of the biggest advantages of the DS18B20 sensor is that multiple DS18B20 sensors can coexist on the same 1-Wire communication channel. Because each DS18B20 sensor has a unique 64-bit serial code assigned at the factory, it is easy to differentiate them from each other. This feature can be a huge advantage when you want to control multiple DS18B20 sensors spread over a large area.
The sensor pins as shown in Fig.3 are:

- 1 - ground;
- 2 - is the 1-Wire communication channel and must be connected to a digital pin on the microcontroller;
- 3 - the sensor power supply, the voltage can be between 3.3 and 5V.

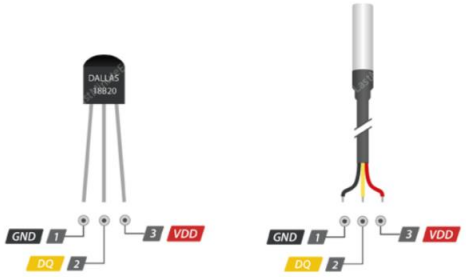

Figure 3. DS18B20 temperature sensor pins [9]

To connect the DS18B20 sensor to the Arduino, as shown in Fig.4, first connect VDD to the 5V output pin on the Arduino and GND to ground.
Then connect the remaining digital signal pin DQ to digital pin 2 on the Arduino. You will also need to add a 4.7k resistor between the signal and power pin to keep the data transfer stable.
If the DS18B20 sensor is not connected correctly, it will heat up and later fail.

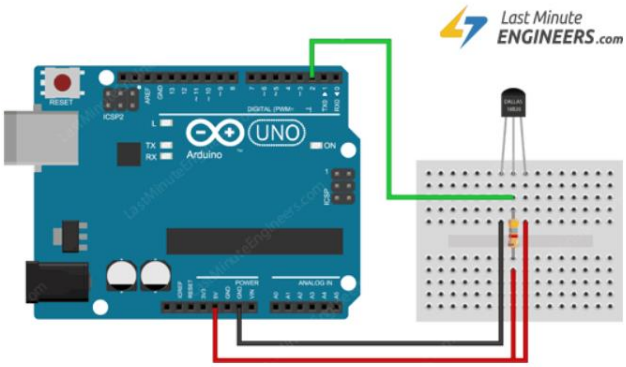

Figure 4. Connecting the DS18B20 temperature sensor to the Arduino [9]

3. SOFTWARE DEVELOPMENT OF THE IOT DEVICE

In order for the IoT device to perform the function for which it was developed, it need to be able to:

- connect to a WIFI network;
- read the temperature from the DS18B20 sensor;
- securely access the API endpoint to register a task. To validate the functionality, we used the Pri-Desk software solution [11], a commercially available solution.

The C++ programming language and the Arduino IDE programming environment were used to implement these functionalities. An Arduino program that run on a microchip has the following default structure:

void setup() { // This function includes code that will run once when the program starts:}

void loop() { // In this function insert the code that will run in the loop:}

Any library to be used must be included before declaring the setup () function.

3.1 Wi-Fi connection

To connect to WiFi we used the ESP8266WiFi.h library. Connecting to WiFi is done within the setup () function. Once the library is included, a global WiFi object is exposed, which will handle the connection. The code for the connection is:

```
WiFi.begin(ssid, password);
while(WiFi.status() != WL_CONNECTED) {
  delay(500);
}
```

Basically, you are trying to connect to the access point with the SSID name using the password. In the while loop you wait for the connection and check for a successful connection every half second.

3.2 Temperature acquisition from the sensor

To acquire the temperature from the sensor we used the combined DallasTemperature.h and OneWire.h libraries.
The Dallas 1-Wire protocol is somewhat complex and requires some complex code to parse the communication. To hide this unnecessary complexity, the DallasTemperature.h library is used so that we can issue simple commands to get temperature readings from the sensor. This Dallas Temperature library is a hardware-specific library that handles lower-level functions and must be paired with the One Wire library that allows communication with any device that uses the 1-Wire protocol, not just the DS18B20 sensor.
To begin with, a OneWire object must be declared, specifying at declaration which BUS it will communicate on. In our case this is done as follows:

OneWire oneWire(ONE_WIRE_BUS);

Once this 1-Wire communication object is declared, an object can be declared to manage the temperature sensor, such as:

DallasTemperature sensors(&oneWire);

Subsequently, in the setup() function, communication with the sensor must be initialized in the following way:

sensors.begin();

Whenever you want to read the temperature, use the requestTemperatures() method of the sensors object. Since the 1-Wire protocol allows multiple sensors to be accessed simultaneously, this method basically generates a temperature string corresponding to all available sensors. To access a particular sensor, one can do so based on its index. In our case the index is 0 because we have only one sensor. Accessing the temperature collected by it expressed in degrees Celsius is done as follows: sensors.getTempCByIndex(0).
The temperature in degrees Celsius thus obtained is a decimal number.

3.3 Secure access to Pri-Desk API endpoint

To achieve secure access to the Pri-Desk API endpoint we used the WiFiClientSecure.h and ESP8266HTTPClient.h libraries. The WiFiClientSecure.h library is intended to open a secure communication channel using port 443, and the ESP8266HTTPClient.h library is intended to enable communication using TCP/IP protocols.
In order to open a secure communication channel, a WiFiClientSecure object must be declared as follows:

WiFiClientSecure client;

Once this object is declared, in the setup() function you must set how the communication will be done, namely:

- without checking that the host being accessed is the one you want. This is done using the method: client.setInsecure();
- with verification of the host fingerprint, using the client.verify(fingerprint, host) method;
- with verification of the host root certificate, in which case the certificate to be verified must be specified, as follows: client.setCaCert(root_ca).

From a practical point of view, the difference between the fingerprinting and the certificate method is that the fingerprint is valid for a few months and the certificate for a few years, so changes to the code will have to be made more often or less often. However, if we are sure that the host we are trying to access is the one we want, we can use the Insecure version, which is the method we also use in our practical application.

To establish the actual connection, the client.connect(server, port) method will be used. In our case we connected to the portal.pri-desk.ro server, using port 443.

Next, in order to communicate using the TCP/IP protocol, an object of type HTTPClient must be declared. This class is exposed by the ESP8266HTTPClient.h library, which is the library specific to the ESP8266 microchip. Our declared object is http.

To be able to make an HTTP request, we do the following:

- initiate communication using the previously opened secure communication channel and specify the API endpoint with which communication is desired. This in our case is done as follows: http.begin(client, "https://portal.pri-desk.ro/api/tickets?company=primus");
- add the necessary information in the header. In our case we have specified what type of content will be sent, as well as how to authorize access. These things were done as follows:
    - http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    - http.addHeader("Authorization","Bearer BeareValue");
- construct the string that will represent the useful information to be transmitted to the endpoint. In our case this is as follows:

String httpRequestData = "channel=611e6c84e0ff3808a0c93269&category=611e69de8758920c912c3295&message=The temperature is "+ String(temperature) +"degrees and might be a problem! Please check it!&priority=regular&title=High temperature alert in area 1! ";

- the POST method http.POST(httpRequestData) is used to send the request and the HTTP code corresponding to how the request was processed is returned;
- close the connection to release the resources: http.end()

Once the request is processed, the predictive maintenance task will be visible in the Pri-Desk interface, as shown in Fig. 5.

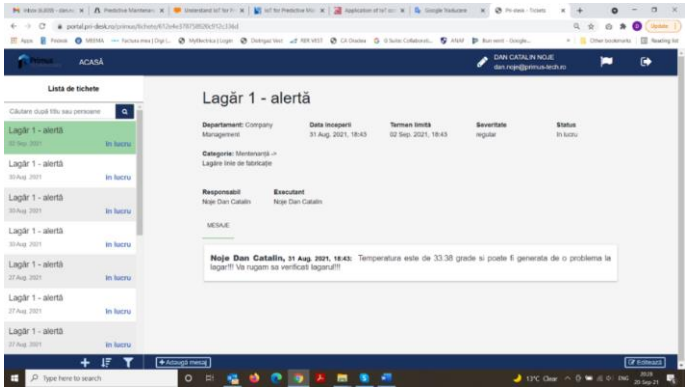

Figure 5. Predictive maintenance task recorded in Pri-Desk

3.4 Use of a timer

In order to set a timer to trigger certain processes from time to time, the SimpleTimer.h library was used. We declared the timer object, and the methods used were:

- setInterval(value), to set that a duration of value milliseconds will be timed. This will be done in the setup() function;
- isReady(), to determine if the set time has elapsed;
- reset(), to trigger the timing of a new time interval.

4. CONCLUSIONS

The development of an IoT device using the DS18B20 temperature sensor has been proposed and realized so that it exhibits a series of functionalities that allow its use in a small-scale implementation to perform predictive maintenance in an industrial hall or other types of buildings in terms of fire prevention. Since the IoT device as developed can operate independently from the rest of the components, as a further direction of development it is also possible to identify the use of certain interdependencies between certain nodes of the network of devices used in the monitoring process in order to increase the accuracy of alerts and also to provide more details on possible incidents that may occur.

Other future research directions could be the addition of other types of sensors, such as smoke detection, to the IoT device network, as well as the use of specific artificial intelligence algorithms to generate state predictions of the monitored systems.

5. ACKNOWLEDGEMENTS

The research has been funded by the University of Oradea, within the Grants Competition "Scientific Research of Excellence Related to Priority Areas with Capitalization through Technology Transfer: INO - TRANSFER - UO", Project No. 326/21.12.2021.

6. REFERENCES

1. S. Selcuk, “Predictive maintenance, its implementation and latest trends,” Proc. Inst. Mech. Eng. Part B J. Eng. Manuf., vol. 231, no. 9, pp. 1670–1679, Jul. 2017, doi: 10.1177/0954405415601640.
2. M. Pech, J. Vrchota, and J. Bednář, “Predictive Maintenance and Intelligent Sensors in Smart Factory: Review,” Sensors, vol. 21, no. 4, p. 1470, Feb. 2021, doi: 10.3390/s21041470.
3. L. S. Csokmai, R. C. Ţarcă, C. Bungău, and G. Husi, “A Comprehensive Approach to Off-line Advanced Error Troubleshooting in Intelligent Manufacturing Systems,” Int. J. Comput. Commun. Control, vol. 10, no. 1, p. 30, Nov. 2014, doi: 10.15837/ijccc.2015.1.1561.
4. C. Franciosi, A. Voisin, S. Miranda, and B. Iung, “Integration of I4.0 technologies with maintenance processes: what are the effects on sustainable manufacturing?,” IFAC-Pap., vol. 53, no. 3, pp. 1–6, 2020, doi: 10.1016/j.ifacol.2020.11.001.
5. K. Rahul, “Understand IoT for Predictive Maintenance in Manufacturing,” Software Advice, May 28, 2021. https://www.softwareadvice.com/resources/iot-predictive-maintenance/ (accessed Aug. 25, 2021).
6. R. C. Parpala and R. Iacob, “Application of IoT concept on predictive maintenance of industrial equipment,” MATEC Web Conf., vol. 121, p. 02008, 2017, doi: 10.1051/matecconf/201712102008.
7. A. S. Adila, A. Husam, and G. Husi, “Towards the self-powered Internet of Things (IoT) by energy harvesting: Trends and technologies for green IoT,” in 2018 2nd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS), Cavan, Apr. 2018, pp. 1–5. doi: 10.1109/SIMS.2018.8355305.
8. H. M. Hashemian, “State-of-the-Art Predictive Maintenance Techniques,” IEEE Trans. Instrum. Meas., vol. 60, no. 1, pp. 226–236, Jan. 2011, doi: 10.1109/TIM.2010.2047662.
9. “Interfacing DS18B20 1-Wire Digital Temperature Sensor with Arduino,” Last Minute Engeneers. https://lastminuteengineers.com/ds18b20-arduino-tutorial/ (accessed Dec. 07, 2021).
10. “ESP8266 - A cost-effective and highly integrated Wi-Fi MCU for IoT applications,” Espressif. https://www.espressif.com/en/products/socs/esp8266 (accessed Dec. 07, 2021).
11. “Pri-Desk,” Pri-Desk. https://pri-desk.ro/ (accessed Jul. 06, 2021).