Temperature Logger
Report

Quentin Rauschenbach

March 12, 2023 - WiSe 2022/2023

1 Overview

This report serves as a data sheet for a temperature logger build in the one-week block course "Practical Measurement Electronics and Interfaces in Ocean Sciences" run by Niels Fuchs and Markus Ritschel in winter term 2022/2023 at the Universität Hamburg. The course put a focus on parts and components of scientific instruments, basics of electronics, data transmission and acquisition methods as well as Arduino coding. The ultimate goal of the course was to enable us to build our own simple scientific temperature measurement device with two main components: an Arduino-like Himalaya Uno Rev3 and a 12-bit DS18B20 temperature sensor. This logger measures and writes to SD card spot and average temperature measurements in degrees Celcius in user defined time intervals. By default it takes a spot measurement every second, averages over ten values and writes out the 10th spot and the corresponding average together with the current time.

In the following, this report will cover a short technical description of the components used, wiring and code. Furthermore, I describe retrieval of the calibration and time constant for my logger and compare it to the other sensors created by other participants of the course. Finally, a final listing of the specification of the temperature logger is given.

2 Components

The "Arduino" with a data logger shield stacked on top is connected via three cables and a "Pull up" resistor to the DS18B20 temperature sensor (see Fig. 1). The following two subsections will describe the two main components in more detail.

2.1 Arduino and Data Logger Shield

The Himalaya UNO Rev3 comes with an ATmega328 Microcontroller running with up to 16MHz, 32KB of flash memory, 2KB RAM and 1KB ROM. It mimics an Arduino UNO R3 (Arduino, 2021) and will, from now on, for simplification be referred to as Arduino. The Arduino operates at a voltage of 5V supplied via a USB-B Port or a power jack. The USB port is also used for the serial communication to the computer. In addition to USB it is equipped with 14 digital input/output pins and six analog output pins. Hardware extensions can be equipped via shields on top of the Arduino. We use a data-logger shield, which provides us with an SD card slot and a battery-powered real time clock (RTC).

Our temperature loggers only use digital pins for temperature sensor, SD card andserial communication. The baud rate for this communication is set to 9600. Arduino code is written in C++. As an IDE we use Arduino IDE to code, compile and upload to the Microcontroller. This code then starts to be executed as soon as the Arduino is supplied with power.

2.2 Maxim Integrated DS18B20 Temperature sensor

The DS18B20 Temperature sensor by Maxim is a 1-Wire digital thermometer with (9 to) 12-bit Celsius measurements of temperature (Maxim-Integrated, 2019). It has three pins: one for ground (GND), one for voltage supply (V_{DD}) and, due to its 1-Wire protocol, only one data line (DQ). In order to operate more than one of these sensors at the same data line, each sensor comes with a unique 64 bit serial ID. This ID is stored on the ROM and starts with the family code on the least significant 8 bits, then continues for 48

bits with the serial ID number and ends with a cyclic redundancy check.

The sensor is able to operate between temperatures of -55 to +125°C with an accuracy of +/- 0.5°C between -10 to 85°C under a supply voltage V_{DD} between 3 and 5.5 V. In the 12-bit setup the output temperature has a resolution of 0.0625°C. This temperature information is stored after internal AD conversion on the 8 byte scratchpad in the two least significant bytes. This conversion can take up to 750 ms. When reading from scratchpad or ROM, data is transferred always starting with the least significant bit (LSB) of byte 0. The temperature data is stored into the 2 bytes or 16 bits as a sign-extended two's complement number. Hence the five MSBs, bit 11 to 15, represent the sign (1 indicating negative and 0 positive). The three LSBs, bit 0 to three, stand for the digits before and bit 4 to 10 for the digits behind the decimal point.

Transaction sequences for communication with the DS18B20 must always follow the same scheme:

1. During the *initialisation* a reset pulse is sent from bus master to slave(s) and answered with a present pulse by the slave(s) to indicate the sensor is ready for operation.

2. A *ROM Command* allows the master to get information about the sensor.

3. And finally a *DS18B20 Function Command* allows the master to e.g read and write from/to the scratch pad

3 Technical Description

3.1 Wiring

The previously described main components are connected via cables and a resistor as follows (Fig. 1): First to ground the sensor its ground cable is connected to one of the two ground pins of the Arduino. Then the temperature sensor gets its power supply with direct current via the 5V pin of the Arduino which is also connected to a pull up resistor of 4.7 kΩ. The DQ output pin of the DS18B20 is connected to one of the digital pins of the Arduino for communication. Additionally, it is connected to the pull up resistor too, to prohibit a short circuit between D3 and V_{DD} while still eliminating a floating state at DQ when the "switch" (D3) is open.

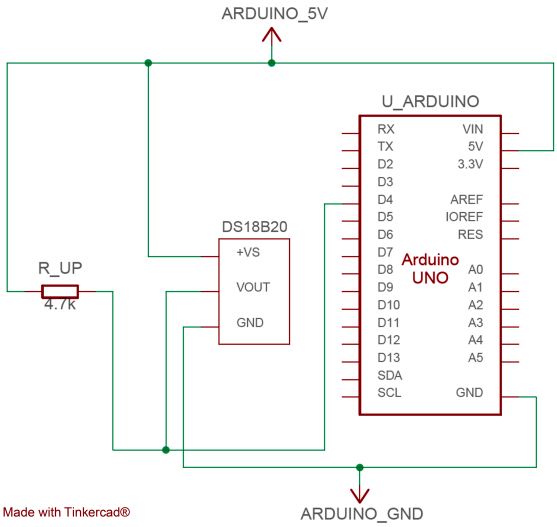

Figure 1: Circuit diagram of the temperature logger, with the pull up resistor R_UP, Vout the data line (DQ) and VS for the supply voltage pin V_{DD}.

3.2 Code

The C++ Arduino Code so-called sketch, which reads temperature and serial number information from the DS18B20 and writes it to SD card, consists of three parts: The first one to load libraries and declare global variables, then a void setup which is only run once after the Arduino has powered up and afterwards it continues with the void loop which in repeated until the Arduino is powered off again.

3.2.1 Libraries & Helpers

For our temperature loggers we use four different libraries: RTClib to communicate with the RTC, SD which enables communication with the SD memory card connected to the Serial Peripheral Interface (SPI), which therefore also requires the SPI library. Finally, OneWire allows to communicate 1-wire devices made by Maxim/Dallas.

In addition to the libraries we also use two helper files. One file (helper.ino) by Markus Ritschel contains several helper functions, e.g. two functions to write to SD card and serial monitor in one line. The other file (ds18B20_hex-helper.h) holds all of the HEX codes of the DS18B20 function and ROM commands needed for this specific task as well as the family ID number.

Type	**Variable Name**	**HEX**	**Description** (Maxim-Integrated, 2019)
ROM Command	READ_ROM	0x33	"allows the bus master to read the slaves 64-bit ROM code"
	SKIP_ROM	0xCC	"addresses all devices on the bus simultaneously without sending out any ROM code information"
DS18B20 Function Command	CONVERT_T	0x44	"initiates a single temperature conversion, resulting thermal data is stored in the 2-byte temperature register on scratchpad"
	READ_SCRATCHPAD	0xBE	"reads the content of the scratchpad, data transfer from least significant bit of byte 0 continuing to read through the scratchpad"
Family Code	IS_DS18B20_SENSOR	0x28	-

Table 1: Variables stored in ds18B20_hex-helper.h for readability of the main code.

3.2.2 Void Setup

After the declaration of global variables, loading the libraries and initialization of the OneWire bus to pin 4 the void setup starts by setting up the serial communication. With `Serial.begin(9600)` the baud rate is set to 9600. Afterwards an if statement checks whether the RTC is running and the SD card is connected. In case of a defect we print an error message to the serial monitor and set the RTC to the time of compilation of the sketch. Finally we print the header to the serial monitor and SD card with `printOutputln` from helper.ino.

3.2.3 Void Loop

The void loop consists of three transaction sequences of communication with the temperature sensor. In the first one after the always required reset pulse executed as `ow.reset` (see Sec. 2.2) we read the ROM (`READ_ROM`) in order to check the family code and get the registration number. ROM and DS18B20 function commands are always accessed via `ow.write(hex-code)` with hex code corresponding to the executed command. We store the 64 bits or 8 bytes of the ROM in the variable `rom_code` and then check if the first byte equals the family code `IS_DS18B20_SENSOR`. Then we store the next 48 bits as a string in `registration_number`.

In the second and third transaction sequence we skip the ROM command (`SKIP_ROM`). Afterwards we trigger the temperature conversion with `CONVERT_T` which is then saved on the scratchpad. Since we converted the temperature in the previous sequence we can now read it from the scratchpad in the third one with `READ_SCRATCHPAD` and save it to `sp_data`. Data transmission and therefore reading starts with the LSB of byte 0. To access the temperature information from `sp_data` we need to perform an 8-bit shift for byte 1 and combine it with byte 0 by ORing. Finally we have to shift the decimal point by multiplying the temperature with 2^{-4} to receive the correct float for the variable `tempCelcius`.

Now that the communication with the sensor is completed we sum the spot measurements of 10 iterations in an extra variable to calculate the average. So after 10 iterations of the void loop we write the average and the last spot temperature measurement together with the time, milliseconds of code run time and sensor ID to the serial monitor and the SD card. Finally the last part of the code ensures a constant measurement interval of one second by taking the difference between the run time and the next second and delaying the code by that amount waiting before going into the next iteration of the void loop.

4 Measurments

In order to calibrate and measure the time constant of all our temperature sensors we tied them around our reference sensor, a Greisinger GMH 3700 Series Pt100 High-Precision Thermometer, to minimise spatial differences (Fig. 2). For these two causes we executed one experiment each in the small tank of the ice laboratory at Max Planck institute for meteorology.

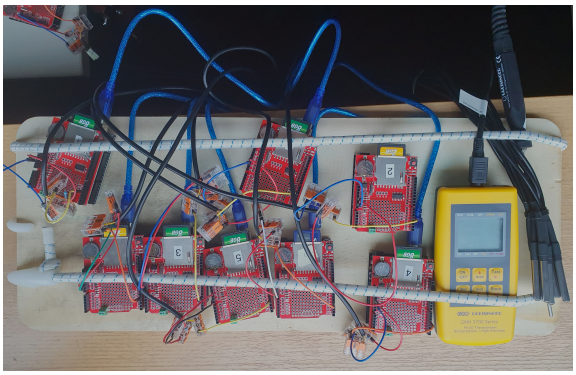

Figure 2: Arduinos secured on a wooden board and sensors tied together for the experiments. (Credit: Kirsten Schulze)

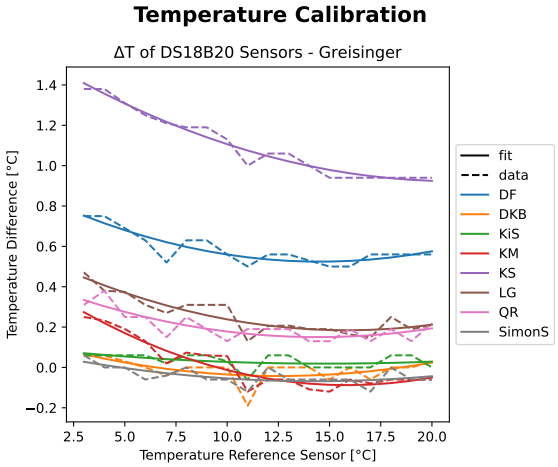

Figure 3: Differences between the individual sensors and the Greisinger reference sensor plotted against the latter. The solid lines show the second degree polynomial fit.

4.1 Calibration

To calibrate our sensors they were held into about 3 °C cold water. Then the heating plate below the tank was set to 40 °C to warm the water. We note down the time from the Greisinger at every degree increase. The experiment ended after reaching 20°C with the Greisinger due to time constraints.

In order to evaluate the measurements the time axis of the different sensors has to be synchronised first. For the two sensors without a functioning RTC we first create a time axis with the help of the milliseconds output. Then we can use shock time from the experiment described in section 4.2 and adjust the time axis of each sensor by the offset to the time of shock written down in the laboratory. Now we are able to calculate the difference between the DS18B20 and Greisinger values for each data logger and plot this against the Greisinger temperature to perform a second degree polynomial fit (numpy's polyfit) to retrieve a correction function for temperatures between 3 and 20°C.

$$y = ax^2 + bx + c \tag{1}$$

with offset y, Temperature x and fit parameters a, b and c. My sensor (QR) shows an offset below DS18B20's accuracy of 0.5 °C with a positive bias of 0.1 to 0.4 °C within the measured temperature range (Fig. 3). The polynomial fit leads to a correction function with $a = 1.407 \times 10^{-3}$, $b = -4.061 \times 10^{-2}$, $c = 0.443$.

4.2 Time constant

To retrieve the time constants of the sensors' we first let them adjust to the air temperature of the laboratory and then suddenly changed the temperature of the sensors surrounding medium by exposing them to previously cooled water. This has the beneficial side effect that we have a distinctive shock to synchronise our RTCs with. After this shock we waited for the temperature of our sensors to stabilize again. The temperatures and time of the measurements of the Greisinger had to be noted down manually every 10 seconds.

The time constant quantifies the time it takes for the sensor to reach 62% of the temperature difference. For our setup, transitioning from warm to cold, we can therefore use the following equation.

$$T_\tau = T_{end} + 1/e \cdot (T_{start} - T_{end}) \tag{2}$$

$$\tau = t(T_\tau) - t_{start} \tag{3}$$

Hence we first have to know when exactly our experiment started (t_{start}). In order to do that we detect the time when the temperature starts to drop for each sensor. Since we find fluctuations of 0.06°C even for stable temperatures we set a condition of a change of 0.1°C as a criterion. The starting temperature T_{start} we then retrieve by the mean over 30 values before t_{start}. Similarly we proceed for the end temperature. However, the slow stabilization requires a slightly different procedure to detect the end time after which we can take this mean. To compensate for this smooth transition we define the end time at the point where the running mean over 100 values doesn't change more than the usual fluctuations we get from the temperature sensor.
Together with the calibrations from section 4.1 we are now able to calculate the time constant with equation 2 and 3 (Fig. 4). For my sensor (QR) this results in a time constant of 253 seconds.

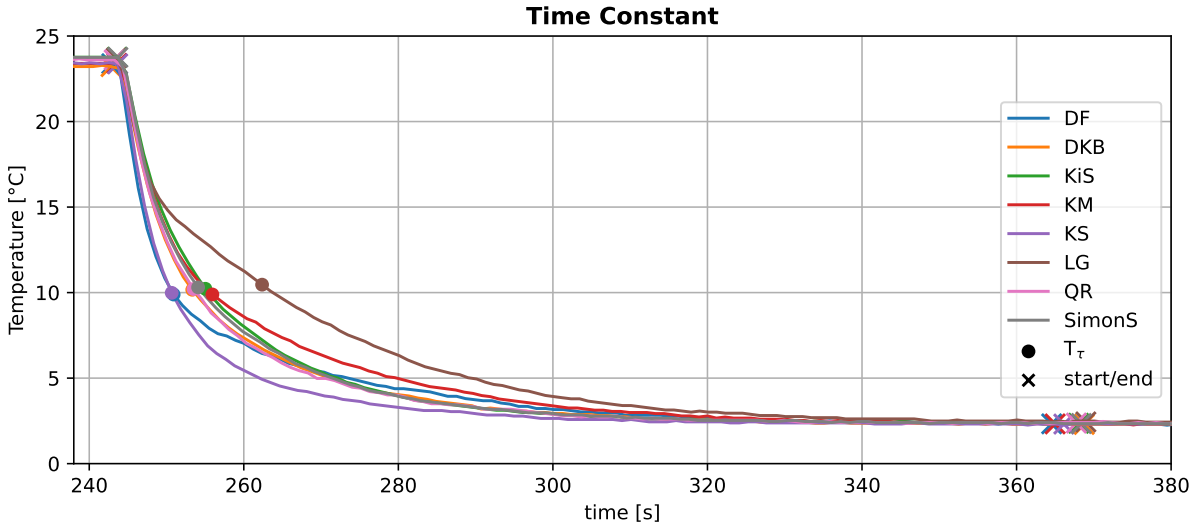

Figure 4: Temperature evolution of the "time constant experiment" for all sensors. Start and end times are indicated as crosses, the times when T_τ is reached are marked with dots.

The time constants of all sensors are summarized in table **??**. Finally, it should be mentioned that T_{start} and T_{end} are outside of the calibrated temperature range.

Sensor	T_{start} [°C]	T_{end} [°C]	τ [s]	Calibration Coefficients a [$\times 10^{-3}$]	b [$\times 10^{-2}$]	c
DF	23.38	2.33	250.9	1.71	-4.99	0.887
DKB	23.25	2.29	253.3	1.19	-2.98	0.144
KiS	23.76	2.36	255.0	0.36	-1.09	0.100
KM	23.40	2.32	255.9	2.19	-6.95	0.463
KS	23.38	2.32	250.7	1.47	-6.24	1.583
LG	23.75	2.44	262.4	1.59	-5.03	0.582
QR	**23.65**	**2.37**	**253.49**	**1.41**	**-4.06**	**0.443**
SimonS	23.74	2.35	254.09	0.76	-2.16	0.086

Table 2: Summary of all start T_{start} and end temperatures T_{end} (after calibration) as well as time constants τ and calibration coefficients a, b and c.

4.3 Comparison to other sensors

Before the calibration, two of the sensors (KS & DF) show a positive offset larger than DS18B20's accuracy of 0.5°C across the whole temperature range of 3 to 20°C used for the calibration experiment (Fig. 3). The DKB sensor, on the other hand, shows the best performance. For the time constant experiment and after applying the calibration we find a spread of 0.5°C for T_{start} and 0.15°C for T_{end} with DKB always at the lower and LG on the higher extreme (Tab. **??**). However both temperatures (start and end) lie outside of the calibrated range. The time constants range from 250 to 262 seconds. Again with LG at the upper end of the spectrum but this time DF and KS at the lower end. My own sensor lies roughly in the middle both for the offset before and differences after calibration as well as for the time constant.

5 Concluding specifications of the system

The temperature logger has the following technical specifications:

Resolution:

- Temporal: 10 s
  (user defined in main Arduino code)
- Temperature: 0.0625°C (given by DS18B20)

Range:

- Calibrated: 3°C to 20 °C
- DS18B20: min/ max -55°C to 125 °C

Accuracy:

- DS18B20: +/- 0.5°C between -10°C to 85°C

References

Arduino (2021). Arduino® uno r3 product reference manual. URL https://docs.arduino.cc/static/4cfa87b75db487ab7d2fa4aef2aebf08/A000066-datasheet.pdf.

Maxim-Integrated (2019). Ds18b20 programmable resolution 1-wire digital thermometer. *Rev*, 6:URL https://www.analog.com/media/en/technical–documentation/data–sheets/DS18B20.pdf.