Ultra Low Power Sensor Readout and Data Logger

For a battery-less system

Alec Reunis
Kjell de Wit

In collaboration with

BSc Electrical Engineering
Bachelor Graduation Thesis

TU Delft

Preface

This thesis is written as part of the requirements for the Bachelor Graduation Project at TU Delft, Electrical Engineering. It documents the work on the design and implementation of a batteryless temperature logger for cold-chain logistics.

We would like to express our gratitude to our supervisors Prof.Dr. K.A.A. Makinwa and Dr. Sijun Du for their guidance and support. We also like to thank our daily supervisors Floris van Mourik and Teije Onstein for their knowledge and support and we would like to extend special thanks to Martin Schumacher for his valuable assistance throughout the project. Finally, we are grateful for our colleagues: Rolando Russel, Waris Ibrahimi, Brian Joemmankhan, and Stoyan Dinev.

Alec Reunis & Kjell de Wit
Delft, June 2025

Abstract

Cold-chain logistics demand precise temperature monitoring to ensure the safety and quality of perishable goods during transport. Conventional solutions rely on battery-powered temperature loggers, which contribute significantly to electronic waste due to limited reusability. This thesis addresses this environmental concern by contributing to the development of a fully batteryless, wireless temperature logger designed for long-duration cold-chain monitoring. Focusing on the sensing and data logging subsystem, this work presents the design and implementation of an ultra-low-power system capable of accurate temperature readout and multi-week data storage under strict energy constraints. The system integrates a microwatt-level temperature sensor, a power-optimized microcontroller, and energy-efficient logging strategies to balance measurement accuracy, memory use, and power consumption. The final implementation is shown to have an idle power draw of <1µW and an energy use of 14µJ over the measurement and storage period, with a peak power of 411µW . The temperature measurements were found to be accurate within ±0.5°C . This accuracy and energy efficiency, demonstrates its potential as a sustainable alternative to traditional battery-powered loggers.

Contents

1 Introduction

1.1. Problem Justification

Cold-chain logistics, such as the transport of perishable foods or pharmaceutical items, require precise temperature monitoring to preserve product quality and safety [1], [2]. Ensuring that these items stay within safe temperature ranges during transport is critical to ensure the quality at arrival. Even short-term deviations from the optimal temperature range during transport can lead to spoilage, significantly reducing shelf life and contributing to global food waste. For example, bananas must be kept between 13.2°C and 14°C to maintain their four-week shelf life [3]. Slight deviations can trigger premature ripening or chilling injury, both of which diminish the product's value upon arrival [4].

To manage this, temperature loggers are commonly used to continuously record temperatures throughout the shipment. These devices allow post-transport evaluations and help predict remaining shelf life, ultimately reducing waste due to spoilage [2]. However, their widespread use comes with a downside. Many temperature loggers are powered by disposable batteries and are built for single or limited reuse. As the number of shipments grows, so too does the contribution of the loggers to electronic waste, which is a fast growing environmental problem [5], [6].

1.2. State of the Art

Current cold chain monitoring systems primarily focus on tracking temperature during food transportation using a variety of sensing and data recording technologies. Companies such as Sensitech, LogTag, and Coldstream offer reliable commercial temperature loggers capable of monitoring conditions throughout the shipping process [7]–[10]. These devices typically sample temperature at intervals of 30 to 60 minutes and can log data over several days to weeks. Data is retrieved post-transport via USB, PDF export, or wireless transmission, depending on the model. These systems offer temperature tracking using onboard data storage, but they are all battery-dependent which limits their sustainability.

Several academic efforts have explored batteryless sensor systems, often powered by RFID, NFC, or ambient energy harvesting [11], [12]. These designs eliminate batteries, making them more environmentally friendly and maintenance-free. However, these systems provide only real-time temperature readout and cannot store temperature histories over time. Their functionality is limited in scenarios where unattended monitoring is required.

Recent developments in ultra-low-power temperature sensors have significantly contributed to efforts toward batteryless or energy-harvested sensing systems. Commercially available digital sensors such as the Texas Instruments TMP75B [13] and the Microchip AT30TS75A [14] provide accurate temperature measurements (±0.5°C typical) over a wide range while consuming microampere-level current during active operation and nanoampere-level standby currents. In academic research, sensor designs have pushed power consumption even lower. Makinwa's survey compares various state-of-the-art CMOS temperature sensors achieving power consumptions below the nanowatt range

with resolutions below 0.1°C, showing even more potential for fully batteryless sensing platforms [15].

Despite progress, a key challenge remains: integrating batteryless operation, local data storage, and reliable wireless readout into a single autonomous platform. Current solutions tend to trade off one capability for another, either supporting long-term storage with batteries or offering batteryless operation with only on-demand sensing. However, ongoing advancements in ultra-low-power electronics, energy-efficient memory technologies, and wireless protocols present promising opportunities to develop integrated, fully autonomous cold chain monitoring devices that address these limitations.

1.3. Project Objective

This thesis is part of a larger engineering project that aims to design and prototype a fully batteryless and wireless temperature logger. The system is designed to autonomously sense and log temperature data throughout cold-chain transport and to allow later retrieval of the data via a contactless interface. The logger is composed of multiple subsystems:

1. **Energy harvesting:** Supply power from an external source. [16]
2. **Temperature readout and storage:** Capture and store temperature readings under tight power constraints.
3. **Data transmission and display:** Wirelessly retrieve and display temperature data. [17]

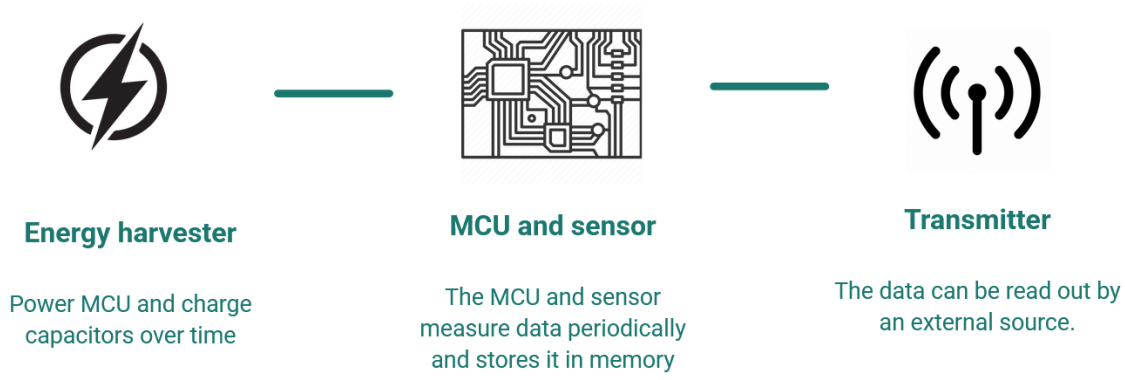


Figure 1.1: Diagram of Full Temperature Logger System

This thesis focuses specifically on the second subsystem shown in figure 1.1: the design and implementation of the temperature readout and storage. The goal is to achieve sensing and data logging functionality that meets the reliability and usability standards of existing battery-powered loggers, while operating under the tight power constraints of batteryless operation. This requires the integration of an ultra-low-power temperature sensor into a power-optimized system, as well as developing logging strategies that balance measurement accuracy, power consumption, and memory usage within the limited energy budget provided by energy harvesting.

1.4. Thesis Outline

The remainder of the thesis is structured as follows:

- **Chapter 2** describes the programme of requirements, defining the operational conditions, power constraints, sampling needs, and memory limitations that shape the design of the subsystem.
- **Chapter 3** describes the design and implementation of the sensing and logging subsystem. It discusses the selection of the temperature sensor and microcontroller, memory strategies, and the power-saving techniques used to ensure reliable operation under energy harvesting conditions.

- **Chapter 4** presents the evaluation of the system. It includes measurements of sensor accuracy, energy consumption, and performance characterization of the key components, as well as verification of the full subsystem. It also reflects on these results using the program of requirements.
- **Chapter 5** summarizes the key outcomes of the project and highlights possible directions for future work.

Further technical information and supporting data are provided in Appendices A, B and C.

2

Programme of requirements

2.1. General project requirements

The system outlined in this thesis is a subsystem of the larger project as explained in the previous chapter. To ensure the standalone readability of this document, the project's general requirements are outlined first, with the subsystem's more specific requirements following after. In the general requirements, all italicized entries are *not* applicable to this thesis, though they may influence the generation of certain requirements.

[1.1] *The device must operate solely from energy harvested from an external magnetic field.*

[1.2] *The device must not contain batteries.*

[1.3] The device must measure temperature with a low power sensor.

[1.4] The device must store the measured data to be read out at a later time.

[1.5] The device must also support a one-time readout to verify its operation.

[1.6] *The data must be readable wirelessly.*

[1.7] The prototype must be built out of commercially available products.

[1.8] The prototype must be designed and built within 8 weeks.

[1.9] The cost of the prototype should not exceed €200.

2.2. Subsystem requirements

From the above requirements and the chosen scope outlined in the previous chapter, it is possible to hone in on requirements specific to the subsystem. These have been divided into three categories: Operational, Power, and Memory/Performance.

2.2.1. Operational requirements

The operational requirements as set out in this section outline demands for the functionality of the device in a global sense. The non-italic requirements above can also be seen as part of this set. The *Agreement on the International Carriage of Perishable Foodstuffs and on the Special Equipment to be Used for such Carriage (ATP Agreement)* sets standards for temperature control and traceability in international transport [18]. This motivates requirements such as minimum measurement frequency, logging duration, and operating temperature range.

[2.1] The system must measure temperature at least every hour.

[2.2] The system must be capable of storing measurement data for a period of 4 weeks.

[2.3] The system must operate around a voltage of 2V DC.

[2.4] The system must operate reliably in a temperature range of -30 to 50°C.

[2.5] The components used must be solderable by hand.

[2.6] The system should preferably store timestamps along with the temperature data

2.2.2. Power requirements

As the whole system needs to work off harvested energy, very little power will be available. The main KPIs here are peak power (should be less than what the capacitor can deliver) and energy usage (should be less than what the capacitor can store). As the performance of the energy harvester was not known at the start of this project, and implementations of magnetic energy harvesting vary greatly [19], any kind of hard boundary on both power and energy would be incredibly arbitrary. Nevertheless, the goal of the project is to make an ultra low power device, so the maximum peak power was set at 1mW. Thus, this section consists mainly of trade-off requirements that all point to the same goal: balance peak power with energy draw. Similar challenges in balancing power consumption and energy availability have been highlighted in prior research on ultra-low power temperature sensors and RFID-enabled cold chain monitoring systems [20]–[23]. The hardest boundary here is that of the temperature sensor. As this project builds upon existing research, the goal is to select a sensor that approaches the performance characteristics of research-grade sensors as closely as possible while using commercially available products [15]. Energy is not a consideration in idle mode, since the energy harvester is continually generating energy. However, the device should leave enough surplus power so that the energy harvester can charge the capacitor in time for the hourly measurement.

[3.1] The temperature sensor peak power should be as close to 100nW as possible.

[3.2] The design must have a peak power draw of less than 1mW.

[3.3] The design should preferably minimise the peak power draw.

[3.4] The design should minimise the energy used per measurement.

[3.5] The design should minimise the power draw when in idle.

2.2.3. Performance/memory requirements

Finally, cold-chain management practices set certain conditions on the performance of the temperature measurement. The main KPIs here are data resolution, which directly influences the amount of bits needed to store the data, and temperature accuracy, which is a characteristic of the chosen sensor. To ensure comparable performance to existing commercial systems, the system must meet similar standards [7]–[10].

[4.1] The temperature data must have a resolution of 0.25°C per bit or less (10 or more bits).

[4.2] The temperature data must have an accuracy of ±1°C or less.

[4.3] In order to support 4 weeks of measurement data at 1 measurement per hour and at least 10 bits of resolution, the storage memory must have a size larger than 1344 Bytes.

[4.4] The temperature data should preferably have a resolution of 0.0625°C per bit (12 bits).

3

System Design

3.1. Design overview

The proposed design is the middle part of the three-part system outlined in chapter 1: temperature readout and storage. It is comprised of a microcontroller unit (MCU) and a low-power temperature sensor. The MCU expects about 2 volts at the input, and outputs temperature data to the non-volatile memory of a low-power NFC module. The MCU is normally idle, woken up hourly by the onboard real-time clock (RTC). After waking up, it performs an *active cycle*, whereby it reads the current temperature value from the sensor and stores it in the memory. At the end of the cycle, the MCU returns to its idle state.
The remainder of this chapter will describe the used components, an overview of the firmware, implemented power-management techniques, and the data storage method.

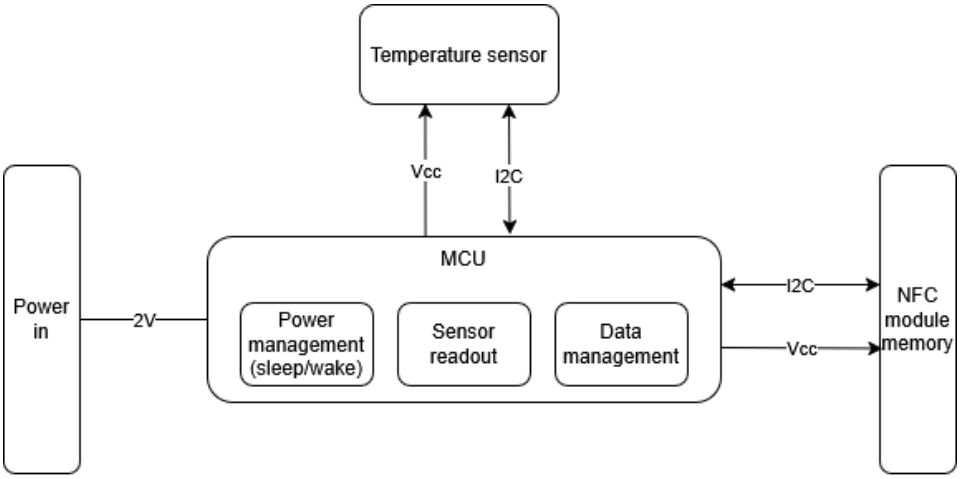


Figure 3.1: Functional block diagram of the subsystem

3.2. Component Selection

3.2.1. Microcontroller unit

The requirements outlined in chapter 2 do not necessarily call for an MCU to be used. However, from requirements [2.1] and [2.6], it follows that some kind of timekeeping device is needed. Furthermore, storing the temperature data, as per requirement [1.4], is most easily done digitally. Therefore, the temperature data must be converted to a digital value either natively inside the component, or via an analog-to-digital converter. This, along with similar projects also using microcontrollers [21], lead to the choice to use an MCU in this project.

As there are many different MCUs on the market, some base criteria were set up using the requirements stated in chapter 2:

[5.1] Real-Time Clock (RTC) for periodic wakeup and timestamps (from requirement [2.6])

[5.2] Low power modes (from requirement [3.3] - [3.5]

[5.3] Around 2V operation (from requirement [2.3])

[5.4] Development board available (from requirement [2.5], [1.8])

This list served to narrow down the possible options and after careful consideration, the STMicroelectronics *STM32U083MC* was selected, onboard the *STM32U083C-DK* development board [24] [25]. This choice was made base on the provided features, available development board and its price (requirement [1.9]). Hereafter, these will be referred to as the MCU and the development board, respectively. Some of the most relevant features of the MCU are summarised in table 3.1, everything else can be found in the datasheet [24]. A top-view picture and layout drawing of the development board can be found in Appendix C.1.

Table 3.1: STM32U083MC main features from the datasheet.

Feature	Value	Unit
CPU	32-bit Arm Cortex M0+	
Operating frequency	0.1 - 56	MHz
Operating voltage	1.7-3.6	V
Operating current (Run mode)	120	µA/MHz
Operating current (Stop 2 mode)	2.3	µA
Operating current (Standby mode)	900	nA

Those familiar with the technology might consider a 32-bit MCU overkill, and that would be a valid concern. The MCU is overflowing with features not relevant to the project and could probably be used to send someone into space. However, as of writing, the STM32U0 line is brand new (released around March 2024 [26]) and has some of the lowest power 32-bit MCUs on the market. It does get outclassed by some 16-bit and many 8-bit MCUs in terms of power draw, but none were found that had both an RTC and a development board, or non-RTC counters able to count up to 1 hour. A similar study used the Texas Instument MSP430 platform [21], but the models on available development boards did not offer lower power consumption than the STM32U0 MCU.

The MCU's system clock is highly configurable, with the following possible sources as per the datasheet [24]:

- 4-+48 MHz high-speed oscillator with external crystal or ceramic resonator (HSE).
- 16 MHz high-speed internal RC oscillator (HSI16).
- Multispeed internal RC oscillator (MSI) able to generate 12 frequencies from 100 kHz to 48 MHz.
- System PLL (Phase Locked Loop), which can be fed by HSE, HSI16 or MSI. It provides a system clock up to 56 MHz.

These clock signals can be downscaled by dividers before reaching their destination, allowing some customisation of otherwise static oscillator frequencies. The RTC and select other peripherals can be clocked by the 32kHz Low-Speed RC Oscillator (LSI), or the Low speed oscillator with external crystal (LSE). Some peripherals can be configured to use a different clock than the system clock, and oscillators can be enabled and disabled through software. As the MCU's current draw is highly dependent on the clock speed, this is an important piece of the power saving puzzle.

Another powerful power-saving measure is the inclusion of low-power modes. The MCU has many different modes of operation, each providing a trade-off between the availability of features, power

draw, and wake-up time. In this project, the following modes were used:

- Run: Default mode after startup or wake-up from Standby mode.
- Low-power run: Slightly lower power usage than Run, CPU limited to 2MHz or lower.
- Stop 2: Lowest power consumption while still retaining the MCUS SRAM and register contents, as well as GPIO states. All system clocks disabled, LSI and LSE still running and able to clock timers. System can wake up from interrupts.
- Standby: Second lowest power consumption. SRAM contents (thus program data) lost. Only LSI and LSE enabled, RTC can be enabled. System can wake up from interrupts.

How exactly these are used and configured will be explained in section 3.4.

Additionally the MCU features up to 68 GPIO pins, most of which are left unused. Two pins are used as the respective voltage sources for the sensor and external memory module. This way, the devices can be powered off completely when not in use, fully negating their shutdown current consumption. This technique will be referred to as *power gating* for the remainder of this thesis. The consequence of this technique is that the devices cannot be placed on the same I^2C bus, because the V_S line source varies. Thus, four pins are used as I^2C bus connectors, allowing the sensor and the memory to be on different buses. As an open-drain system, the I^2C buses require pull-up resistors, which are the third large source of power draw for the system. This is also explored further in section 3.4

The development board features a configurable voltage regulator, allowing simulation of the final operating conditions while powered by a USB Type-C cable. While useful for prototyping, this does form an obstacle for eventually hooking the device up to the energy harvester. Through the provided connectors, the board can only be powered by a 5V source, which is then down-converted by the LDO regulator to feed the MCU. Fortunately, the board also features two header pins placed between the regulator and the MCU. These are found at JP7 in figure C.2. The pins are normally connected by a jumper and are intended for facilitating current measurements. Therefore, by removing the jumper, the MCU is disconnected from the LDO regulator. Then it is possible to connect an external voltage source to the MCU side of the header, completely bypassing the regulator and providing free choice of supply voltage, like the 2 volts expected from the energy harvester.

3.2.2. Temperature sensor

The absolute most important requirement for the sensor, is requirement [3.1]: peak power as close to 100nW as possible. Combined with the requirement for hand-solderable components, two possible options were identified: the Texas Instruments (TI) *TMP75B* [13] ATMEL (AT) *AT30TS75A* [14]. According to their respective datasheets, both sensors operate at maximum 170-180µW at 2V. This is several orders of magnitude more than the research-level sensor, which shows the clear limitation of working with commercial products. As with the MCU, some relevant characteristics are shown in table 3.2.

Due to some ambiguity in the datasheets, it was not immediately clear which sensor would perform best. Therefore, both sensors were ordered and tested for peak power and total energy consumption for a single temperature measurement. In the end, the TMP75B was chosen due to both a lower peak power, as well as a lower energy consumption. A more detailed comparison can be found at the end of this subsection.

Table 3.2: Most relevant features of the temperature sensors from the datasheets.

Feature	AT30TS75A (AT)	TMP75B (TI)	Unit
Operating voltage	1.7 - 5.5	1.4 - 3.6	V
Current during conversion (typ - max)	60 - 85	45 - 89	µA
Conversion time (typ - max)	50 - 75[1] 200-300[2]	27 - 35	ms
Shutdown mode current (typ - max)	0.4 - 2.5	0.3 - 8	µA
Accuracy (-25°C to 85°C)	±0.5 (typ) - ±2.0 (max)	±0.5 (typ) - ±2.0 (max)	°C
Resolution	9 - 12	12	Bits
Data form factor	Digital	Digital	
Communication protocol	I^2C	I^2C	
Device form factor	SOIC-8	SOIC-8	

[1]At 10 bits resolution (0.25°C).
[2]At 12 bits resolution (0.0625°C).

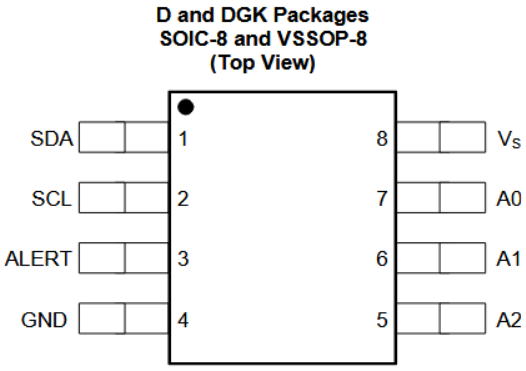


Figure 3.2: Pinout of the TMP75B, from [13].

In order to understand the comparison, it is useful to first examine the operational characteristics of the sensors. Since the TMP75B is used, this explanation will be limited to this sensor. Nevertheless, both sensors are designed as drop-in replacements for the industry-standard LM75 and are thus very similar.

The TMP75B used in this project is a low-voltage cousin of the TMP75, and the predecessor of the newer TMP1075, however the latter was not available for purchase. It is packaged in the 8-lead SOIC format, a schematic of the pinout can be found in figure 3.2. This package includes a BJT bandgap-based temperature sensor, which generates a voltage Proportional To Absolute Temperature (PTAT). This voltage is then digitised by an analog-to-digital converter and converted to a 12-bit temperature value, allowing for a resolution of 0.0625°C per bit. The value is then stored in two internal 8-bit registers. This conversion takes time and is the primary driver for the sensor's power and energy consumption. Figure C.4 in appendix C.2 shows the functional block diagram of the sensor, taken from [13].

A serial interface allows access to the digitised data and configuration of the device. This interface supports two-wire communication protocols and is specifically designed for I^2C and SMBus. The I^2C interface operates in slave mode, with speeds up to High-speed mode, at 3.4Mbits/s. The temperature data is stored in a left-justified format, with the most significant byte (MSB) storing 8 out

of the 12 bits, corresponding to a resolution of 1°C. The remaining 4 bits are stored in the 4 most significant bits of the least significant byte (LSB). When reading out the temperature, the MSB is sent first, followed by the LSB.

Using the serial interface, the device can also be configured by altering the configuration register. Many features are unused in this project, like the alert function when crossing configurable maximum and/or minimum temperatures, or the configurable measurement frequency. The features of interest here are the shutdown mode, which lowers the sensor's current draw when measurements are not needed, and the one-shot feature. The latter allows the sensor to exit shutdown mode to perform a one-time measurement before returning to shutdown mode. That is the ideal use case for this project.

However, the TMP75B offers an additional advantage: after power-up, it immediately starts a conversion without the need for configuration. This is the main function that sets it apart from the AT sensor, which requires the user to configure the measurement resolution. Combined with the GPIO-based power gating, this means that the only serial communication necessary is that to read out the temperature of the initial measurement. This allows the MCU to operate in a low-power mode during the measurement period, lowering the power and energy consumption.

Now that a base understanding of the sensor functionality has been achieved, it is possible compare both devices. From table 3.2, the TI sensor appears to be the more favourable option due to its lower typical conversion current and faster conversion time. In theory, this should result in both lower power consumption and lower energy used per measurement. The main drawback is its shutdown mode current, which is higher than the AT sensor, but this is solved by the power gating. However, these values are pulled from the datasheets and are based on characterisation, which does not always reflect reality. This was especially true for the conversion time of the AT sensor. For a resolution of 10 bits, it was possible to reliably read accurate values after 20ms, promising the possibility of a lower energy usage due to a faster execution time.

This highlights an important consideration. With all else being acceptable, the deciding factors were the sensors' respective peak power and total energy consumption. While these are connected, it is possible that a high-power device can still use less energy due to a faster execution time, leading to a trade-off situation. In consideration with the energy harvesting team, low energy was determined to be a priority, so the device with the lowest total energy consumption was selected.

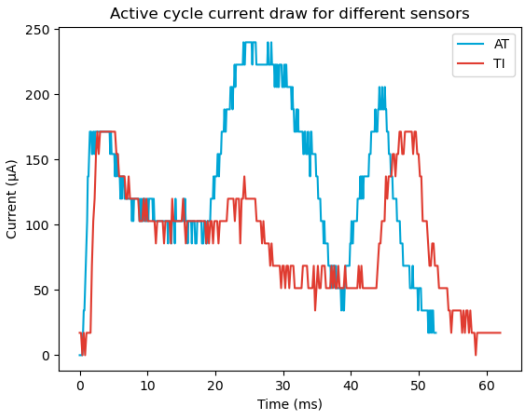


Figure 3.3: Current draw of the AT sensor vs the TI sensor, at 2V

The comparison was structured as follows: a program was written to the MCU, which simulated end-of-project conditions. The MCU starts in Standby mode, wakes up, powers on the sensor, enters Stop 2 mode if possible, reads data, and returns to Standby. The current drawn during this cycle was measured over time using an oscilloscope and a shunt resistor. More information about this measurement setup can be found in chapter 4, the Python code can be found in appendix B.2.

From figure 3.3, the TI sensor shows superior performance, which is also backed up by the numbers:

The power can be calculating by multiplying the measured current with the static supply voltage of 2V. With an approximate peak power of 343 µW and a total energy of 10.8 µJ, the TI sensor easily outperforms the AT, which has a peak power of approximately 479 µW and total energy of 14.01 µJ.

3.3. Firmware

The firmware for this project was written entirely in C, leveraging the powerful HAL library included with the STM32CubeIDE. This library provides high-level functions for most actions one would want to take within a program, thereby avoiding tedious management of the MCU's various registers. This drastically speeds up program development. The code can be found in appendix A

As mentioned earlier, the MCU in Standby mode most of the time. Only the RTC is active, sending an interrupt when the wake-up counter reaches 1 hour. This kicks off the *active cycle*, a diagram of which can be found in figure 3.4.

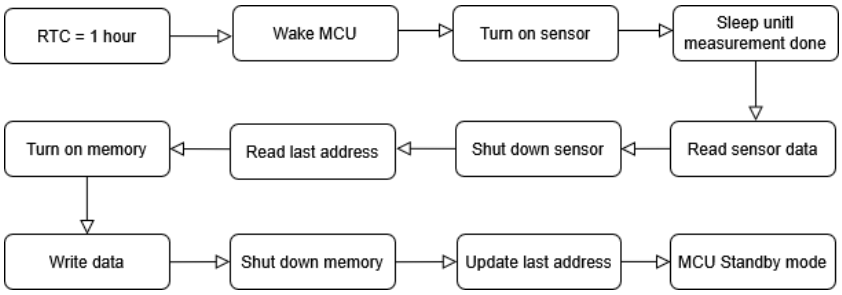


Figure 3.4: High-level flowchart of an MCU active cycle.

Wake MCU:
When the RTC's wake-up counter reaches 1 hour, an interrupt is generated which wakes up the device from Standby mode. Because Standby mode powers down all system memory, the program state and variable values are lost, and the MCU behaves as if it were just powered on. This also means that it operates in Run mode after wake-up, with a frequency of 4MHz. As soon as possible after waking up, the clock frequency is lowered to 800kHz, and the MCU enters Low Power Run mode. Once in Low Power Run, the MCU is ready to perform its tasks.

Sensor readout:
The sensor readout function performs four steps:

1. Turn the sensor on. The GPIO pin connected to the V_S of the sensor is turned on, providing power to the sensor and the I^2C bus.
2. Sleep until the measurement is done. The MCU enters Stop 2 mode for 15 ms to conserve power while the sensor takes a measurement. This time was determined experimentally, as it includes startup time of the sensor, which is not explicitly mentioned in the datasheet. A low-power timer in the MCU is used to control this period, waking the MCU by interrupt.
3. Read the sensor data. The I^2C bus is used to read the measured temperature. This consists of writing the sensor's device address *0x9E*, followed by the address of the temperature register *0x01*. Then, two sequential read requests transfer the data into a 16-bit signed integer. For ease of debugging, this left-justified data was then right-shifted by 4 bits, so that the positive value could just be multiplied by the resolution. This data format was kept for the final product, even though this functionality was not strictly necessary.
4. Shut down the sensor. After obtaining the temperature value, the GPIO pin is switched back off, cutting off power to the sensor and its I^2C bus.

The function returns the right-shifted sensor data as a 16-bit signed integer. This was useful for debugging, because this data could simply be multiplied by 0.0625 to obtain the measured temperature, which could then be printed to the COM port of a computer via UART. The same is true for the values in the memory, so this format was kept in the final design.

Memory operations
The memory module was provided by a different team, so will only be described briefly. It is the STMicroelectronics ST25DV64KC, an NFC chip with 64kb of built-in EEPROM memory. As usual, the most relevant features are listed in table 3.3 and more detailed information can be found in the datasheet [27].

Table 3.3: Most relevant features of the memory module from the datasheet.

Feature	Value	Unit
Supply voltage	1.8 - 5.5	V
Address length	16	Bits
Block size	4	Bytes
Communication protocol	I^2C	
Block write time	5	ms

Table 3.4: Data storage format in the memory module.

Block address	Byte 0	Byte 1	Byte 2	Byte 3
0	Reserved			
4	Reserved			
8	MSB0	LSB0	MSB1	LSB1
12	MSB2	LSB2	MSB3	LSB3
16	MSB4	LSB4	MSB5	LSB5
...	...			

Data can be written byte-per byte into the memory via I^2C . Only the first byte address needs to be given, after which the module auto-increments the write address for each consecutively received data byte. Because the temperature data is 12 bits long, it is stored in 2-byte words, in big-endian format: the MSB is stored at the lowest byte address. This results in a logical left-to-right legibility of the number for easier debugging. Table 3.4 shows a diagram of how the data is stored. Byte address 0-7 are reserved by the readout team for NFC-specific features, so the temperature data starts at byte address 8. The memory is assumed to be cleared via NFC before starting a logging period, and it provides plenty of space for the four week target. Therefore, there is no need for any data management.

In order to avoid overwriting previous measurement data, and to ensure all data is stored sequentially, the MCU needs to keep track of the last-written address. However, program variables are lost when entering Standby mode. Fortunately, the MCU provides nine so-called backup registers for this express purpose. Where most other registers onboard the MCU return to their reset values upon entering Standby mode, the backup registers are only reset upon a Tamper event. As long as all Tamper pins are disabled, which they are for this project, these registers will not lose their value. The exception to this is full system power loss, where the registers enter a latched state and the data is neither lost nor reliably kept. Storing the last-written address, which corresponds to the address of the MSB byte, in a backup register allows the program to read it back during the active cycle. This value could alternatively be stored in the EEPROM of the external memory at a fixed address, but this would add an extra read and write operation (reading the previous address and updating the new one) to the active cycle, requiring the memory to be turned on for three times longer, causing additional energy drain.

The memory write function takes the temperature data as an argument and carries out the following

five steps:

1. Read the last address. The last-written MSB address is read from backup register 0 and incremented by 2 to obtain the new memory address. If the value of the backup register is zero, the function assigns address 8 as the new memory address.
2. Turn the memory on. Just like the sensor, the GPIO pin connected to the V_S pin of the memory module is turned on, providing power to it and its I^2C bus.
3. Write data. The previously obtained temperature data is placed into a write buffer consisting of an array of two unsigned 8-bit integers. Via the I^2C interface, the component's I^2C address is sent first, followed by the target byte address, the MSB, and lastly the LSB.
4. Shut down memory. Once more, the GPIO pin is turned off, cutting power to the memory and its I^2C bus.
5. Update the last address. The memory address of the just-written MSB is saved back into backup register 0.

With the active cycle complete, the MCU re-sets the RTC wake-up timer for 1 hour and enters Standby mode.

3.4. Power Saving

A variety of techniques were used to bring the power consumption of the system down from the default consumption of the components. This section will explain the techniques and their impact on the power profile.

3.4.1. I^2C pull-up resistors

The following section will assume basic knowledge about I^2C . All relevant information can be found in the I^2C manual by NXP [28].

As touched upon briefly earlier, the size of the pull-up resistor has a large influence on the power consumption of the I^2C bus. This is due to the open drain design: normally, the bus is held high by a pull-up resistor connected between the bus and the V_S line. To perform a write, the bus is pulled to ground, allowing current to flow through the resistor. The value of this current can easily be calculated with Ohm's law:

$$I = \frac{V}{R} \tag{3.1}$$

Figure 3.5 shows a mathematical model of the current for different values of the pull-up resistor. To stay within the 1mW peak power requirement ([3.2]), only values above 4kΩ were considered. The model clearly shows the inverse relationship between the current and the resistance, the asymptotic shape implies that each increase in resistance will give diminishing returns. Care must also be taken not to increase the resistance so high that it cannot reliably pull the bus up to the level of V_{OH} . The Python code for the model can be found in appendix B.3.

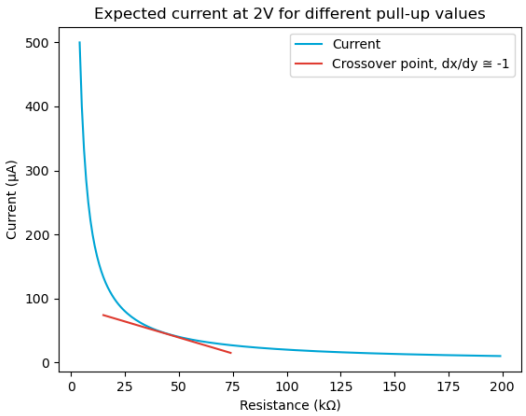


Figure 3.5: Model for the current sunk during I²C communication, in function of the pull-up resistance.

Selecting the resistor value was done by calculating the gradient for each point and then finding the point where this gradient is closest to -1. Past this point, an 1kΩ increase in resistance, results in a decrease of less than 1µA and it becomes less and less impactful. The line tangent to this crossover point is shown in red. For the current model, this point is at a resistance of 44kΩ , which corresponds to a current of approximately 45µA . Mapping this to available components, a value of 47kΩ was ultimately selected. This corresponds to a current of 42.5µA , which is well within the requirements.

Of course, using a pull-up resistor that is more than ten times larger than the standard size has a profound effect on the data rate. This is directly linked to the bus frequency and thus the SCL period T_{SCL}. The resistor value influences the period by affecting the rise time of the bus voltage, given by the following equation, from [28]:

$$t_r = 0.8473 \cdot R_p \cdot C_p \tag{3.2}$$

With t_r, R_p and C_p the rise time, pull-up resistance and bus capacitance, respectively.

According to section 7.2 of the I²C manual, it is possible to calculate the new SCL frequency with the following formula [28]:

$$T_{SCL} = \frac{1}{T_{LOW(min)} + T_{HIGH(min)} + t_{r(actual)} + t_{f(actual)}} \tag{3.3}$$

With T_{LOW} and T_{HIGH} being the Standard mode values of 4.7µs and 4.0µs , respectively. t_f is generally drastically faster than the other parameters, so it can be ignored. t_r can either be calculated with eq. 3.2, or measured on an oscilloscope. Because the bus capacitance was not known and the manual also warns that the calculated value will be faster than reality, the frequency was determined another way.

The I²C controller self-limits its frequency by reading back the value of the SCL and SDA lines before modifying them. Therefore, it is possible to measure the SCL frequency associated with a 47kΩ pull-up resistor. To do this, the SCL and SDA signals were measured using an oscilloscope, using the cursors to determine the SCL frequency. Figure 3.6 shows a part of the oscilloscope display.

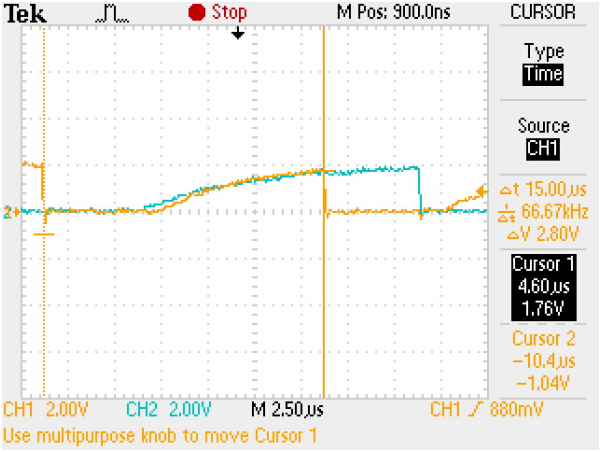


Figure 3.6: Screenshot of the oscilloscope display measuring the I²C signal with 47kΩ pull-up resistors. CH1 is SCL, CH2 is SDA.

As seen in the figure, the measured frequency is about 66kHz. To ensure that the device always operates as expected, the MCU's I²C module was configured to run at a frequency of 60kHz. This way, some margin is built in for any thermal deviations in the resistor. Using this value, the energy per bit can be calculated with the following equations:

$$T_{SCL} = \frac{1}{f_{SCL}} = \frac{1}{60kHz} = 16.7\mu s \tag{3.4}$$

$$P = I \cdot V = 42.5\,\mu A \cdot 2V = 85\,\mu W \tag{3.5}$$

$$E = P \cdot T_{SCL} = 85\,\frac{\mu J}{s} \cdot 16.7\,\mu s = 1.42\,nJ \tag{3.6}$$

This results in an energy per bit of 1.42 nJ. Every byte transmission consists of at least 9 bits (8 data + 1 ACK), so the energy per byte becomes 12.78nJ, ignoring start and stop signals. Through the entire active cycle, a total of 8 bytes are sent and received by the MCU (device adresses and temperature data to and from the sensor and external memory), which brings the energy cost of the I²C communication to about 102nJ.
The calculation also shows the peak power draw of the bus in eq. 3.5. At 85µW , the peak power is well below 1mW and sufficiently minimised, without sacrificing the reliability of the bus connection.

3.4.2. MCU clock speed

As mentioned in section 3.2, the MCU offers a wide range of clock sources with variable frequencies. The choice of clock source and frequency greatly impacts the MCU's current consumption, so with energy and power being such important parameters in this project, choosing the right source and frequency is paramount. Running the system at a lower frequency is an easy way to lower its power consumption, but the slower execution time might lead to a higher energy usage over the course of an active cycle.

Aside from the frequency, the actual clock source also plays a role in the current consumption. Every clock has an associated draw, often inversely correlated to its accuracy. For example, the HSI16 clock runs at 15.88 - 16.08MHz, with a temperature drift of -1 to +1% at best and consumes about 170µA . The MSI on the other hand, can be configured for many different frequencies. The actual frequency is then between -1% and +1% of the configured frequency. Its temperature drift is -3.5% to +3% at best, and almost doubles when operating outside of the 0°C - 85°C range. In return, it only consumes about 80µA at 16MHz, reaching as low as 1µA at 100kHz. Since the development board

does not include the resonator required to use HSE, the only other option is the PLL, which serves to modify the existing clock signals, thus adding to the power consumption.

Since clock accuracy is not important for the active cycle, the choice to use the MSI is easily made. However, determining the optimal frequency is not as straightforward. The datasheet gives a power consumption estimate of 120 µA/MHz , but the actual number is dependent on many different variables like temperature, enabled peripherals, low power modes, and much more. Therefore, the optimal frequency was determined experimentally.

Just like the temperature sensor comparison, the MCU was loaded with the final program, complete with low power modes. The current was measured using an oscilloscope, allowing calculation of the peak power draw, as well as the total energy used. These values were compared, giving more weight to a low energy consumption. Figure 3.7 shows the current graphs for all of the tested frequencies and table 3.5 shows the respective power and energy. Because Low Power Run mode is only available for clock speeds less than or equal to 2MHz, higher frequencies were not expected to manage a lower energy use. 4MHz was included in the comparison to verify this hypothesis. 100kHz was excluded from the figure due to a resulting execution time of 400ms, making the plot unreadable. The associated energy use was also multiple times more than the other frequencies, so 100kHz was dropped from consideration.

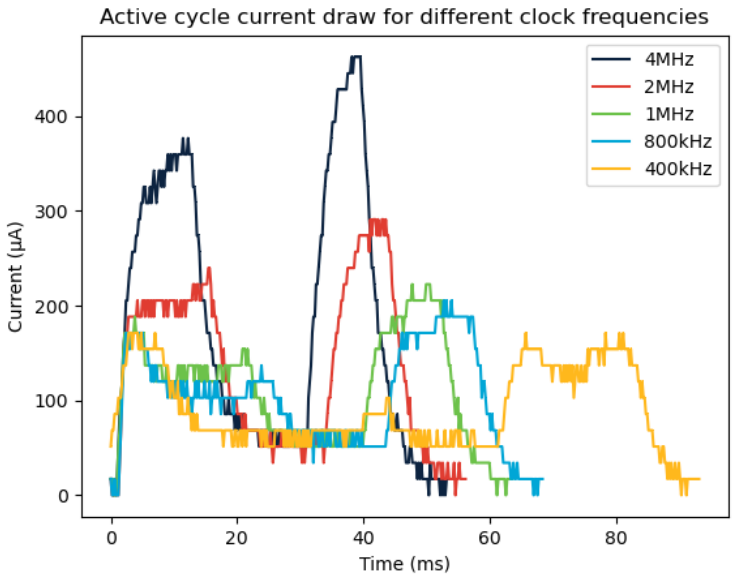


Figure 3.7: Combined plot of the current draw during an MCU active cycle for selected clock frequencies.

Table 3.5: Power and energy values for the different clock frequencies.

Frequency (MHz)	Peak power (µW)	Energy (µJ)
4	925	19.94
2	582	15.39
1	445	14.29
0.8	411	14.43
0.4	342	16.98
0.1	342	41.31

Figure 3.7 and table 3.5 show that the power decrease translates to an energy increase somewhere between 800 kHz and 400kHz. The least difference is between 1MHz and 800kHz, these are also the closest-spaced values. The 4MHz plot validates the earlier hypothesis with a drastically higher peak power and higher energy consumption than 2MHz. Overall, the energy difference between 800kHz and 1MHz is negligibly small, so 800kHz was chosen as the MCU clock frequency based on its lower peak power.

The clock choice for the RTC was done purely based on power, because both the LSE and LSI run at about 32kHz. The external oscillator offers an accuracy of 20 ppm, at the cost of 250-690 nA depending on drive strength. The LSI is less accurate, but offers a current consumption of max. 180 nA. Because there are no time accuracy requirements for this project, the LSI was chosen for its lower power draw.

3.4.3. Low power modes

The MCU features various modes of operation to further control its power consumption. Most of these suspend the program until an interrupt or event occurs, with the exception of Low Power Run mode. An overview of the used low power modes was already given in section 3.2.1, their usage is explained in section 3.3. Their impacts on power will be discussed here.

Starting with Run mode, this mode is the default mode after startup and after wake-up from the lowest-power modes. The MCU is powered through the main regulator (MR) and uses no power-saving techniques. The power consumption corresponds to the approximate 120 µA/MHz current consumption advertised in the datasheet.

Low Power Run mode allows the program to run while ensuring a lower power draw than Run mode. This is achieved by powering the MCU with the low power regulator, which has a lower current consumption than the MR. Furthermore, the CPU frequency is limited to 2MHz in this mode. The power draw corresponds to about 100 µA/MHz in this mode.

Stop 2 mode halts the program until an interrupt or an event occurs. It provides the lowest power consumption with SRAM retention, allowing it to resume the program from the point it halted after serving the interrupt. In Stop 2 mode, the MCU consumes about 2-5 µA . After waking up from this mode, the MCU is in Run mode, clocked by the MSI at the previously configured frequency. Thus, after wake-up, the MCU must manually enter Low Power Run mode as soon as possible.

Finally, Standby mode is used in the period between active cycles. In this mode, all SRAM contents are lost, so the program cannot use interrupt callbacks or resume from its previous state. Instead, when waking up from Standby mode, the MCU acts as if it was just powered on, executing the program from the beginning. Clocks and peripherals need to be reconfigured as only the backup registers retain their contents. In return, this mode offers the second-lowest power consumption. With the RTC active and clocked by the LSI, the current consumption is around 400nA.

3.4.4. GPIO power gating

Both the sensor and the memory have low-power modes to allow minimisation of their power consumptions when idle. Taking the worst case scenario, the sensor has a maximum shutdown-mode current of 8µA and the memory has a current of 1.5µA . Combined, this results in a shutdown mode current of 9.5µA , which is 23 times more than the Standby mode current consumption of the MCU. To eliminate this extra consumption, and to give the energy harvester as much surplus power as possible to charge the capacitor, the devices needed to be powered down completely. This was achieved by connecting the V_S pins of the components to GPIO pins of the MCU. With this technique, the components could be shut down fully, thereby eliminating their shutdown-mode consumption.

4

Testing and verification

4.1. Test setup

4.1.1. Current draw

To evaluate the system's power consumption and operational behaviour, current measurements were conducted using an oscilloscope in combination with a 500Ω resistor placed in series with the entire circuit. The voltage drop across this resistor was captured using a high-resolution oscilloscope, and instantaneous current was calculated using Ohm's law. The system was powered from a regulated 2.0V supply during all tests to reflect operating conditions.

The choice of a 500Ω resistor provided a good balance between measurement sensitivity and minimal impact on the circuit's operation. Although the setup enabled accurate current waveform capture, some measurement error remains due to noise and parasitic capacitance present in the measurement setup.

With the MCU, and thus the sensor, powered at 2V, the peak instantaneous power and total energy could be calculated using the Python libraries *numpy* and *scipy* [29][30], implementing the following equations:

$$p[n] = V \cdot i[n] \tag{4.1}$$

$$P_{max} = max(p[n]) \tag{4.2}$$

$$E = \int_{t_0}^{t_1} P(t)dt \tag{4.3}$$

Since $p[n]$ is an array of discrete values, equation 4.3 cannot be used directly. Therefore the scipy function *scipy.integrate.simpson()* was used to approximate the integral using Simpson's rule [31]. This approximation introduces some error, but for the purposes of the comparison negligible. The code used to process the measurement data can be found in appendix B.1.

4.1.2. Temperature Accuracy

Temperature accuracy tests were conducted in a controlled thermal environment. A calibrated, high-precision reference temperature sensor was placed in close proximity to the system to enable comparison. Temperature measurements were taken from 5°C to 65°C in 10°C increments. Between each temperature step, sufficient time was allowed for both the environment and the sensors to stabilize. The measurement data was collected in MATLAB and exported to Python for further processing. The Python code can be found in appendix B.4.

4.1.3. Memory Verification

Memory verification was performed by reading out the stored data using a NFC reader to ensure data integrity. Measurements were taken every 2 seconds in the same environment, causing the results of each measurement to be quite similar. Each measurement is stored as a 2-byte value. The first 8 bytes were intentionally left empty to meet requirements from the readout and display subgroup, however, this offset can easily be adjusted if needed.

4.2. Results

The power profile of the system during an active measurement cycle is shown in Figure 4.1. This active state includes wake-up, sensor readout, data storage, and transition back to standby.

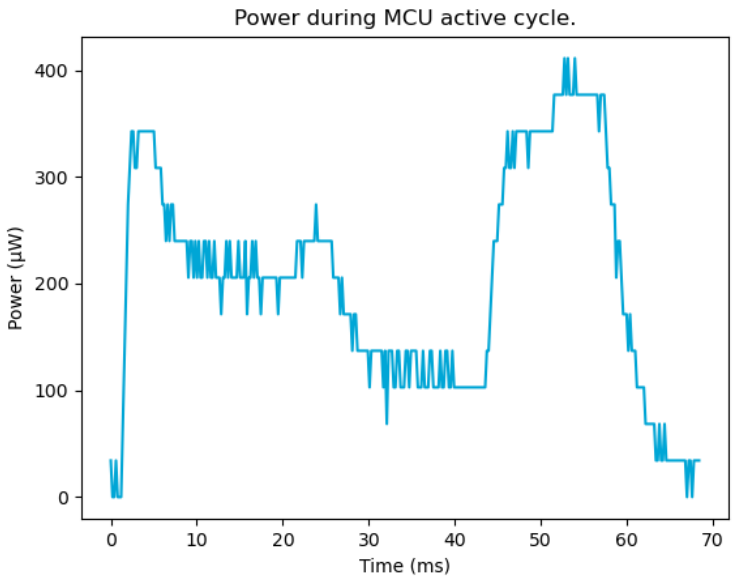


Figure 4.1: Power graph of the finished system. Operating voltage: 2V

The sensor measurements compared to a highly accurate reference temperature sensor across a range of temperatures (5°C - 65°C) are shown in figure 4.2. This comparison is further shown in figure 4.3, which is a plot of the difference between the two sensors.

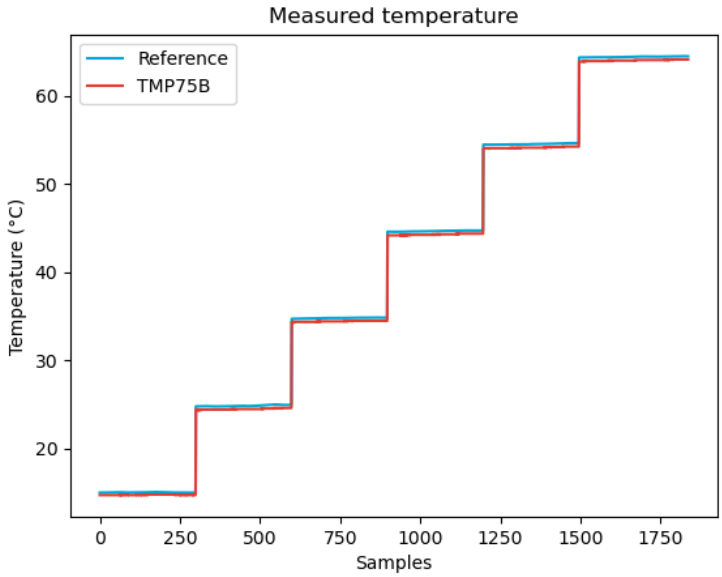


Figure 4.2: Temperature sensor comparison with ultra-accurate reference sensor.

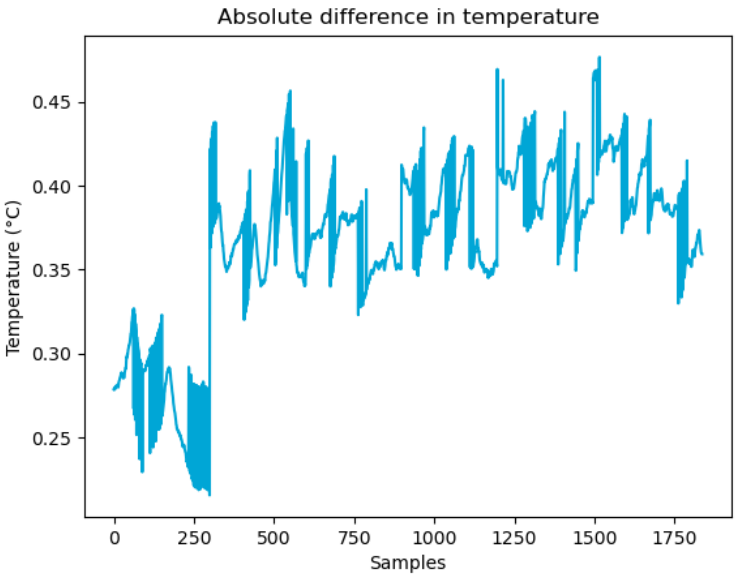


Figure 4.3: Temperature difference between TI75B and reference sensor.

Figure 4.4 presents the readout of the first 64 bytes of memory using NFC. Each pair of hexadecimal digits represents one byte, with 2 bytes being one measurements. The values shown are not related to the above accuracy test, they are meant to verify that the data is stored successfully in the external memory.

```
Addr   0: 00 00 00 00
Addr   4: 00 00 00 00
Addr   8: 01 9D 01 9D
Addr  12: 01 9C 01 9C
Addr  16: 01 9C 01 9C
Addr  20: 01 9C 01 9C
Addr  24: 01 9C 01 B0
Addr  28: 01 9C 01 9C
Addr  32: 01 9C 01 9C
Addr  36: 01 9B 01 9B
Addr  40: 01 9B 01 9B
Addr  44: 01 9B 01 9B
Addr  48: 01 9B 01 9B
Addr  52: 01 9B 01 9B
Addr  56: 01 9A 01 9A
Addr  60: 01 9A 01 9A
```

Figure 4.4: Memory readout of first 64 bytes after running the system.

Summarizing these results into the following key measurements:

- **Peak power:** 411 µW
- **Active cycle duration:** 70 ms
- **Energy usage per active cycle:** 14 µJ
- **Idle power consumption:** Less than 1µW
- **Sensor accuracy:** Within ±0.5°C

4.3. Discussion of results

The experimental results demonstrate that the system meets the core requirements laid out in the Programme of Requirements (PoR), particularly within the domains of power consumption, data integrity, and measurement performance.

4.3.1. Power

Great progress has been made in minimizing both the peak power consumption and the energy usage per measurement, in line with the requirements [3.3] and [3.4]. The system has a peak power draw of just 411µW during the active measurement cycle, which takes about 70 ms. For the full cycle, the total energy usage is around 14µJ per measurement. The idle power remains below 1µW , enabling significant power savings during periods without measurement. Although the sensor's peak power heavily exceeds the aspirational 100nW target ([3.1]), this trade-off was necessary to meet the practical constraints of using hand-solderable, commercially available components ([1.7], [2.5]).

4.3.2. System Operation

Operational stability was verified by consistent microcontroller behaviour at 800 kHz throughout the active cycle. Despite employing high-value pull-up resistors 47kΩ on the I²C lines, communication with both the temperature sensor and memory remained stable. This approach lowers peak power consumption while maintaining reliable operation and staying consistent with the system's 2V operating voltage constraint ([2.3]).

Temperature measurements and logging were performed every 2 seconds during testing, significantly exceeding the minimum requirement of one measurement per hour ([2.1]). This demonstrates that the system can operate at much higher logging frequencies than required, with the main limiting factor becoming available energy rather than system capability. Additionally, the system was able to fill the

entire memory without issue, confirming its ability to log sufficient data to meet the 4-week storage requirement ([2.2]).

The system was partially tested across the specified temperature range of -30°C to 50°C ([2.4]). As shown in figure 4.2, tests were performed from approximately 5°C up to 65°C, confirming stable operation and accurate temperature readings within this subrange. Additional testing at lower temperatures would be required to fully verify performance over the entire specified range.

4.3.3. Performance/memory

Regarding data storage, Figure 4.4 confirms correct operation of the non-volatile memory system. Each 2-byte value represents one temperature measurement. For instance, the first value starting at address 8, 0x019D, converts to a decimal value of 413. When multiplied by the resolution of 0.0625°C ([4.4] and [4.1]), this yields a temperature of approximately 25.8°C, which aligns with the ambient conditions during testing. This confirms that the system correctly stores and retrieves temperature data via NFC, fulfilling requirements [1.4] and [1.5]. The total available memory, defined by another subgroup, combined with the 2-byte measurement format used in this thesis, allows at least four weeks of hourly data logging. This satisfies the storage capacity requirement of exceeding 1344 bytes ([2.2] and [4.3]).

Validation of the temperature sensor accuracy ([4.2]) is shown in figure 4.2 and figure 4.3, which compares the system's sensor output to a reference sensor over a range of temperatures. The measured deviations remained easily within ±1°C across the tested range, confirming compliance with the specified accuracy.

5 Conclusion

5.1. Conclusion

This thesis presented the design and implementation of a batteryless temperature sensing and logging subsystem intended for use in a wireless and batteryless data logger in cold-chain logistics. The system was developed to operate reliably under extreme energy constraints, leveraging ultra-low-power components to autonomously sample and store temperature data for later retrieval.

The final prototype successfully met the key performance objectives. It achieved accurate temperature readings with deviations within $\pm 1°C$ when compared to a reference sensor and demonstrated successful logging of the temperature data to non-volatile memory. The system's peak power consumption during active operation was measured at 411 μW, with an active cycle duration of 70 ms. This resulted in an average energy use of 14 μJ for the full active cycle.

These results show that it is possible to design a low-power, autonomous logging subsystem using commercially available components, which is suitable for batteryless operation. Such a system can help reduce electronic waste by eliminating the need for disposable batteries in single-use temperature loggers. The design presented here therefore contributes toward more sustainable monitoring solutions for cold-chain logistics.

5.2. Recommendations

While the subsystem has demonstrated to work well within the intended scope, several areas offer opportunities for future improvement.

Firstly, it is important to emphasize that this design represents a working prototype using commercially available components. With the use of academic-level components, the efficiency of this subsystem can be increased significantly. As component technologies continue to advance, power consumption is expected to decrease while performance increases. This means that systems like the one presented here will naturally benefit from these advancements.

One other recommendation is the replacement of the development board microcontroller with a custom-designed PCB that implements all components. This would eliminate all unnecessary circuitry associated with the MCU's development board, reduce losses due to inefficient routing, and result in a more compact design.

Due to requirement [2.5] the options in component choices were limited. For upcoming projects, other options can be considered that are more power-efficient. It is advised to also look at options containing other communication protocols, like I³C [32], since I^2C plays a big role in the power consumption during the active cycle.

Validating the subsystem through real-world field testing in refrigerated transport environments would provide insight into its robustness and practical performance.

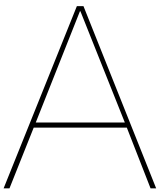
Small gains can still be made by exploring additional power-saving features of the microcontroller. While the prototype already uses several low-power techniques, it may be beneficial to investigate other features like Direct Memory Access (DMA), which could further reduce energy usage by offloading data transfers from the CPU.

Another consideration is the choice of microcontroller. While the current implementation uses a 32-bit MCU, this level of processing power is not strictly necessary for this type of application. Available 16-bit MCUs with development boards were not more power-efficient than the 32-bit option. However, without the overhead of a development board, a 16-bit MCU could be a more efficient alternative in future designs. 8-bit MCUs are generally even more power-efficient, but the onboard timers or RTCs often lack the bit depth needed to count to one hour. To make use of an 8-bit MCU, alternative strategies for hourly wake-up would be required, such as external timing components or shorter wake-up intervals. While the system is capable of operating at a higher measurement frequency, increasing the wake-up rate will also raise overall energy consumption. Therefore, careful balancing between timer capability and power budget is essential when exploring lower-bit microcontrollers.

Finally, extending the sensing capabilities beyond temperature to include parameters such as humidity or shock could make the system useful for a wider range of applications in logistics and environmental monitoring.

Through these enhancements, the batteryless logging system could evolve into a fully autonomous, multifunctional platform with significant impact on sustainable supply chain management.

Appendices



Appendix A: Firmware (C code)

The following code can also be found at the path: `Firmware(C)/Core/Src/main.c`, which is available under the following GitHub link: `https://github.com/Kdw123woee/Firmware-Datalogger-BEP-thesis-/blob/main/`. This GitHub repository also contains the other files associated with the project.

```
/* Includes
   ------------------------------------------------------------------*/
#include "main.h"
#include <stdio.h>

/* Defines
   ------------------------------------------------------------------*/
#define STANDBY_WAIT_TIME_MIN 59 //hourly measurement
#define STANDBY_WAIT_TIME_S 3  // Debug/demo purpose

/* Peripheral Handles
   ------------------------------------------------------------------*/
I2C_HandleTypeDef hi2c2;
I2C_HandleTypeDef hi2c3;
LPTIM_HandleTypeDef hlptim1;
RTC_HandleTypeDef hrtc;

/* Function Prototypes
   ------------------------------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C2_Init(void);
static void MX_RTC_Init(void);
static void MX_I2C3_Init(void);
static void MX_LPTIM1_Init(void);
void WriteToMem(I2C_HandleTypeDef *hi2c, uint16_t data);
static int16_t TI_ReadTemperature(I2C_HandleTypeDef *hi2c);

/* Read temperature from TI sensor */
static int16_t TI_ReadTemperature(I2C_HandleTypeDef *hi2c) {
    const uint8_t TEMP_SENSOR_ADDR = 0x9E; // i2c device address
    const uint8_t TEMP_REG_ADDR = 0x00;
```

```
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_5, GPIO_PIN_SET);  // Sensor power
        ON, sensor automatically starts measurement

    //15ms stop2 mode during wait for sensor measurement
    HAL_LPTIM_TimeOut_Start_IT(&hlptim1, 15);
    __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
    HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);
    HAL_LPTIM_TimeOut_Stop_IT(&hlptim1);

    HAL_PWREx_EnableUltraLowPowerMode();
    HAL_PWREx_EnableLowPowerRunMode(); //re-enable low power run mode

    //receive data from sensor register
    uint8_t reg = TEMP_REG_ADDR;
    uint8_t temp_raw[2];
    if (HAL_I2C_Master_Transmit(hi2c, TEMP_SENSOR_ADDR, &reg, 1,
        HAL_MAX_DELAY) == HAL_OK &&
        HAL_I2C_Master_Receive(hi2c, TEMP_SENSOR_ADDR, temp_raw, 2,
            HAL_MAX_DELAY) == HAL_OK) {
        int16_t temp = (int16_t)((temp_raw[0] << 8) | temp_raw[1]);
        temp >>= 4;  // 12-bit data, right-justified
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_5, GPIO_PIN_RESET);  // Sensor
            power OFF
        return temp;
    } else {
        return 0;
    }
}

/* Write temperature data to external EEPROM memory */
void WriteToMem(I2C_HandleTypeDef *hi2c, uint16_t data) {
    const uint8_t devaddr = 0x53 << 1; //i2c device address
    const uint16_t BASE_ADDRESS = 0x0008; //optional offset of 8 bytes for
        subgroup 3 purposes, else 0x0000

    uint16_t last_addr = READ_REG(TAMP->BKP0R) & 0xFFFF; //backup register
        which saves last written address
    //if last address = 0, start writing from byte 8
    uint16_t next_addr = (last_addr == 0) ? BASE_ADDRESS : last_addr + 2;

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);  // Memory power
        ON

    // wait until the memory is ready
    while (HAL_I2C_IsDeviceReady(hi2c, devaddr, 5, HAL_MAX_DELAY) !=
        HAL_OK) {}

    //write 2 data bytes
    uint8_t data_buf[2] = {(uint8_t)(data >> 8), (uint8_t)(data & 0xFF)};
    if (HAL_I2C_Mem_Write(hi2c, devaddr, next_addr, I2C_MEMADD_SIZE_16BIT,
        data_buf, 2, HAL_MAX_DELAY) != HAL_OK) {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);  // Memory
            power OFF
        return;
    }
```

```
    while (HAL_I2C_IsDeviceReady(hi2c, devaddr, 5, HAL_MAX_DELAY) !=
        HAL_OK) {}

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);  // Memory power
        OFF
    WRITE_REG(TAMP->BKP0R, next_addr); //store MSB address
}

/* Main program */
int main(void) {
    HAL_Init();
    SystemClock_Config();

    HAL_PWREx_EnableUltraLowPowerMode();
    HAL_PWREx_EnableLowPowerRunMode();

    MX_GPIO_Init();
    MX_I2C2_Init();
    MX_RTC_Init();
    MX_I2C3_Init();
    MX_LPTIM1_Init();

    if (__HAL_PWR_GET_FLAG(PWR_FLAG_SB) != RESET) {  // Wake from standby
        __HAL_PWR_CLEAR_FLAG(PWR_FLAG_SB);
        __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
        __HAL_RTC_WAKEUPTIMER_CLEAR_FLAG(&hrtc, RTC_FLAG_WUTF);
        HAL_RTCEx_DeactivateWakeUpTimer(&hrtc);

        int16_t temperaturedata = TI_ReadTemperature(&hi2c2);
        WriteToMem(&hi2c3, temperaturedata);

    } else { // first startup
        WRITE_REG(TAMP->BKP0R, 0); //initial startup set backup register (
            used for last address)
    }

    HAL_NVIC_ClearPendingIRQ(RTC_TAMP_IRQn); //clear interrupt from wake-
        up timer

    //set RTC timer for wake up from standby
    if (HAL_RTCEx_SetWakeUpTimer_IT(&hrtc, STANDBY_WAIT_TIME_MIN,
        RTC_WAKEUPCLOCK_CK_SPRE_16BITS, 0) != HAL_OK) {
        Error_Handler();
    }

    HAL_PWR_EnterSTANDBYMode(); //enter standby mode
}

/* System Clock Configuration */
void SystemClock_Config(void) {
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE2);
```

```
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI |
       RCC_OSCILLATORTYPE_MSI;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;
    RCC_OscInitStruct.MSIState = RCC_MSI_ON;
    RCC_OscInitStruct.MSICalibrationValue = RCC_MSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_3; //800kHz
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;

    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
        Error_Handler();
    }

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
       RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK
       ) {
        Error_Handler();
    }
}

/* I2C2 Initialization */
static void MX_I2C2_Init(void) {
    hi2c2.Instance = I2C2;
    hi2c2.Init.Timing = 0x00000000;
    hi2c2.Init.OwnAddress1 = 0;
    hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    hi2c2.Init.OwnAddress2 = 0;
    hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
    hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;

    if (HAL_I2C_Init(&hi2c2) != HAL_OK) {
        Error_Handler();
    }
    if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) !=
       HAL_OK) {
        Error_Handler();
    }
    if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK) {
        Error_Handler();
    }
    if (HAL_I2CEx_ConfigFastModePlus(&hi2c2, I2C_FASTMODEPLUS_ENABLE) !=
       HAL_OK) {
        Error_Handler();
    }
}

/* I2C3 Initialization */
static void MX_I2C3_Init(void) {
    hi2c3.Instance = I2C3;
    hi2c3.Init.Timing = 0x00000000;
```

```
    hi2c3.Init.OwnAddress1 = 0;
    hi2c3.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hi2c3.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    hi2c3.Init.OwnAddress2 = 0;
    hi2c3.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
    hi2c3.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    hi2c3.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;

    if (HAL_I2C_Init(&hi2c3) != HAL_OK) {
        Error_Handler();
    }
    if (HAL_I2CEx_ConfigAnalogFilter(&hi2c3, I2C_ANALOGFILTER_ENABLE) !=
       HAL_OK) {
        Error_Handler();
    }
    if (HAL_I2CEx_ConfigDigitalFilter(&hi2c3, 0) != HAL_OK) {
        Error_Handler();
    }
    if (HAL_I2CEx_ConfigFastModePlus(&hi2c3, I2C_FASTMODEPLUS_ENABLE) !=
       HAL_OK) {
        Error_Handler();
    }
}

/* LPTIM1 Initialization */
static void MX_LPTIM1_Init(void) {
    hlptim1.Instance = LPTIM1;
    hlptim1.Init.Clock.Source = LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC;
    hlptim1.Init.Clock.Prescaler = LPTIM_PRESCALER_DIV32;
    hlptim1.Init.Trigger.Source = LPTIM_TRIGSOURCE_SOFTWARE;
    hlptim1.Init.Period = 20;
    hlptim1.Init.UpdateMode = LPTIM_UPDATE_IMMEDIATE;
    hlptim1.Init.CounterSource = LPTIM_COUNTERSOURCE_INTERNAL;
    hlptim1.Init.Input1Source = LPTIM_INPUT1SOURCE_GPIO;
    hlptim1.Init.Input2Source = LPTIM_INPUT2SOURCE_GPIO;
    hlptim1.Init.RepetitionCounter = 0;

    if (HAL_LPTIM_Init(&hlptim1) != HAL_OK) {
        Error_Handler();
    }
    HAL_LPTIM_TimeOut_Stop_IT(&hlptim1);
}

/* RTC Initialization */
static void MX_RTC_Init(void) {
    hrtc.Instance = RTC;
    hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
    hrtc.Init.AsynchPrediv = 127;
    hrtc.Init.SynchPrediv = 255;
    hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
    hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
    hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
    hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;

    if (HAL_RTC_Init(&hrtc) != HAL_OK) {
        Error_Handler();
```

```
    }
}

/* GPIO Initialization */
static void MX_GPIO_Init(void) {
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    GPIO_InitTypeDef GPIO_InitStruct = {0};

    GPIO_InitStruct.Pin = GPIO_PIN_5;  // Sensor power control
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_1;  // Memory power control
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

/* Error Handler */
void Error_Handler(void) {
    __disable_irq();
    while (1) {}
}
```

B

Appendix B: Testing Software (Python)

The following codes can also be found at the path `Testing Code (Python)/`, which is available under the following GitHub link:
`https://github.com/Kdw123woee/Firmware-Datalogger-BEP-thesis-/blob/main/`.

B.1. Current measurement for power and energy calculations

```
import matplotlib.pyplot as plt
import pandas as pd

import numpy as np
import scipy.integrate as si

mu = chr(956)
Ohm = chr(937)


def convertdata(data):
    # extract voltages and time points from csv
    volts = np.asarray(data.iloc[:,3].values)
    time = np.asarray(data.iloc[:,2].values)
    # read sample interval from csv
    sample_interval = float(data.iloc[0,:].values[1])
    R = 467
    amps = volts / R  # amps in A
    amps *= 1e6  # amps in uA

    # uses sample interval to convert linspace
    # to actual timestamps starting at 0ms

    # very rudimentary way to find the start:
    # find the first value where amps > 120 (arbitrary high enough)
    # find the last value where amps > 120
    # apply some offset
    a = np.where(amps > 120)[0][0]
    b = np.where(amps > 120)[0][-1]
    amps = amps[a-10:b+50]
    time = np.linspace(0, len(amps) * sample_interval, len(amps))
```

```
    time *= 1000  # convert to ms

    return amps, time


def calc_energy(amps, time):
    watts = amps * 2
    energy = si.simpson(y=watts, x=time/1000)
    return watts, energy


if __name__ == "__main__":
    # read oscilloscope csv into pandas databuffer
    data1 = pd.read_csv("4MHz_47k.csv", usecols=[0,1,3,4])
    data2 = pd.read_csv('2MHz_47k.CSV', usecols=[0,1,3,4])
    data3 = pd.read_csv("1MHz_47k.CSV", usecols=[0, 1, 3, 4])
    data4 = pd.read_csv('800kHz_47k.CSV', usecols=[0, 1, 3, 4])
    data5 = pd.read_csv("400kHz_47k.CSV", usecols=[0,1,3,4])

    data1_name = "4MHz"
    data2_name = "2MHz"
    data3_name = "1MHz"
    data4_name = "800kHz"
    data5_name = "400kHz"
    data6_name = "100kHz"

    # convert voltage/time data to current/time,
    # normalise start of active cycle at 0ms
    amps1, time1 = convertdata(data1)
    amps2, time2 = convertdata(data2)
    amps3, time3 = convertdata(data3)
    amps4, time4 = convertdata(data4)
    amps5, time5 = convertdata(data5)

    # calculate the power at every time point
    # and the total energy over the active cycle
    watts1, energy1 = calc_energy(amps1, time1)
    watts2, energy2 = calc_energy(amps2, time2)
    watts3, energy3 = calc_energy(amps3, time3)
    watts4, energy4 = calc_energy(amps4, time4)
    watts5, energy5 = calc_energy(amps5, time5)

    # can be made prettier but does the job just fine
    print(f'energy, peak power {data1_name} : {energy1} ,  {max(watts1)}')
    print(f'energy, peak power {data2_name} : {energy2} ,  {max(watts2)}')
    print(f'energy, peak power {data3_name} : {energy3} ,  {max(watts3)}')
    print(f'energy, peak power {data4_name} : {energy4} ,  {max(watts4)}')
    print(f'energy, peak power {data5_name} : {energy5} ,  {max(watts5)}')


    # plot all graphs at once
    plt.figure(1)
    plt.plot(time1, amps1, color='#0C2340', marker=',')
    plt.plot(time2, amps2, color='#E03C31', marker=',')
    plt.plot(time3, amps3, color='#6CC24A', marker=',')
    plt.plot(time4, amps4, color='#00A6D6', marker=',')
```

```
    plt.plot(time5, amps5, color='#FFB81C', marker=',')

    plt.ylabel(f"Current ({mu}A)")
    plt.xlabel("Time (ms)")
    plt.title("Active cycle current draw for different clock frequencies")
    plt.legend(["4MHz", "2MHz","1MHz","800kHz", "400kHz"])

    # plot power of 800kHz, 2V
    plt.figure(3)
    plt.plot(time4, watts4, color='#00A6D6')
    plt.ylabel(f"Power ({mu}W)")
    plt.xlabel("Time (ms)")
    plt.title("Power during MCU active cycle.")

    plt.show()
```

B.2. Temperature sensor comparison

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.integrate as si


mu = chr(956)
deg = chr(176)

def convertdata(data):
    # extract voltages and time points from csv
    volts = np.asarray(data.iloc[:,3].values)
    time = np.asarray(data.iloc[:,2].values)
    # read sample interval from csv
    sample_interval = float(data.iloc[0,:].values[1])
    R = 467
    amps = volts / R  # amps in A
    amps *= 1000000  #  amps in uA

    # uses sample interval to convert linspace
    # to actual timestamps starting at 0ms
    # very rudimentary way to find the start:
    # find the first value where amps > 100 (arbitrary high enough)
    # find the last value where amps > 100
    # apply some offset
    a = np.where(amps > 80)[0][0]
    b = np.where(amps > 80)[0][-1]
    amps = amps[a-10:b+50]
    time = np.linspace(0, len(amps) * sample_interval, len(amps))
    time *= 1000 # convert to ms
    return amps, time


def calc_energy(amps, time):
    watts = amps * 2
    energy = si.simpson(y=watts, x=time/1000)
    return watts, energy
```

```
if __name__ == "__main__":
    # read oscilloscope csv into pandas databuffer
    data_AT = pd.read_csv('AT_47k.CSV', usecols=[0,1,3,4])
    data_TI = pd.read_csv('TI_47k.CSV', usecols=[0,1,3,4])

    # convert voltage/time data to current/time,
    # normalise start of active cycle at 0ms
    amps_AT, time_AT = convertdata(data_AT)
    watts_AT, energy_AT = calc_energy(amps_AT, time_AT)
    print('energy AT: ', energy_AT)
    print("max power AT: ", max(watts_AT))

    amps_TI, time_TI = convertdata(data_TI)
    watts_TI, energy_TI = calc_energy(amps_TI, time_TI)
    print(" ")
    print("energy TI: ", energy_TI)
    print("max power TI: ", max(watts_TI))

    # plot both sensor graphs
    plt.figure(1)
    plt.plot(time_AT, amps_AT, color='#00A6D6')
    plt.plot(time_TI, amps_TI, color='#E03C31')
    plt.ylabel(f"Current ({mu}A)")
    plt.xlabel("Time (ms)")
    plt.title("Active cycle current draw for different sensors")
    plt.legend(["AT", "TI"])
    plt.show()
```

B.3. I²C Pull-up resistor current model

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

mu = chr(956)
Ohm = chr(937)
approx = chr(8773)

if __name__ == '__main__':
    V = 2
    R = np.arange(4, 200, step=1)  # R in kOhm
    I = V / R  # I in mA
    I *= 1000  # I in uA

    I_diff = np.gradient(I, R)
    print(I_diff[40:50])
    print(R[40])

    # tangent line for visualisation
    x = np.arange(15, 75, step=1)
    y = (-1 * x) + 89

    plt.figure(1)
    plt.plot(R, I, color="#00A6D6")
    plt.plot(x, y, color='#E03C31')
```

```
    plt.ylabel(f"Current ({mu}A)")
    plt.xlabel(f"Resistance (k{Ohm})")
    plt.title("Expected current at 2V for different pull-up values")
    plt.legend(["Current", f"Crossover point, dx/dy {approx} -1"])
    plt.show()
```

B.4. Temperature sensor accuracy

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.io as io

deg = chr(176)


if __name__ == "__main__":

    ref1 = io.loadmat("Ref_data.mat")
    ref2 = io.loadmat("T2.mat")
    ti1 = io.loadmat("TI_data.mat")
    ti2 = io.loadmat("num2.mat")

    reftemp1 = np.asarray(ref1['T'][0])
    reftemp2 = np.asarray(ref2['T'][0])[1400:]
    titemp1 = np.asarray(ti1['num'][0])
    titemp2 = np.asarray(ti2['num'][0])[1400:]
    combinedref = np.hstack((reftemp1, reftemp2))
    combinedti = np.hstack((titemp1, titemp2))

    # delete samples to only keep the ones when oven stable
    samples_to_keep = 400
    samples_to_delete = 1800 - samples_to_keep + 1
    sample_delete = np.arange(0, samples_to_delete, 1)
    for i in range(5):
        i += 1
        index = 1800*i
        sample_delete = np.append(sample_delete,
                                  np.arange(index, index+samples_to_delete, 1))

    combinedref = np.delete(combinedref, sample_delete)
    combinedti = np.delete(combinedti, sample_delete)

    sample_delete = np.arange(300, 401, 1)
    for i in range(5):
        i += 1
        index = 400 * i
        sample_delete = np.append(sample_delete,
                                  np.arange(index-100, index+1, 1))

    combinedref = np.delete(combinedref, sample_delete)
    combinedti = np.delete(combinedti, sample_delete)

    # calculate difference and statistics
    diff = abs(combinedti - combinedref)
    mean_diff = np.mean(diff)
```

```
std_diff = np.std(diff)
max_diff = np.max(diff)
min_diff = np.min(diff)

# display nicely in pd dataframe
a = np.column_stack((mean_diff, std_diff, max_diff, min_diff))
df = pd.DataFrame(a, columns=['mean diff', 'std',
                              'max diff', 'min diff'])
print(df.to_string(index=False))

plt.figure(1)
plt.plot(combinedref, color="#00A6D6")
plt.plot(combinedti, color='#E03C31')
plt.ylabel(f"Temperature ({deg}C)")
plt.xlabel("Samples")
plt.title("Measured temperature")
plt.legend(["Reference", "TMP75B"])

plt.figure(2)
plt.plot(diff, color='#00A6D6')
plt.ylabel(f"Temperature ({deg}C)")
plt.xlabel("Samples")
plt.title("Absolute difference in temperature")

plt.show()
```

Appendix C: Device technical information

C.1. STM32U083C-DK board and layout

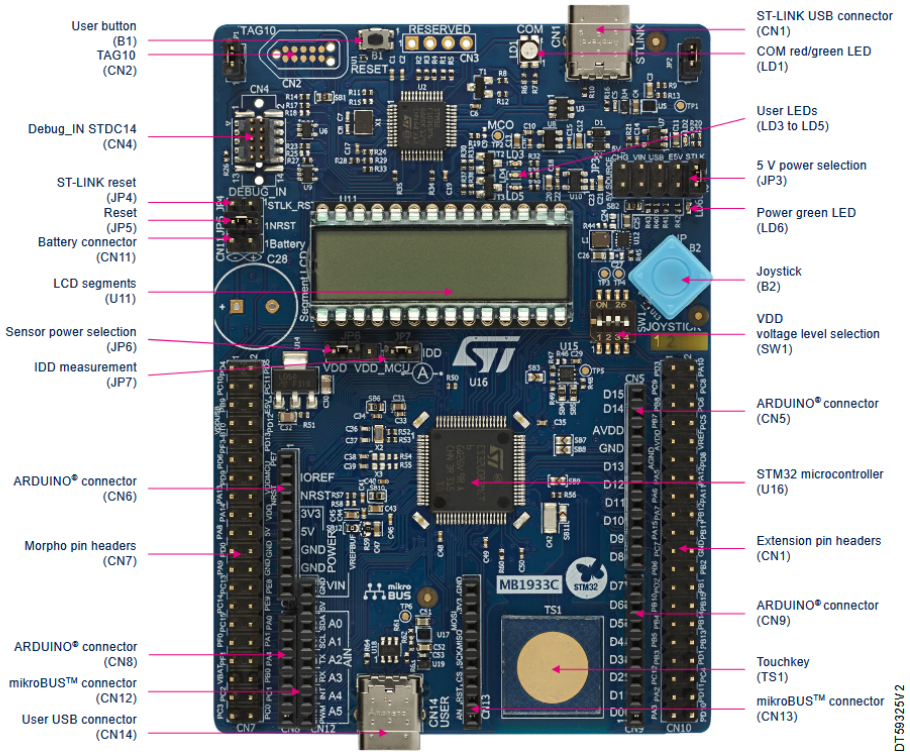


Figure C.1: Annotated top view of the development board, from [25]

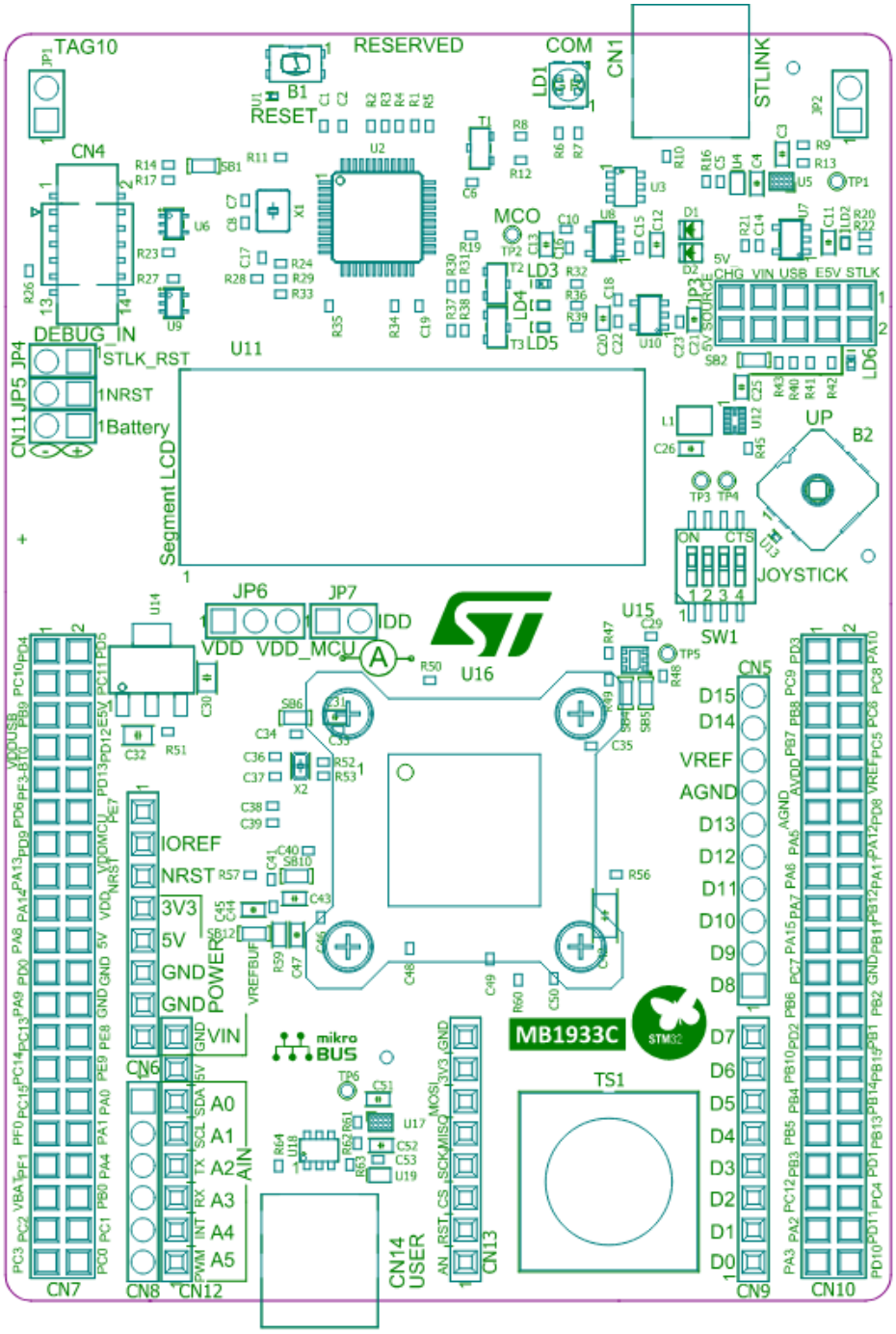


Figure C.2: Development board layout top view, from [33]

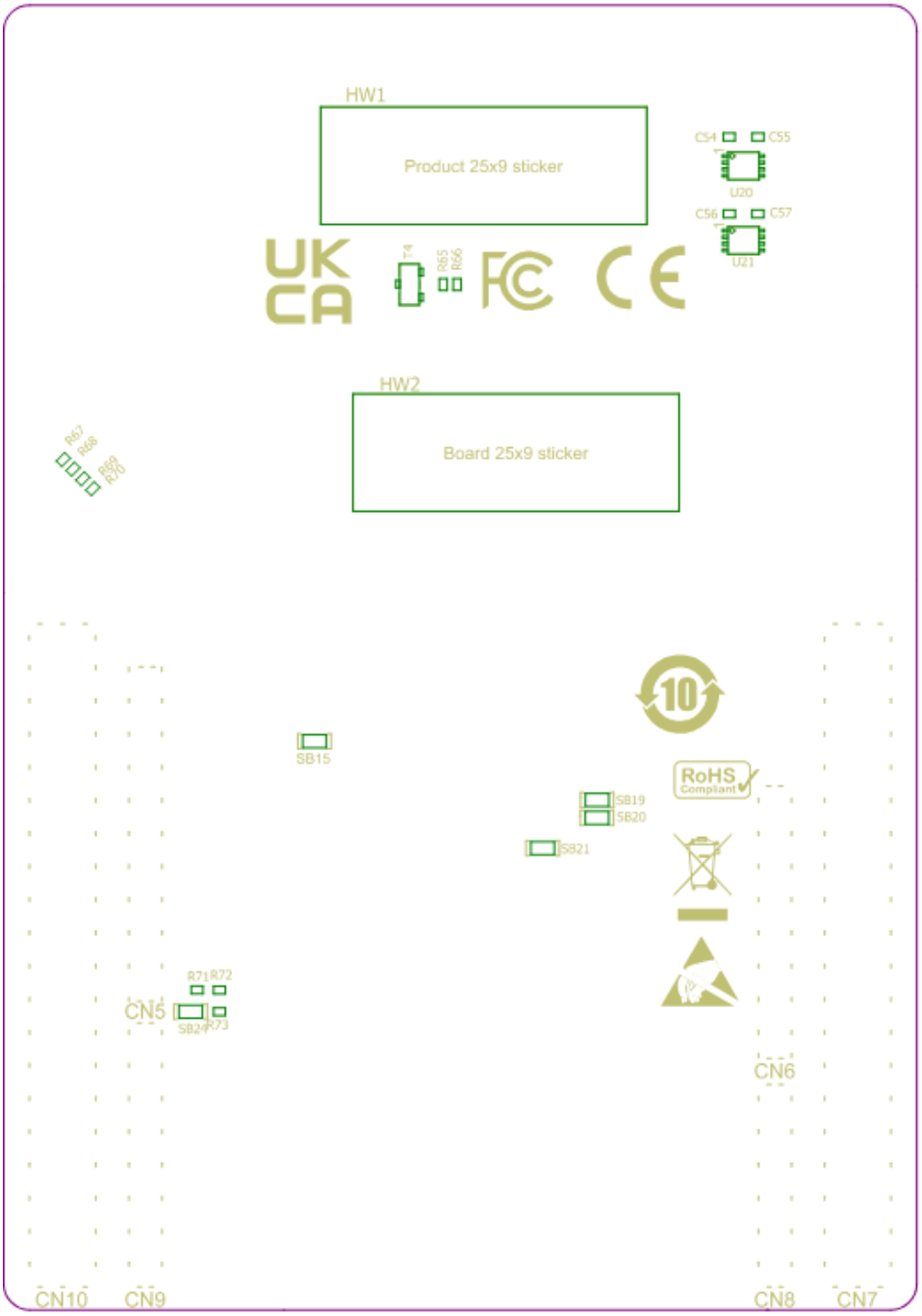


Figure C.3: Development board layout bottom view, from [33]

C.2. TMP75B functional block diagram

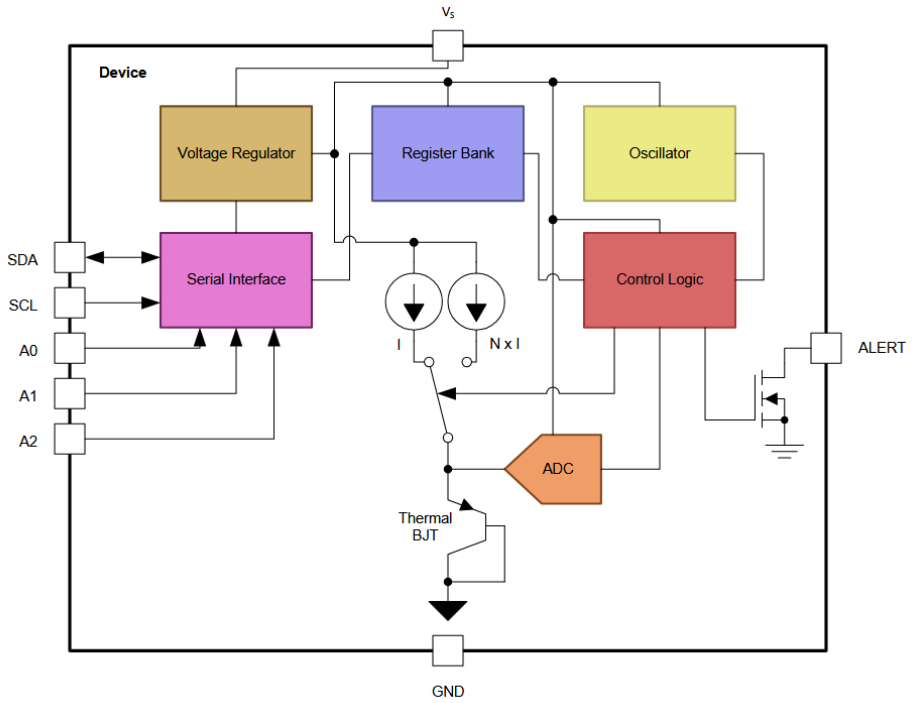


Figure C.4: Functional block diagram of the TMP75B, from [13].

Bibliography

[1] "Post-harvest technologies and cold chain management: A review," *Plant Archives*, vol. 25, no. 1, pp. 3275–3283, Oct. 2024. DOI: 10.51470/plantarchives.2025.v25.supplement-1.447. [Online]. Available: https://doi.org/10.51470/plantarchives.2025.v25.supplement-1.447.

[2] R. Badia-Melis, U. McCarthy, L. Ruiz-Garcia, J. Garcia-Hierro, and J. I. Robla Villalba, "New trends in cold chain monitoring applications – a review," *Food Control*, vol. 86, pp. 170–182, 2018, ISSN: 0956-7135. DOI: 10.1016/j.foodcont.2017.11.022.

[3] "Bananas." *cargohandbook.com*. (n.d.), [Online]. Available: https://www.cargohandbook.com/Bananas (visited on 10/05/2025).

[4] S. Ahmad, A. K. Thompson, I. A. Hafiz, and A. A. Asi, "Effect of temperature on the ripening behavior and quality of banana fruit," *International Journal of Agriculture and Biology*, vol. 3, pp. 224–227, 2 2001, ISSN: 1560-8530.

[5] United Nations Institue for Training and Research. "Global e-waste monitor 2024: Electronic waste rising five times faster than documented e-waste recycling," *United Nations*. (2024), [Online]. Available: https://unitar.org/about/news-stories/press/global-e-waste-monitor-2024-electronic-waste-rising-five-times-faster-documented-e-waste-recycling (visited on 06/12/2025).

[6] International Telecommunication Union. "Electronic waste rising five times faster than documented e-waste recycling: UN," *International Telecommunication Union*. (2024), [Online]. Available: https://www.itu.int/en/mediacentre/Pages/PR-2024-03-20-e-waste-recycling.aspx (visited on 06/12/2025).

[7] "The ultimate guide to cold chain temperature monitoring," *sensitech.com*. (n.d.), [Online]. Available: https://www.sensitech.com/en/blog/blog-articles/blog-ultimate-guide-cold-chain-monitoring.html (visited on 05/15/2025).

[8] *Usric-8: Usb pdf temperature recorder product user guide*, LogTag Recorders, 2016. [Online]. Available: https://logtagrecorders.com/product/usric-8/.

[9] "Facility and equipment monitoring service for visibility and compliance." *shareddocs.com*. (2004), [Online]. Available: https://www.shareddocs.com/hvac/docs/2004/Public/05/Coldstream-site-food.pdf (visited on 05/15/2025).

[10] Farnell, "Multi-use compact pdf temperature and humidity usb data logger specifications and selection guide," n.d. [Online]. Available: https://www.farnell.com/datasheets/3164608.pdf.

[11] N. Khalid, R. Mirzavand, and A. K. Iyer, "A survey on battery-less RFID-based wireless sensors," *Micromachines*, vol. 12, no. 7, 2021, ISSN: 2072-666X. DOI: 10.3390/mi12070819. [Online]. Available: https://www.mdpi.com/2072-666X/12/7/819.

[12] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith, "Design of an RFID-based battery-free programmable sensing platform," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 11, pp. 2608–2615, Nov. 2008, ISSN: 0018-9456. DOI: 10.1109/TIM.2008.925019.

[13] Texas Instruments, "TMP75B 1.8-V Digital Temperature Sensor with Two-Wire Interface and Alert," SBOS706B, Rev. B, 2014. [Online]. Available: https://www.ti.com/lit/gpn/tmp75b.

[14] Microchip Technology, “9- to 12-bit Selectable, ±0.5°C Accurate Digital Temperature Sensor,” 8839I-DTS, Rev. I, 2014. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8839-DTS-AT30TS75A-Datasheet.pdf.

[15] K. A. A. Makinwa, “Smart temperature sensor survey.” [Online]. Available: http://ei.ewi.tudelft.nl/docs/TSensor_survey.xls.

[16] R. Russel and W. Ibrahimi, “A batteryless inductive energy harvesting system,” B.S. thesis, Delft University of Technology, Delft, The Netherlands, 2025.

[17] B. Joemmankhan and S. Dinev, “Wireless RF transmission for a battery less temperature logger,” B.S. thesis, Delft University of Technology, Delft, The Netherlands, 2025.

[18] United Nations Economic Commission for Europe, *Agreement on the international carriage of perishable foodstuffs and on the special equipment to be used for such carriage (ATP)*, 2022. [Online]. Available: https://unece.org/transport/road-transport/text-and-status-agreement.

[19] Y. K. Tan, *Energy Harvesting Autonomous Sensor Systems: Design, Analysis, and Practical Implementation*. CRC Press, 2013.

[20] F. et al., “A low-power RFID enabled temperature sensor for cold chain management,” In Proc. 2015 IEEE International Symposium on Circuits and Systems (ISCAS), May 2015. DOI: 10.1109/iscas.2015.7169096. [Online]. Available: https://doi.org/10.1109/iscas.2015.7169096.

[21] J. dos Santos, F. Gomes, M. A. ds Santos, *et al.*, “Optimized ultra-low power sensor-enabled rfid data logger for pharmaceutical cold chain,” In Proc. 2015 IEEE Brasil RFID, 2015, pp. 1–5. DOI: 10.1109/BrasilRFID.2015.7523835.

[22] H.-J. Kim, H.-S. Song, K.-C. Lee, J.-W. Yu, D.-H. Lee, and M. Han, “Wireless temperature monitoring with shape memory alloy-based antenna,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4680–4688, 2020, ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3029206. [Online]. Available: https://ieeexplore.ieee.org/document/9312399.

[23] S. Lee, Y. Song, and H. Choi, “Wireless sensor network design for tactical military applications: Remote large-scale environments,” In Proc. 2013 International Conference on ICT Convergence (ICTC), Jeju Island, South Korea, Oct. 2013, pp. 366–370. [Online]. Available: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=6676819.

[24] STMicroelectronics, “Ultra-low-power Arm® Cortex®-M0+ 32-bit MCU, 256-Kbyte flash memory, 40- Kbyte SRAM, USB, LCD, AES,” DS14463, Rev. 2, 2024. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32u083mc.html.

[25] STMicroelectronics, “Discovery kit with STM32U083MC MCU,” UM3292, Rev. 2, 2025. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32u083c-dk.html.

[26] “STM32U0: Up to 50% energy saving, the new benchmark in entry-level ultra-low power MCUs #STM32Summit,” *st.com*. (2024), [Online]. Available: https://blog.st.com/stm32u0 (visited on 06/14/2025).

[27] STMicroelectronics, “Dynamic NFC/RFID tag IC with 4-, 16-, or 64-Kbit EEPROM, fast transfer mode capability, and optimized I2C,” DS13519, Rev. 8, 2024. [Online]. Available: https://www.st.com/en/nfc/st25dv64kc.html.

[28] NXP Semiconductors, “I2C-bus specification and user manual,” UM10204, Rev. 7, 2021. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[29] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[30] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[31] K. Cartwright, “Simpson’s rule cumulative integration with ms excel and irregularly-spaced data,” *Journal of Mathematical Science and Mathematics Education*, vol. 12, Sep. 2017.

[32] M. Cole, MIPI Alliance, “Introduction to the MIPI I3C Standardized Sensor Interface,” White Paper, Aug. 2016.

[33] STMicroelectronics, “STM32U083C-DK Board Schematic,” MB1933, Rev. C-02, 2024. [Online]. Available: `https://www.st.com/resource/en/schematic_pack/mb1933-u083c-c02-schematic.pdf`.