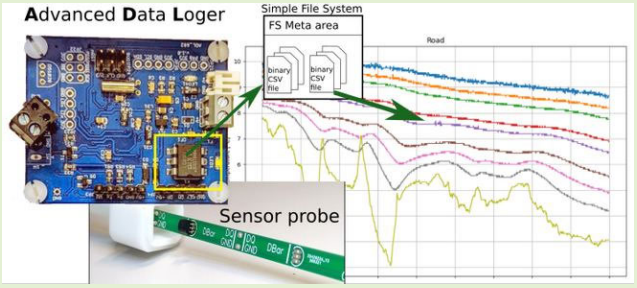
A Low-Power Data Logger With Simple File System for Long-Term Environmental Monitoring in Remote Areas

Juraj Ďud'ák, Gabriel Gašpar, Roman Budjač, Ivan Sládek, and Peter Husár

***Abstract*—This research addresses the long-term measurement of environmental data in geographically remote areas and an energy-optimized method of storing data on a storage medium. For this purpose, we have developed our measurement module Advanced Data Logger (ADL). In terms of connectivity, the module operates in three modes: 1) offline—when measured data is primarily stored on the storage medium; 2) Internet of Things (IoT) ready—measured data is stored on the storage medium and sent to the remote server in defined batches; and 3) online mode—when measured data is preferably sent to the remote server immediately after measurement. The design aims to minimize the module's power consumption so that the autonomous operating time is close to one year. As part of the design, the Simple File System (simpleFS) software module is designed for the role of a simple file system (FS) optimized to minimize I/O operations. Its other feature in data storage is the automatic normalization of the data transmitted from the attached sensors. The last part of the design is the AdlReader software solution, used to configure the hardware (HW) module and to retrieve the measured data files. We verified the correct operation of the ADL module along with nine sensors built in a vertical soil temperature profile probe in experimental installation and operation for two months. According to the requirements for our solution, the expected operation time of the ADL module is 9–12 months.**



***Index Terms*— Data logger, energy consumption, file system (FS), memory, microcontroller, optimization, sensor.**

I. Introduction

OVER the years, Internet of Things (IoT) devices have established their place in sensing applications, mainly due to their ability to be implemented as compact devices that can perform a task and be connected to the Internet or a local area network [1]. IoT devices and especially in the form of Industrial IoT are a vital part of the digital transformation in Industry 4.0 [2]. The contribution of IoT devices to the Industry 4.0 concept is mainly assumed by sensing, collecting, and sending data among themselves or to a higher-level system in a vertical control structure. From our point of view, interesting situations may occur while using IoT devices in rural areas. These are devices in forests, mines, and remote farms to monitor air quality, soil composition, moisture measurement, and further processing of the collected data [3]. In this case, power consumption is a critical concern. It is also important to mention that wireless connection significantly impacts device power consumption [4]. Typically, the power supply for IoT devices is realized using solar panels and accumulators. Consequently, in such a case, it is necessary to provide a mechanism for collecting the measured data, and with this comes the necessity to address the power consumption efficiency of the individual modules of the off-grid device and data processing.

The issue of optimizing the power consumption of data acquisition equipment in Smart OffGrid IoT Deployments is still an open question, and contributions can be made to this

Manuscript received 10 October 2023; revised 25 October 2023; accepted 25 October 2023. Date of publication 13 November 2023; date of current version 14 December 2023. This work was supported by the European Regional Development Fund under the project of Operational Program Integrated Infrastructure: Support of research and development capacities in the field of generating advanced software tools designed to increase the resistance of economic entities against excessive volatility of the energy commodity market under Grant ITMS2014+ code 313011BUK9. The associate editor coordinating the review of this article and approving it for publication was Prof. Nitaigour P. Mahalik. *(Corresponding author: Juraj Ďud'ák.)*

Juraj Ďud'ák and Gabriel Gašpar are with the Faculty of Materials Science and Technology, Slovak University of Technology, 917 24 Trnava, Slovakia, and also with the Research Centre, University of Žilina, 010 26 Žilina, Slovakia (e-mail: juraj.dudak@stuba.sk; gabriel.gaspar@uniza.sk).

Roman Budjač and Ivan Sládek are with the Research Centre, University of Žilina, 010 26 Žilina, Slovakia (e-mail: roman.budjac@uniza.sk; ivan.sladek@uniza.sk).

Peter Husár is with the Institute of Biomedical Engineering and Informatics, Technical University of Ilmenau, 98693 Ilmenau, Germany (e-mail: peter.husar@tu-ilmenau.de).

Digital Object Identifier 10.1109/JSEN.2023.3328357

issue. Device power consumption can be influenced already in circuit design. By selecting suitable electrical components with lower power consumption and subsequent testing, lower power consumption during operation can be achieved. Another option to reduce the power consumption of the system is software design. The power consumption of the microcontroller directly affects the power consumption of the entire Data Logger [5]. Along with designing the running time of the OffGrid system to operate efficiently in the time domain (scheduled operation), the issue of reducing code complexity rises. Both of these approaches are tailored to the specific application considering the environment, the environmental conditions, and the goals to be achieved by the data collection system.

In some cases, reliable connectivity is not available due to the absence or unsatisfactory data network infrastructure. Nevertheless, it is quite often necessary using independent (autonomous) devices or measuring stations and obtaining data in another way [6]. In the offline mode, data is often transferred via the serial bus or USB interface. The use of radio frequency identification (RFID)-type wireless communication for the transmission of measured files is also potentially beneficial and could increase the usability at the module installation site. A proposal for such a solution is presented in [7].

In this article, we propose a comprehensive "Advanced Data Logger" (ADL) solution, which includes hardware (HW) and software design with emphasis on the minimal power consumption of the HW module and efficient use of the logger's memory capacity for long-term measurements (in the order of months), with support for a wide range of sensors.

From a scientific point of view, our objectives are to verify the following hypotheses.

1) Reduction of total energy consumption can be realized by the following:
   a) appropriate integration of HW components;
   b) optimal adjustment of the microcontroller operating frequency;
   c) optimization of the program code—minimizing the time required to communicate with the sensor and the connected peripherals.
2) The way data is stored in the external memory also has an impact on reducing the overall power consumption. Due to the nature of the target application (data logger), it is possible to implement a more straightforward read-only file system (FS).

Having verified these hypotheses, our targets are mainly given as follows.

1) Design and development of a low-power data logger that allows efficient and reliable long-term monitoring of environmental variables in remote areas. We will focus our research on optimizing energy consumption through advanced energy management techniques and verifying its functionality and efficiency in real condition.
2) Design and implementation of an energy and HW-inefficient and efficient way of storing collected data on a storage medium through an optimized data organization algorithm.
3) Creation of supporting software in the form of clients and applications for communication with the proposed HW.

The main scientific objective is to develop a data logger for long-term environmental monitoring in remote areas with optimized energy consumption.

A. Advanced Data Logger Proposal

A common problem with data collection systems deployed in remote locations is their difficulty in accessibility. It means that it is not possible to have immediate access to them. In our case, it is a measuring station that is installed in remote locations. One of the main problems in the design of the proposed data logger is to reduce the total power consumption of the proposed module and thus prolong its effective operation time. Several approaches will realize the reduction of power consumption:

1) *HW Design:*
   a) selection of components with minimum consumption;
   b) appropriate choice of overall electrical wiring.
2) *Software Design:*
   a) utilization of an efficient way of storing data logger data—Simple File System (simpleFS);
   b) exploiting the low-power modes of the microcontroller used;
   c) optimizing the instructions in the firmware code—reducing the number of operations required.

The reasons for choosing HW components are given in Section III-C. Reducing power consumption through power-saving modes is described in Section III-E. Last but not least, optimizing the source code to reduce the number of machine instructions needed is an important optimization that should be addressed appropriately. The main task of the whole device is to perform measurements on the connected sensors and store the data on local storage. This storage can be a solid state drive (SSD) memory card or a FLASH ROM. We have excluded using SSD memory cards because of possible mechanical damage due to external meteorological conditions at the place of installation. The other option is to use FLASH ROM memory, characterized by the speedy memory read operation—on the order of units of clock ticks. In contrast, writing operations are time-consuming. Another aspect is the way the data overwrite operation is implemented. In FLASH ROM, once written, data cannot be modified. If data needs to be modified, the memory area or page needs to be erased, and the data needs to be rewritten. Using the standard file allocation table FS (FATFS) is possible, but the negative consequence is the necessary modification of data in the FAT area of the FS. In this article, we present a data storage method using our proposed Read-Append FS system, in which the need for data modification is eliminated. We have called this system simpleFS. The essential features of this design are as follows.

1) Support of memories with capacity from 4 kB.
2) Data is stored in binary format.
3) Possibility to create up to 128 files.

4) There is only one file in the system to which data can be written.
5) It is suitable for creating data files similar to CSV format.
6) The file name is generated automatically.
7) Each file has a well-defined structure: the number of columns or values for one record and the data-type specification for each value, respectively, column.

When designing the method of storing the measured data, the overall determination or realistic conditions for using the data logger and the requirements for minimum energy requirements had to be taken into account. The primary task of the data-logger is, therefore, to measure and receive data from the connected sensors and store this data in a storage medium in the briefest possible time. Since the ADL module is to operate autonomously, there is a need to eliminate all not explicitly required operations. In particular, this is the ability to open files on the storage medium—an algorithm needs to be defined to determine which file to open automatically. In the process of storing sensor readings, it is necessary to know the format of the measured data in advance. Since quantities of the same type (e.g., temperature) can come from different sensors with different raw data format. The simpleFS software module ensures a standardized data format for the same measured quantities. No modification of the stored data is required in the ADL lifecycle. Data is gradually incremented and written to memory. Thus, no file modification is required. Formatting or erasing the entire storage medium is the only destructive operation required.

The ADL device described herein contains a FLASH memory on which the measured data is stored. Part of the overall design is the proposal of a universal FS suitable for embedded modules such as data loggers, where data is captured at a point in time and a set of measured data needs to be stored, and these data may be of different types. The proposed simpleFileSystem data storage system stores data in line-oriented binary data files, where each line contains values from all sensors. The simpleFS software module presented in this article addresses the data storage itself, the data line format, and also the interface for processing the raw sensor data. Therefore, there is no necessity to deal with data file manipulation and formatting on the ADL application level.

The main goal of the work in this article is to significantly (measurably) extend the working mode of IoT data logger and support record level consistency using a custom FS implementation—simple FS. Simultaneously, verify the proposed solution experimentally for meteorological data measurements.

The presented solution, ADL is an autonomous data acquisition device implemented as a stand-alone module intended mainly for usage in remote environments in areas with limited or absent communication capability. The purpose of data logging can be to observe the condition of the monitored object or location over a long period of time. For example, to monitor changes in atmospheric pressure, relative humidity and temperature at a selected location. The ADL module is an embedded device based on the STM32L082 microcontroller and can communicate with standard commercial sensors with I2C, OneWire, SPI, or UART interfaces. Thanks to the real time clock (RTC) block, the ADL module keeps the current time after initial configuration via the software handler. For the measurement itself, a single timestamp is assigned to all measured data in a given measurement batch. The number of different measurements is limited only by the internal configuration of the library providing the implementation of the FS used. These limitations will be described in detail in the FS section.

II. Related Work

In this section, the generally available solutions for logging data are presented. There are several approaches available and well-known to loggers design.

A. Data Loggers Application

Depending on the application, various solutions differ mainly in the HW architecture with distinct limitations. Highly popular is utilizing electronic prototyping platforms such as Arduino and Raspberry Pi. For instance, Arduino was used to monitor weather with a set of digital and magnetic weather sensors connected to a data logger. As Bernandes et al. [8] presented, utilizing low-cost commercial and open-source IoT technologies fully meets the expectations regarding the reliability known for high-cost professional weather instruments. Of course, in professional applications, these prototyping platforms are rare. Devices based on world's well-known and reliable microcontroller platforms are more common in the practice of logging long time series of data. Specifically, this includes: STM32, Atmel, ESP, etc. Asaduzzaman et al. [9] used a platform for building smart data loggers preventing data losses caused by global system for mobile communication (GSM) and Wi-Fi network instability. For this purpose, a reduced instruction set computing (RISC) ARM microcontroller was implemented in the proposed system. The authors also declared that the system saves power on account of implementing fragmentation and request to send (RTS) threshold methods. The low-cost microcontroller Atmel ATmega328 was used to control climate conditions and their climate stability in the world heritage buildings. The dispersion observed in daily averages is much higher than that of the weekly or monthly levels [10]. Often, methods for data collection use wireless technologies, especially in wearable smart devices. The epidermal data logger based on RFID was used in Miozzi et al. [11] to transfer measurement data of skin and to upload the stored data into a crossing gate. Presented solution focuses on logging the temperature and the moisture of the skin. Another example for using low-cost devices in medical applications is presented by Srisuchinwong et al. [12]. The ESP32 microcontroller was used to transfer biomedical data such as the heart rate, SpO_2, or the body temperature to record data for analyzing sleep quality. Moreover, Del-Valle Soto [13] implemented a model using sleeping algorithms, focused on the analysis of energy impact on different types of routing protocols. Another example shows Mulyana et al. [14] which demonstrates a data logger as a part of smart city solutions. Their HW module is based on the ATmega microcontroller. The proposed module monitors voltage level from photovoltaic

TABLE I
EXISTING DATA LOGGERS SOLUTION COMPARISON

Paper	Application	Special properties	Platform
[8]	Low-cost automatic weather stations	Offline datalogger, Storage: SD card	Arduino Mega
[9]	Smart Data Logger for Enhancing Data Communication	WiFi and GSM connection	PC
[10]	An Intelligent Bed Sensor System for Non-Contact Respiratory Rate Monitoring	Wi-Fi connection, remote data collecting, ML algorithms	commercial solution
[14]	Data Monitoring System of Solar Module with DataLogger	Storage: SD card	ATmega 328
[15]	A Data Collection Collar for Vital Signs	LoRa connection	STM32Lx
[16]	Sensor-Logger for recording vertical movement in free-living organisms	storage: Internal memory	MSP430
[17]	A Smart Real Time Portable Multichannel Data Logger	Communication: Bluetooth, smartphone app	PIC18F4620
[20]	IoT-based data logger for weather monitoring	external Wi-fi module	Arduino Uno
[21]	Portable data logger	Communication: serial interface, data storage: EEPROM	Microchip PIC17C73B

system of public street lighting. Furthermore, STM32 solutions were applied in agriculture [15] as a logger for measuring vital parameters of cattle on farms where data was transferred by the LoRa system. Another agricultural example for the application of low-cost and low-powered systems is the open source freely available and noninvasive data logger for measuring individual movements in free ranging animals respecting ecological principles designed by Shipley et al. [16]. Developed system is based on a tiny and low-powered MSP430 microcontroller, which enables for almost 100 days of work in the active mode. Another way is the application for collecting data in monitoring of industrial machines operating parameters [17], [18]. As in other fields, microcontrollers were affected by the opportunity of implementation of advanced algorithms based on machine learning. Kalliris et al. [19] implemented global positioning system (GPS) and controller area network (CAN) bus data logger to classify road conditions based on acoustic waves. As we show above, IoT data loggers are implemented in various fields.

Mabrouki et al. [20] proposed a system based on Arduino, made to monitor air and weather conditions without using much energy. It uses special sensors that do not need much power and has a built-in Wi-Fi chip, making it affordable and efficient. Luharuka et al. [21] proposed another data logger (based on Microchip PIC16C73B) that is designed with readily available components and housed in a compact case capable of storing up to 280 h of Galvanic Skin Response data, thanks to a data compression algorithm and a 32-Hz sampling rate. Recording data sessions were validated in laboratory settings. The device consumes 210 mW of power and can function for around 50 h, which can be in this specific domain considered as low powered.

As is obvious from our research (see Table I), many researchers use different MCU platforms to design low-energy systems. For instance, Microchip PIC, Microchip Atmel, or STM. This is mainly due to their accessibility, versatility, and compatibility with various sensors and modules, which makes them suitable for energy-efficient projects. However, in our opinion, for the deployment conditions, we have preferred a custom board design utilizing STM32 microcontrollers. In the domain of low-powered data loggers, STM32 microcontrollers and custom-designed boards are known to their adaptability and efficiency. The STM32 microcontrollers (especially L family) are known for their minimal power consumption and high-performance processing capabilities. These are particularly effective in environments where energy conservation and real-time data processing are paramount. Custom-designed boards provide flexibility and cost-efficiency, allowing for the inclusion of only the necessary electrical components and the optimization of each for minimal energy usage, which is critical in applications with strict power constraints. Additionally, these boards can be tailored to optimize the physical layout for specific applications, offering enhanced environmental resilience. The advanced features such as integration capabilities and customization possibilities offered by STM32 and custom-designed boards make them more suited for sophisticated, power-sensitive data-logging solutions in professional settings.

In existing solutions that use battery power, the focus is on minimizing power consumption. In these cases, the authors used HW components with reduced power consumption (see [15], [16], [17]). In our present proposal, in addition to reduced-power HW components, we also focus on other power reduction methods described at the end of Section I.

B. Data Storage for Data Logger Purposes

Storing measured data is an essential requirement in designing a data logger. The first parameter in the implementation is the selection of the physical medium. When using a microcontroller as the controller, the option is to use SD card (SDIO interface), FLASH (SPI interface), or EEPROM (SPI or I2C interface) memory. The second parameter is the choice of a low-level data format or a suitable FS. The use of the standard FATFS seems to be a logical choice. Jinhai [22] analyzes the use of FATFS in embedded systems and compares it with other FSs that are usable in embedded applications. In its practical application, FATFS is used with a layered architecture implementation where additional algorithms are used for optimized searching, reading, and writing of data to physical media. The use of FATFS for data storage study [23], where the authors describe a portable device built on the STM32F411 microcontroller that reads human physiological data and stores it on a prepared SD card. Here the authors used a write/read reliable FATFS, where they used a first in first out (FIFO) buffer in the buffer function so that at the time the data is read from the FS, they can prepare the just-scanned data for writing. The proposed buffer has a capacity of 512 B which is the maximum sector size in FATFS. He et al. [24] demonstrates two SD (Dual-TF) interfaces in use with an 8-bit STC microcontroller. Again, the FATFS is used.

Using the standard FATFS may not be the primary choice in some applications. This decision may be due to the nature

of the application, for instance, storage format requirements, or additional limitations. Xu et al. [25] in their paper "A Fault-Tolerant Non-Volatile Main Memory File System," proposed a specific way of storing data: the NOVA FS. This FS represents an advanced FS design comparable to commonly used systems such as ext4 or btrfs. The authors also provide a comparison of FSs in the study. However, the solution is more performance demanding for use with microcontrollers (microcontroller unit). It is rather intended for microprocessors category devices (microprocessor unit) [26] and used with PC operating systems. A detailed analysis and determining factors in selecting a particular type of physical storage medium can be seen in [27]. Mazumder and Hallstrom [27] list the parameters of existing memory types such as write/read speed, number of write/read cycles from DRAM, SRAM, FRAM, NVSRAM, and FRAM. The authors proposed a fast, lightweight, and reliable LoggerFS FS designed for wireless sensor networks. In designing their FS, they considered three types of data: sensor data, binary data, and configuration data. The LoggerFS FS utilizes a hybrid implementation approach when RAM, FRAM, and FLASHoperates simultaneously. The presented system is also using a fixed size for the file of metadata and it is implemented on an MSP430 8-bit microcontroller containing 4-kB RAM and 64-kB FLASH.

The problem of minimizing energy consumption is described in Othman and Maga [28], where the authors discuss the energy consumption concerning the set of functions provided by a sensor node in a wireless network. It is advisable to add the possibility of supplying the autonomous system from a source other than the battery or combining the following sources. The use of photovoltaic cells for energy harvesting in a sensor wireless network module environment is described in [29]. The authors compare different energy harvesting methods ranging from solar radiation through the use of mechanical vibrations to the use of thermal energy.

C. Energy Consumption

There are several ways to provide enough energy to power OffGrid systems. For simplicity, we can divide them into three main ways: 1) measurement-based energy profiling; 2) model-based energy estimation; and 3) simulator-based energy estimation [30].

Ensuring sufficient energy is also possible by using energy harvesting methods or decreasing the power consumption [31]. In our proposal, the method for reducing power consumption and extending the battery life has been selected. Bradley and Wright [32] focus on minimizing the power consumption of Atmega328P. The main research focus was to decrease the power consumption of SD card readers and its impact on the data logging system. Lambert et al. [33] proposed a strong foundation for power consumption distributional characteristics based on manipulation of processors clocking frequencies to reduce power consumption for Raspberry Pi Computers. The presented solution reduced power consumption significantly, with the purpose to use a full operating system on devices where it was not possible due to power consumption. Guo et al.'s [30] survey came up with a few crucial elements that affect energy consumption in the case of embedded devices.

TABLE II
EXISTING POWER MANAGEMENT METHODS FOR EMBEDDED DEVICES

Paper	Application	Special properties	Platform
[5]	Processor power and energy consumption estimation techniques in iot applications	HW design, Low Power modes	MCU for IoT devices
[32]	Optimizing SD saving events to maximize battery lifetime for Arduino data loggers	Utilizing of MCU Low Power modes	Arduino/ Atmega328
[33]	Power consumption profiling of a lightweight development board.	Change CPU clock frequency	MCU Teensy 4.0 and RasPi
[30]	A survey of energy consumption measurement in embedded systems	Detailed analysis of key elements in energy consumption	various
[34]	A 2.5-D Integrated Data Logger for Measuring Extreme Accelerations	Data storage: FLASH, comm. interface: serial	Xilinx FPGA

Gakkestad et al. [34] introduce a 2.5-D integrated data logger characterized by its low power consumption and efficiency. This model prioritizes energy-saving, enabling prolonged and consistent data collection, low power and efficient 2.5-D integrated data logger with minimal power consumption.

Table II shows an overview of the analysis performed on the current state of the art in reducing power consumption in embedded devices. Existing solutions consistently focus on a specific application task. In our proposal, it is a device for data collection in remote areas, and the whole proposal is subordinated to this power saving. The strength of our design is the targeted focus on power consumption from several aspects of design and development: from the choice of HW components, the design of the HW circuitry, the way the firmware is created, and the way the data is stored on persistent memory with a minimum number of operations, but using a particular data organization; the time optimization of the measurement cycle for the temperature sensors used. The weak point of our design may be the way of storing the measured data on the storage medium—to read this data, a software operator is needed, but it is included in this design.

III. MATERIALS AND METHODS

This section describes the implementation of the ADL module itself. The overall solution consists of HW and software design. In software design, a significant part is devoted to designing and implementing the simpleFS library. The module is designed to operate in three functional modes (see Fig. 1).

The first, RUN mode, a reduced-power mode in which the necessary measurements are made, data is saved or sent, and the module transitions from active to deep sleep (STANDBY). The RUN mode activates itself based on the measurement frequency settings. In this mode, it will open the last file on the FS, where the measured values will be stored, the values from the connected sensors will be read, and the measured values will be written to the end of the open file. In order to minimize power consumption, the microcontroller core frequency is set to the minimum values at which proper

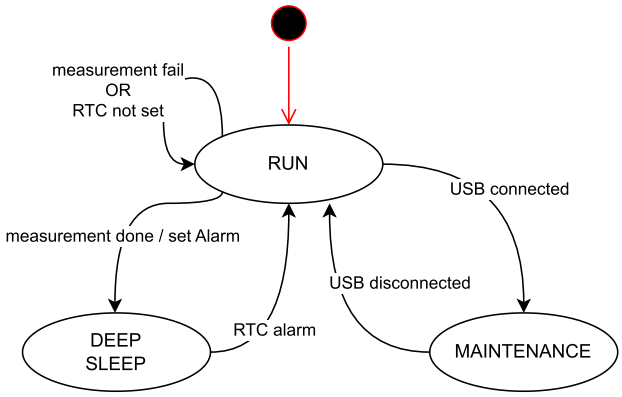


Fig. 1. ADL state diagram.

functionality is ensured, and at the same time, the microcontroller will not limit the connected peripherals with its speed. After this action, the ADL enters the minimum power consumption mode DEEP_SLEEP, in which all peripherals and the microcontroller core are deactivated.

The second mode—MAINTENANCE—is activated when the ADL module is connected to a PC for the purpose of downloading measured data. In this mode, the battery disconnector circuitry is activated, and the microcontroller is set to maximum speed to maximize the transfer rate between the module itself and the PC. In this mode, it is possible to set the parameters of the ADL module, such as the current time for the RTC or the interval for automatic measurements. In this mode, the measured data can be downloaded to the PC.

The third mode is DEEP SLEEP mode. In this mode, the RTC is set in the Alarm function. When the set time interval has elapsed, it will cause the application to wake up set to RUN mode.

A. Simple File System Proposal

The simpleFS custom library defines how to retrieve data, format data into a standardized form, create data files, and distribute data. In this work, we will use simpleFS as a simplified FS suitable for embedded devices. Its basic features include a straight overhead for fast data storage on the used storage medium itself. By "fast storage" we mean minimizing I/O operations, and thus the storage speed is close to the limits of a given storage medium. Due to the nature of the resulting application, simpleFS has been designed as a Read-Append system. The system allows writing a file, but operations such as deleting a file or modifying its name are not supported. The reason for this is that the primary use of simpleFS is in a data logger application, where the main purpose is to store data. Toward implementing simpleFS on a specific storage medium, there is an implementation interface. Any interface (SPI, I2C, one-wire, parallel interface) can be used for the storage medium that implements the simpleFS interface to access basic I/O operations. simpleFS can work with any storage medium.

Implementation features of simpleFS are as follows.

1) The Read/Append method has been implemented for dealing with the memory.
2) SimpleFS is independent of memory type (FLASH, EEPROM) and communication interface (SPI, QSPI, one-wire, I2C).
3) Memory support from 512 B of capacity.
4) The system is optimized for fast data storage and reading. The time-consuming "delete" operation is not used—except for deleting the entire memory in MAINTENANCE mode.
5) Minimal overhead of simpleFS: 112 B–12 kB of the total memory capacity for simpleFS itself, depending on the chosen configuration (see Table III).
6) Maximum number of files depending on the size of memory and chosen configuration (see Table III).

TABLE III
VARIANTS OF SIMPLEFS CONFIGURATIONS

Type	Code	MAX FILES	Values per row	HEADER SIZE	META RECORD	META SIZE
Ultra small	U	1	4	32	48	48
Extra small	X	2	8	64	96	192
Small	S	8	16	256	192	1536
Large	U	22	42	704	504	11088

File properties in simpleFS are as follows.

1) Data is stored in a binary format—a fixed-point format.
2) The file is line-oriented, each line contains a timestamp.
3) Several sensor records can be attached to a timestamp.
4) Data is always appended at the end of the last file.
5) The file name is automatically generated as a unique string derived from the HW identifier (ID) of the memory used.
6) The file size is limited by the size of the storage medium, max. size is 4 GB.
7) Files are self-describing; including meta-information about the stored data and its format in the file.

For storing data on simpleFS, an application programming interface (API) (or library itself) is provided that contains the following provisioning:

1) correct formatting of the same data from different sources. For example, the format for storing temperature will always be the same even when using different sensors with different output value representations;
2) output formats, ensuring that the saved record is formatted according to the specification of supported data types or supported sensors;
3) automatic opening/creating the file according to the number and type of available sensors when starting the data storage.

Fig. 2 shows the principle flowchart for initializing and working with simpleFS. When the application starts, it tries to open the last file in FS. If the file exists, it will open it. It is followed by checking the contents of the file (finding out the number and type of sensors that are stored in the file) and checking the physically attached sensors. If the two sets are not identical, a new file is created with the current list of sensors. If this file is successfully created, the application enters the RUN state. It transitions to the nonstandard ERROR state and signals a failure if unsuccessful.

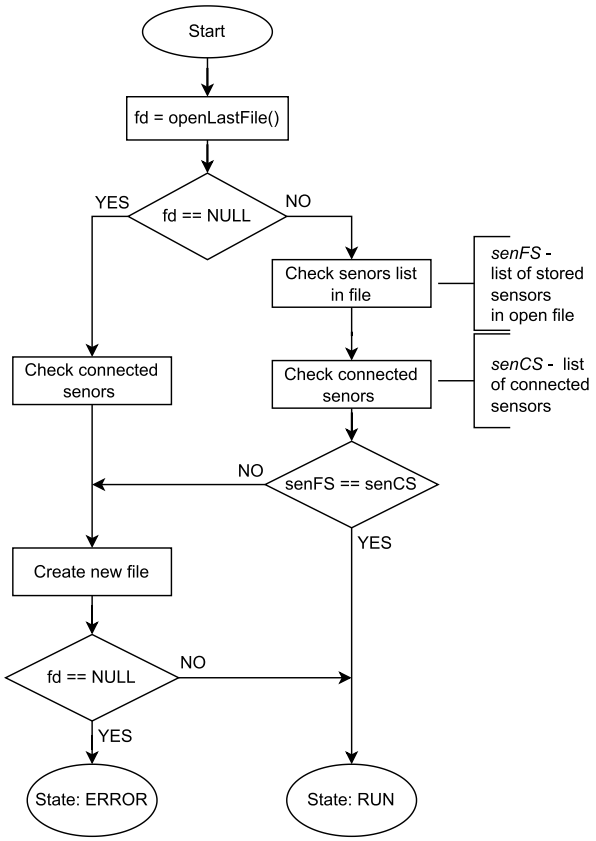


Fig. 2. Principal flowchart for initializing and working with simpleFS.

B. SimpleFS Module Structure

When designing the software module or library of simpleFS, we defined the requirements for the resulting library.

1) C language implementation to ensure easy implementation in other projects—an API will be created to access the essential functions.
2) Independence on the used memory medium and its capacity—simpleFS must be usable for memories with SPI, I2C, one-wire communication interfaces; the structure of simpleFS must be applicable for memories from 512 B (one-wire memories) to tens of MB (FLASH memories).
3) A unified interface shall be created for storing data of any type—sensor data.

In the following text, we describe the memory map for simpleFS.

1) Address Space: The available address space of the storage medium is divided into four parts: PREAMBLE, HEADER, META, and DATA (see Table IV and Fig. 3). The PREAMBLE part is the memory ID; it contains information about the version of the simpleFS implementation, the type and capacity of the memory used, a unique memory ID, the memory space usage variant used (see Table III), and the maximum number of files that can be stored on simpleFS. The size of the PREAMBLE area is constant at 32 B.

The structure of the individual memory areas is designed in such a way that the determining parameters by which the sizes of the individual areas will be set as: the maximum number of files per simpleFS and the maximum number of data in a one file line. These can be chosen arbitrarily, but there are four configuration variants (see Table III): ultrasmall—for memories with a minimum capacity of 512 B–2 kB, extra small—for memories with low capacity (1–64 kB), small—for memories with small capacity (32 kB—few MB), and large—mostly SPI FLASH-type memories with capacity from 1 MB. These values are only recommended and can be modified.

TABLE IV
MEMORY AREAS OVERVIEW

Area	Start address	Size
PREAMBLE	0	32B
HEADER	HEADER_ADDRESS	32B - 704B
META	META_ADDRESS	48B - 11088B
DATA	DATA_ADDRESS	up to memory capacity

The second area of the memory is labeled HEADER and contains a list of files that are stored on the storage medium. The information about each file in this section occupies 32 B. It contains the following sections.

1) File ID (1 B).
2) FILE_META_ADDRESS—the address to the block containing the file format description (4 B); this address is located in the META area.
3) FILE_DATA_ADDRESS—the address of the beginning of the file (4 B); this address is located in the DATA area.
4) Filename (16 B).
5) VALUES_PER_ROW—the number of data in a one line of the data file (1 B).
6) RECORD_LENGTH—the record size for a one line of the file (2 B).
7) N/A—reserved are 4 B.

The size of the HEADER area is defined by the maximum number of files that can be stored on simpleFS. The minimum size for the U variant is 32 B, and for the L variant is 704 B. The size of the HEADER area is expressed by the following equation:

$$\text{HEADER\_SIZE} = \text{MAX\_FILES} * 32\ \text{B}. \quad (1)$$

The `MAX_FILES` value is determined by the used configuration variant (see Table III) but another value can be used.

The third area, META, contains meta-information about the files. This area is located after the HEADER area, is constant, and contains information about existing files on simpleFS. The data in the data file is stored line by line, where each line represents values from attached sensors or generated data. The META section stores information about each data line's type, format, and length. The data in a line is described using 12 B as follows:

1) ID—sensor ID (8 B);
2) sensor type—classification of the stored value according to the internal codebook (1 B);
3) sensor value format—specification of the format for the data (3 B):

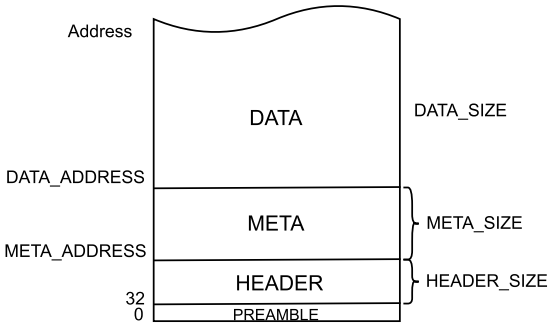


Fig. 3. Memory map of simpleFS.

a) record value type—record length for the stored value and interpretation format (integer, decimal point) (1 B);
b) record Q type—precision specification for representing a real number in the fixed-point format (2 B).

The data required to describe the format of the data in the file taken $(n \cdot 12)$ B, where n is the number of data to store. For variant U, the META area is 48 B, variant X—192 B, variant S—1536 B and variant L—11088 B. The relation gives the size of the META area in bytes

$$\text{META\_SIZE} = \text{MAX\_FILES} * \text{vpr} * 12\ \text{B} \tag{2}$$

where `vpr` represents `VALUES_PER_ROW`. The last, fourth, area of the address space is labeled DATA. This area is used to store the files themselves. A memory map of the simpleFS system is shown in Fig. 3. As can be seen from relations (1) and (2), the parameters that define the size of the overhead (the HEADER and META regions) are the maximum number of files (`MAX_FILES`) and the number of data in a single file line (`VALUES_PER_ROW`).

The HEADER area lists the files, with 32 B reserved for each file. The starting address for the ith file in the sequence is

$$\text{FILE\_HEADER}(i) = \text{FILE\_ID} * 32\ [\text{B}]. \tag{3}$$

From this record, we can read the address for the description of the ith file and the beginning of the data part

$$\begin{aligned} \text{FILE\_MA}(i) &= \text{FILE\_HEADER}(i)[1,\ldots,4 \\ \text{FILE\_DA}(i) &= \text{FILE\_HEADER}(i)[5,\ldots,8] \end{aligned} \tag{4}$$

where `FILE_MA` (`FILE_DA`) is the acronym to `FILE_META_ADDRESS` (`FILE_DATA_ADDRESS`). The expression [1..4] denotes the first to fourth byte in the given data block. The data structure `FileDescriptor_t` (Source code 1) holds information about single file. It contains file ID, address for file structure description (FILE_META_ADDRESS), and address for file content (FILE_DATA_ADDRESS).

At the address `FILE_META_ADDRESS(\textit{i})`—address of the ith file metainformation—there are data blocks describing file structure, each of size 12 B. It contains three parts: identification of data source (e.g., ID of sensor as a data source)—8 B, type of stored data—1 B, and format of stored value—3 B. The principle of this addressing is in

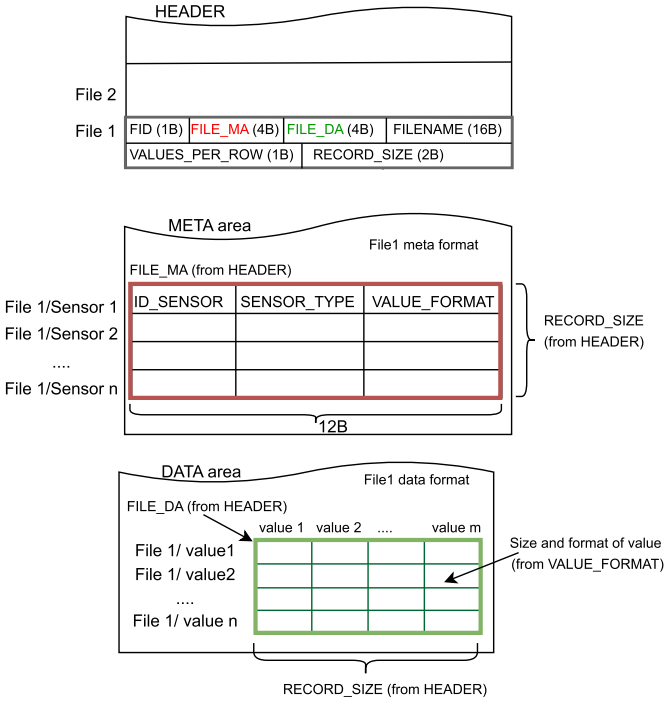


Fig. 4. Map of memory record.

```
// Data structure for HEADER part
typedef struct {
   uint8_t ID;
   uint32_t meta_address;
   uint32_t data_address;
   uint8_t name[FILENAME_LENGTH];
   uint8_t num_parts;
   RecordDescriptor_t format[MAXIMUM_FS_SENSORS_PER_FILE];
   uint16_t record_length;
   uint16_t recordAddressMap[MAXIMUM_FS_SENSORS_PER_FILE];
} FileDescriptor_t;

// Data structures for META part
typedef struct {
   uint8_t sensor_id[8];
   SensorType_t sensor_type;
   RecordTypeValue_t value_format;
} RecordDescriptor_t;

typedef struct {
     ValueFormatMetaFmt_t meta_format;
     ValueFormatMetaLength_t meta_length;
     RecordTypeQ_t Q;
} RecordTypeValue_t;
```

Source Code 1. Data structures for file representation.

Fig. 4. This information is provided as program structure `RecordDescriptor_t` in Source code 1.

2) Memory Interface: The simpleFS library defines the address space, data and file handling, but does not include functions for direct memory access. To implement a particular communication interface, the `MemoryDriver_t` structure (Source code 2) has been defined, which defines the set of the low-level operations to be implemented for a particular memory or memory communication interface.

The simpleFS library provides drivers for the SPI interface, the one-wire interface, and the dummy interface (memory RAM block) that is used for simulation (see Fig. 5).

3) Sensor Values Interface: To provide a uniform interface for loading data into a file, the `SensorInterface_t` interface (Source code 3) was created to provide the same

```
typedef struct{
  uint8_t (*Init)(void *hw1, void* hw2, uint16_t param);
  uint16_t (*Read)(uint32_t addr,uint8_t *buf,uint16_t n);
  uint16_t (*Write)(uint32_t addr,uint8_t *buf,int32_t n);
  uint8_t (*IsBusy)(void);
  uint8_t (*EraseAll)(void);
  uint8_t (*EraseSector)(uint32_t addr_start);
  uint8_t (*PowerUp)(void);
  uint8_t (*PowerDown)(void);
  uint8_t (*Capacity)(void);
  uint8_t (*Manufacturer)(void);
  uint8_t (*Type)(void);
  uint64_t (*Id)(void);
}MemoryDriver_t;
```

Source Code 2. Interface `MemoryDriver_t`.

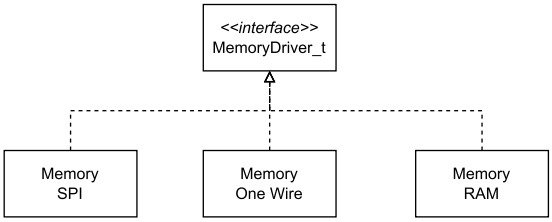


Fig. 5. Implementation of interface Memory_interface in simpleFS.

```
typedef struct{
     void (*Init)(void* obj1, uint16_t obj2);
     void (*Reset)(void);
     SensorID_t* (*getSensorId)(uint8_t id);
     RecordDescriptor_t (*getType)(void);
     void (*Measure)(void);
     SensorValue_t* (*getValue)(void);

 }SensorInterface_t;

 typedef struct {
     uint8_t *value;
     SensorType_t type;
     uint8_t data_length;
 }SensorValue_t;
```

Source Code 3. Interfaces `SensorInterface_t` and `SensorValue_t`.

access to each action: initializing the sensor (`Init`), resetting the sensor (`Reset`), getting the sensor ID (`getSensorId`), getting the sensor-type specification (`getType`), triggering the measurement of the value on the sensor (Measure), and retrieving the measured value (`getValue`).

Using the `SensorInterface_t` interface, all sensors implemented in simpleFS can be accessed in the same way.

In the current version of simpleFS, interfaces for sensor data or data types are implemented.

1) RTC data; the RTC peripheral of the STM32Lx microcontroller is used.
2) A/D converter data; the ADC peripheral of the STM32Lx microcontroller is used.
3) Data from DS18B20 temperature sensors on the one-wire bus.
4) Implementation for storing primitive data types.

4) Application Programming Interface: The API functions in the simpleFS library are divided into two categories.

1) Low-level API—universal memory interface access. A specific implementation of the MemoryDriver_t driver is used.
2) User-level API—user functions for working with simpleFS itself and files.

SimpleFS is designed as a Read-Append system, ready for fast addition and reading of data. The DELETE operation is implemented over the entire address space. The Read-Append implementation defines constraints on creating new files and storing values in files.

1) Files occupy free space sequentially in the DATA area.
2) Data can only be stored in the latest file on simpleFS.
3) When a new file is created, the previous file is closed and no more data can be saved into it.

Other properties of files on simpleFS are as follows.

1) The file name is generated automatically according to the unique ID of the memory used; only alphanumeric characters are used in the name generation; the file name consists of the generated name, the file sequence number, and the suffix "adl."
2) The data is stored in a binary format to save space; the data format for each file is stored in the META area.

For working with files, a user-level API has been created that provides file manipulation on one side and communication with a specific memory on the other side.

The interface for working with simpleFS can be divided into two groups: functions for manipulating files and functions for inserting data. These functions form the primary interface for working with files in simplegaFS.

1) *FS Related Functions:*
   a) FS_init() - init used low-level communication interface;
   b) FS_create_file()—create new file;
   c) FS_open_file(id)—open file with given ID;
   d) FS_num_files()—return number of existing files in simpleFS.
2) *File-Related Functions:*
   a) file_add_column()—add a new column in the data file. The column in the created file represents a sensor value. It has to be called in file creation procedure.
   b) file_data_row_begin()—create a new empty record.
   c) file_store_data()—add new value to a new record.
   d) file_data_row_commit()—write out prepared data to a file.
   e) file_size()—return size of an opened file.

C. Hardware Module ADL

The ADL module uses the STM32L08x microcontroller, which belongs to the ultralow power microcontroller family. A battery is used as the power supply (in STANDBY and RUN mode) or power from the USB communication interface is used in MAINTENANCE mode. The following peripherals are part of the HW design.

1) *External Memory for Storing Measured Data:* The memory-type used is NOR FLASH Winbond W25Q128. SPI is used as the communication interface.
2) *USB or UART Communication Interface:* It is used for communication with the ADL module and a file transfer to PC.

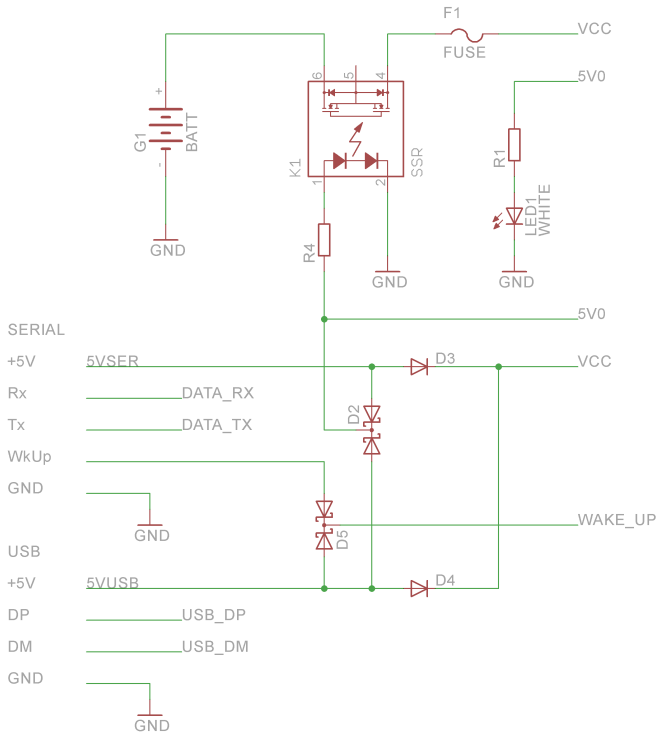


Fig. 6. HW design of automatic battery disconnection in the MAINTENANCE mode.

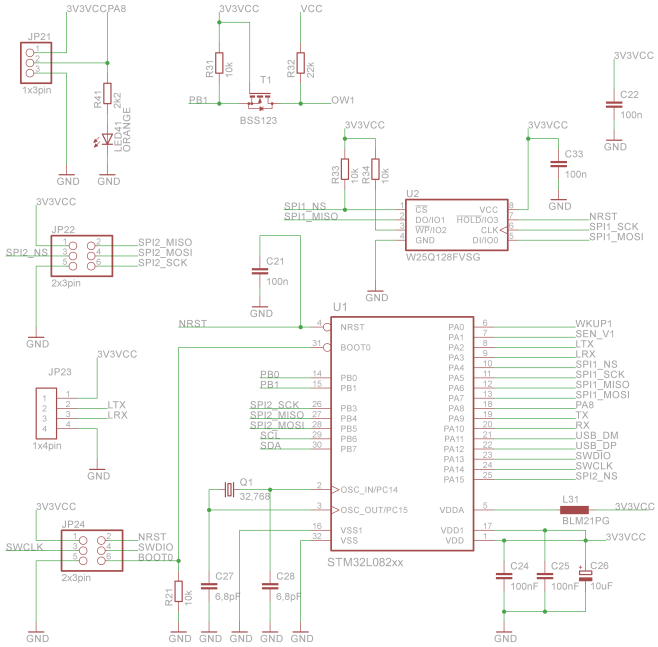


Fig. 7. ADL wiring diagram and microcontroller block.

3) *One-Wire Communication Bus:* This software implementation uses one-digital I/O pin on the microcontroller side.
4) *Control Circuit for Power Management:* When the communication interface (USB or UART) is connected, the SSR relay is opened, ensuring that the battery is decoupled from the supplied voltage. The switching time of the used solid-state relay is short (approx. 5 ms), which prevents unwanted restart of the connected microcontroller (see Fig. 6).
5) Available communication interfaces (I2C, SPI, UART) for connecting additional sensors.

ADL HW module wiring diagram is shown in Fig. 7.

We provide a detailed list of HW components that significantly contribute to the overall reduced power consumption design. This list corresponds to the wiring diagram in Fig. 6.

1) MCU: STM32L082—32-bit ARM Cortex M0+.
2) Low drop output (LDO) voltage controller *MCP1703T3302E*. Not shown in Fig. 7, but it is wired as recommended in the datasheet.
3) Solid-state relay *LCB127* for detecting battery power. Wiring according to the recommended wiring from the datasheet (see Fig. 6).
4) 128-Mb Serial Flash Memory *25Q123FVSG* (see Fig. 7) for storing measured data.
5) N-channel FET transistor *BSS123* for controlling the one-wire bus to which the digital one-wire temperature sensors are connected.

D. ADL Module Firmware

The ADL module operates in two active and one inactive modes, as shown in Fig. 1. In terms of HW implementation, these operating modes are described as follows.

1) RUN mode is the basic mode, which is activated to collect data and store it in a memory. Since the ADL module uses a battery as a power source, the goal is to minimize the time when this mode is active. In this mode, the operating frequency of the microcontroller is set to 4 MHz.
2) MAINTENANCE mode—the mode in which the ADL module is connected via USB or UART communication bus to a PC. The battery power supply is disabled. The available power from the communication interface is used as the power source. The operating frequency of the microcontroller is set to 32 MHz, which is the maximum frequency of the used microcontroller.
3) STANDBY mode—the mode where all microcontroller peripherals except the RTC are disabled. In this mode, the microcontroller power consumption is in the order of microampers. The duration of this mode can be set from 3 s to one day with a resolution of 1 s.

The core part of the ADL HW design is the RTC block—a RTC that works with the alarm to ensure the application wakes up at predefined time intervals. This interval is configurable and can be changed using the AdlReader add-on software, which is used to open and save measured data files and also to configure the ADL HW module in the MAINTENANCE mode. Fig. 8 shows the principle flow diagram of the proposed application. When the module starts or wakes up, after initializing the connected peripherals, an alarm is set first, which represents the time in seconds of the next wake-up of the application. The ADL application operates in two modes: RUN and MAINTENANCE (see also Fig. 1). Automatic mode selection is based on the detection of the connected communication interface. The MAINTENANCE mode is followed by the opening of the latest file located on the simpleFS. This is followed by a sensor detection

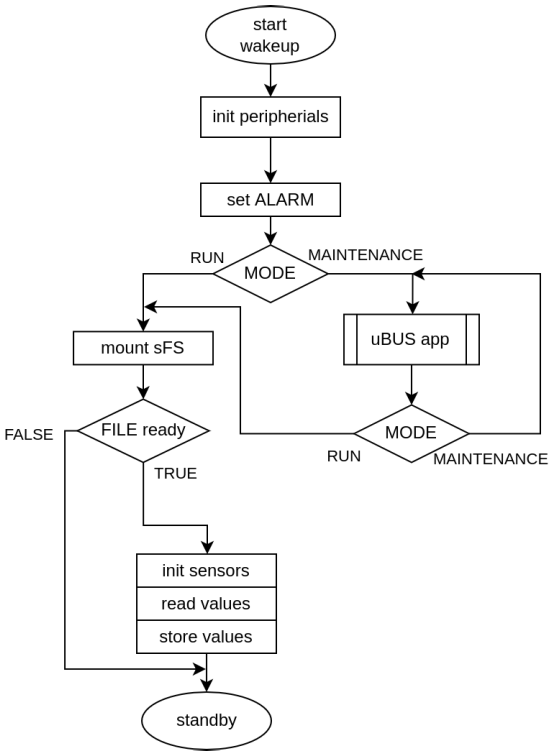


Fig. 8. ADL principle algorithm.

operation. A new data file is created if the number or type of sensors detected is different from the open file. Then the values from the prepared sensors are read, saved to the file, and the microcontroller goes into a deep sleep state (STANDBY mode).

When an active communication interface is detected after waking up the microcontroller, the application enters the MAINTENANCE mode allowing communication with the application using the uBUS communication protocol [35], [36]. If the communication interface is disconnected in MAINTENANCE mode, the application enters RUN mode as shown in Fig. 8.

There are three operational states of RUN mode as follows.

1) *OFFLINE:* The ADL module works offline. All measured data is stored in an external memory. To access the data files with measured values, the module must be connected to a PC.
2) *IOT-READY:* The measured data is stored in an external memory. The prepared data is sent to a remote server at defined time intervals. This interval is configurable.
3) *IOT:* After measuring the data, the data is immediately sent to a remote server.

E. Minimization of Energy Consumption

An essential aspect of the firmware design was ensuring minimal power consumption, especially in RUN mode when the battery is the only source of power. ADL application parameters in RUN mode.

1) Microcontroller operating frequency: 4 MHz.
2) Deactivation of unused peripherals: UART.

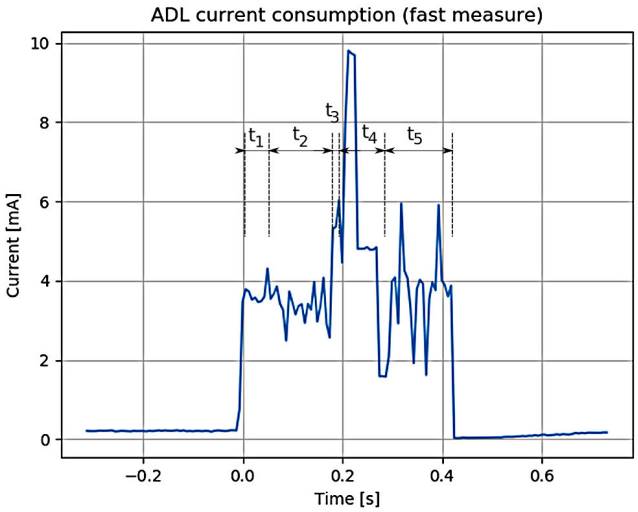


Fig. 9. ADL module current draw in RUN mode.

3) Microcontrollers internal operating voltage-level controller is reduced to 1.5 V when the core operating frequency is limited to 16 MHz (VOLTAGE_SCALE2 level) [37].
4) The operating time in this mode is minimized to the essential minimum, which includes initializing the peripherals, opening the data file, reading the values from the sensors, and saving the values to a file. After this active time, the ADL device switches to the STANDBY mode.

In the STANDBY mode, the power consumption is minimal. Only the RTC module is activated in the microcontroller, which consumes 730 nA. To this consumption is added the current consumption of the HW module, which totals in 210 μA.

Identification of ADL Current Draw in RUN Mode: Fig. 9 shows a plot of the supply current versus time in RUN mode (see Fig. 1). Over time, the current draw is affected by the operations defined in the ADL module firmware. The particular time periods can be divided into five parts.

1) t_1*:* Initialization of the necessary microcontroller peripherals. This time interval starts with waking up the microcontroller from STANDBY mode.
2) t_2*:* Initialization of the simpleFS system and initialization of the connected sensors.
3) t_3*:* Opening the active file, where the measured data will be written.
4) t_4*:* Starting the measurement on the sensors and waiting for the data to be ready.
5) t_5*:* Reading the measured values and writing them to a file.

In the ADL module, DS18B20 sensors were used as temperature sensors, resulting in time sessions that must be considered. The DS18B20 sensors are digital thermometers communicating on a one-wire bus. The one-wire serial bus communication protocol operates at a baud rate of 16.3 kb/s. Therefore, for example, it takes 20 ms to read a single reading.

The following time values t_1–t_5 were measured with the following configuration: five DS18B20 sensors, the number of files per simpleFS was 11.

1) $t_1 = 73$ ms.
2) $t_2 = 122$ ms. This time includes loading the list of available sensors. 20 ms is required to identify one DS18B20 sensor.
3) $t_3 = 15$ ms. This time includes opening the latest file to find the end of this file. The end-of-file search algorithm is optimized for speed—a modified binary search algorithm is used, but the resulting time is affected by the total memory capacity, which in this case was 128 Mb.
4) $t_4 = 100$ ms. Time required to convert the sensor data.
5) $t_5 = 150$ ms. Reading the measured data from the sensors and saving it to a file.

IV. Energy Consumption Minimization

An important part in terms of minimizing energy requirements is the time t_4 (see Fig. 9) when the MCU has to wait for the temperature data transfer to complete. The consumption at time interval t_4 can be minimized by two simultaneous procedures.

1) *Reducing the Accuracy of the DS18B20 Sensor:* After sending a request to measure a value, it is necessary to wait a certain time (conversion time) until the measured value is converted, when using DS18B20 digital thermometers. This time depends on the resolution of the sensor. The resolution of the DS18B20 sensor can be set from 9 to 12 bit. The time required for conversion is from 94 ms for 9-bit resolution to 750 ms for 12-bit resolution. In the ADL application, 9-bit resolution was chosen.
2) *Reduction of the MCU Operating Frequency During the Wait Time:* The microcontroller supports operating frequency from 65 kHz (RANGE 0) to 4 MHz (RANGE 6) when using the internal *RC* oscillator as a clock source.

Table V shows the dependency of the measured microcontroller power consumption during the RUN mode life cycle with respect to the set frequency in the wait state for the measured temperature data conversion. The actual measurement process and reading of the values were as follows.

1) The operating frequency of the microcontroller was set to 4 MHz.
2) After sending the temperature conversion request, the new working frequency of the microcontroller was set from 131 kHz to 2 MHz.
3) After the conversion time expired, the frequency was again set to 4 MHz.

A. Basic Concept of Energy Minimization

From the instantaneous electric current consumption values *i(t)*, the total electric charge (*Q*) consumed was calculated from the following equation. The individual frequencies that were used as the operating frequencies of the microcontroller are listed in Table VI

$$Q = \int_0^t i(t)\,dt. \tag{5}$$

In case we know the different stages of the life cycle in RUN mode, we can write relation 5 in a simplified way as in the following equation:

$$Q \approx \sum_{i=1}^{5} t_i \overline{I_i} \tag{6}$$

where

$$\overline{I_i} = \overline{I(t)}\Big|_{t_{i-1}}^{t_i}. \tag{7}$$

The magnitude of the electric current at time interval t_4 is determined by the manufacturer of the used sensor. The current value represents the consumption in the mode of measuring the value on the sensor. The current level is approximately the same in the other time intervals. The amount of consumption can be influenced by setting the clock frequency to lower values. This method was also used in this case. Table VI lists the frequencies at which the total current consumption was measured.

As shown in Table V, when the frequency is set lower than RANGE 3, i.e., 524 kHz, the total charge consumed increases, which may be due to the increasing time required to change the frequency of the microcontroller. Additionally, the total charge consumed increases when the core frequency is increased above 524 kHz (RANGE 4 and 5). This is due to increased power consumption when the operating frequency of the microcontroller core is increased.

Table V shows the duration of the RUN mode and the measurement of the total power consumption as a function of the microcontroller operating frequency setting at time t_4.

A suitable choice is to reduce the operating frequency of the microcontroller to 524 KHz (RANGE 3) at time t_4, which will minimize the current draw in part t_4.

TABLE V
TABLE OF THE MEASURED POWER CONSUMPTION PARAMETERS OF THE ADL MODULE IN THE RUN MODE

Range	RUN time	Total power consumption
RANGE 1	0.557 s	1.72 mAs
RANGE 2	0.506 s	1.6 mAs
RANGE 3	0.498 s	1.56 mAs
RANGE 4	0.49 s	1.61 mAs
RANGE 5	0.49 s	1.63 mAs

TABLE VI
INDICATION OF MICROCONTROLLER OPERATING FREQUENCY RANGES

Range	MCU core frequency (HCLK)
RANGE 1	131.072 kHz
RANGE 2	262.144 kHz
RANGE 3	524.288 kHz
RANGE 4	1.048 MHz
RANGE 5	2.097 MHz

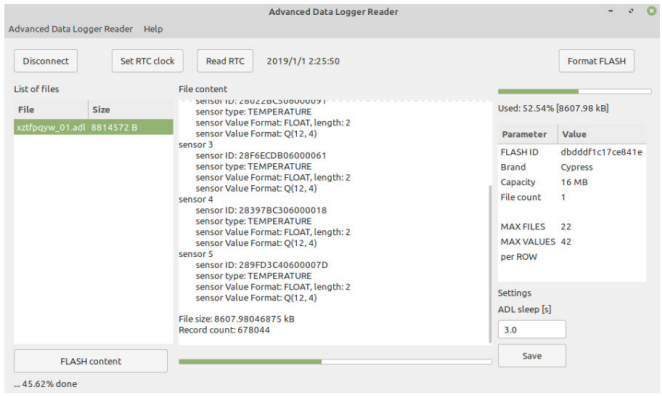


Fig. 10. AdlReader configuration application.

B. Support Software

The MAINTENANCE mode is automatically activated when the ADL module is connected to the USB/UART communication interface. The following operations can be performed in this mode.

1) Display information about the ADL module: memory used, its capacity, and used space.
2) Display the list of files stored in the ADL module.
3) Save existing files to the PC in the CSV format.
4) Set ADL module parameters: current time for RTC microcontroller module and the time interval for RUN mode activation.

Fig. 10 shows a preview of the AdlReader software control screen.

The *AdlReader* software was created in Python using the *senlib* library, which was created to communicate with measurement modules using the uBUS [38] communication protocol. The *WxPython* library was used to create the first version of the GUI. The resulting application is platform-independent and it works under commonly used operating systems.

V. Results

The proposed ADL module represents a complex solution, including the HW design and a software solution for storing the measured data in binary files, performing the measurements on the sensors at defined intervals, creating a software application for setting the parameters, and storing the datasets. An important aspect of the whole design is the total time of module autonomous operation. We can consider a measurement interval of 15 min. During these 15 min, the ADL module is in STANDBY mode and its power consumption is 0.2 mA. According to Table VII, this mode accounts for up to 99.94% of the total time. The working time in RUN mode depends on the number of connected sensors, but according to the results in Table V we can consider 0.5 s. The ADL module uses three pcs of AA batteries connected in series as a power supply. The capacity of AA batteries varies from 2000 to 3000 mAh. Let us consider a battery capacity of 2500 mAh and the consumption given in Table VII.

TABLE VII
Current Consumption in ADL Operating Modes

Mode	Active time	Percentage representation	Average power consumption
STANDBY	$t_{STANDBY} = 900s$	99.94%	0.22 mA
RUN	$t_{RUN} = 0.5s$	0.06%	4 mA

TABLE VIII
Table of the Measured Power Consumption Parameters of the ADL Module in RUN Mode

$t_{standby}$ [s]	$C_{standby}$ [mAs]	C_{overal} [mAs]	Measuring cycles [per day]	Consump. per day [mAs]	Operational time [day]
900	195.8	197.4	96	5.3	323
300	65.3	66.8	288	5.3	318
120	26.1	27.7	720	5.5	307
60	13.1	14.6	1440	5.8	291
40	8.7	10.3	2160	6.2	276

We calculate the average current consumed during one operating cycle as a weighted average

$$I = 0.22\ \text{mA} * 0.9994 + 4\ \text{mA} * 0.006 = 0.24386\ \text{mA}. \quad (8)$$

For the power supply, three AA batteries were used in series, so the supply voltage equals 4.5 V. The LDO voltage regulator used a minimum voltage value of 3.9 V. The accepted supply voltage level is, therefore, $U > 3.9$ V, which means a minimum voltage value for each battery of $U_{\text{bat}} = 1.3$ V. Lorenz [39] dealing with the design of a data logger for pressure sensing analyzed the power consumption requirements. In his study, AA alkaline batteries with the capacity of 2200 mAh and a constant discharge current of 50 mA were used. Comparing these results and the datasheets of different battery manufacturers and the fact that the discharge current in our solution is of short pulse nature, we chose 70% as an acceptable limit for the battery capacity. Thus, if we consider a battery capacity of 2500 mAh, its effective capacity will be 2500 mAh $* 0.7 = 1750$ mAh. The estimated battery lifetime is 1750 mAh/0.243868 mA $= 7176$ h $= 300$ days. Table VIII shows the consumption data in RUN and STANDBY modes as a function of the time t_{STANDBY}, i.e., the time between two measurement cycles. The real measured consumption of 1.56 mAs in RUN mode is used in the calculations. The consumption in STANDBY mode was measured $I_{\text{STANDBY}} = 0.2176$ mA. Column C_{STANDBY} represents the consumption during one interval t_{STANDBY}, and column C_{OVERAL} represents the consumption during a complete cycle: $t_{\text{STANDBY}} + t_{\text{RUN}}$. Depending on the time t_{STANDBY}, the number of measurements for one day is determined and the total consumption for one day is calculated from Fig. 10. The last column is the total operating time.

Another aspect of using the simpleFS library is the memory requirements. Since the data is stored in binary form, the volume of data is reduced. Again, let us consider a data measurement interval of 15 min and the following data to be stored: measurement timestamp, voltage of the battery used, and ten temperature sensors. The timestamp is 4 B in length, the battery voltage is 1 B, the temperature is 2 B in

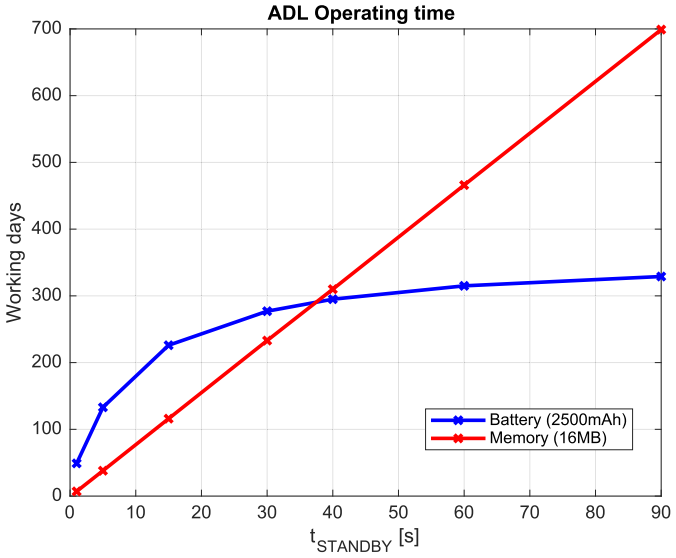


Fig. 11. Dependence of the ADL module operating time on the $t_{STANDBY}$ time.

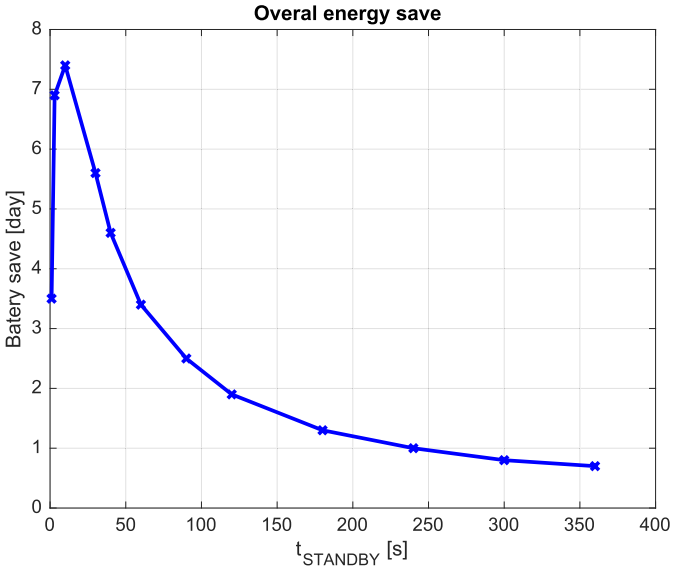


Fig. 12. Energy saving by changing the operating frequency of the microcontroller, expressed in days of operation.

length, for a total of 25 B. The number of measurements in 24 h is 96, so 2400 B is needed in one day. If the battery capacity is sufficient for 360 days, the required capacity is $360 * 2400$ B $= 844$ kB.

When determining the minimum time the ADL module is in STANDBY mode, both parameters must be considered: the capacity of the battery used and the memory capacity. Fig. 11 shows the dependence of the ADL module operation time on the selected interval $t_{STANDBY}$, when the ADL module is in the minimum power state. The parameters chosen for this chart are the battery capacity, where the total time limit approaches 365 days, and the memory capacity for storing all the measured data. If we consider a memory with a capacity of 16 MB, then the minimum STANDBY mode time is 40 s. At this time, the battery life is 295 days and the memory capacity is sufficient for 310 days.

In the section dealing with the ADL module consumption in run mode (see Table V), it has been empirically found that during the time when the microcontroller has to wait for the measured value conversion to complete on the sensors, it is advantageous to change the operating frequency of the microcontroller to 524.288 kHz. In Fig. 12, the energy savings are shown in terms of the number of days gained by this measure, depending on the measurement interval or the time $t_{STANDBY}$ when the ADL module is in the STANDBY state. The curve shows energy savings converted into days after MCU frequency optimization applied (microcontroller frequency reduced to 524.288 kHz).

As can be seen in Fig. 12 the most significant energy savings are when the $t_{STANDBY}$ time is approximately 10 s. However, at this measurement interval, the lifetime is only 189 days (according to Fig. 11). With a measurement interval of 5 min, the saving is converted to 1.2 days.

As mentioned in Section I, the data in ADL is stored in binary format, which guarantees raw data compression. To quantify the compression ratio, we take the file shown in Fig. 10 (the file in simpleFS is shown in AdlReader software),

Fig. 13. Experimental meteo field.

which has a binary size of 8722 kB. After converting it to a readable CSV file, it has a size of 34892 kB. The file contains 677 000 records. The saving of memory space compared to the text format is four times higher.

Case Study

The validation phase of the proposed design was implemented at the University of Zilina's campus (see Fig. 13), within a locally operated meteorological field. There are experimental meteorological stations on the territory of the University for the needs of research of the University of Zilina in this scientific field. In addition, we are planning a simultaneous deployment at a second experimental site with a higher level of influence from the external environment. The second site is located in a forest environment for the long-term measurement of soil temperature parameters near forest roads. Several meteorological systems are part of this field for intercomparison purposes. Further deployment of the proposed solution is planned in a forest road environment, where supporting measurements are performed in order to obtain data input to the pavement degradation simulation model. The measurements are carried out in two separate locations located in the valley at the beginning of the forest road and at the top of the mountain where the forest road ends.

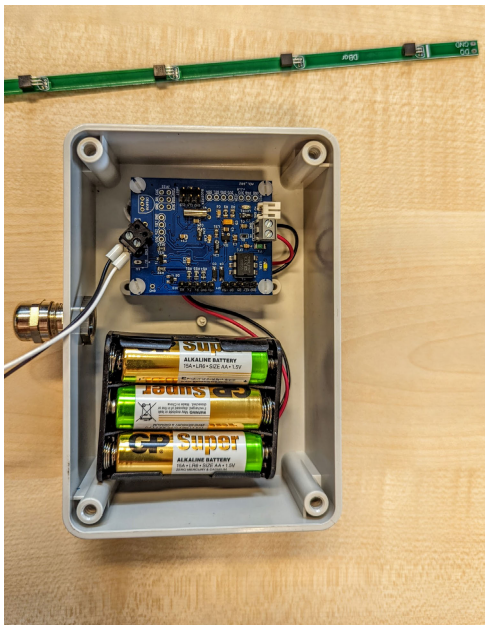


Fig. 14. ADL prototype.

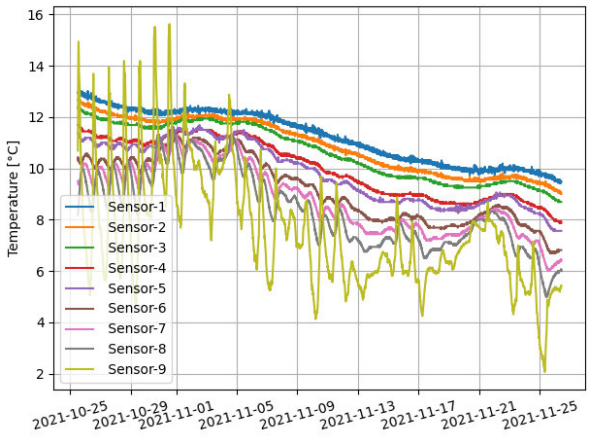


Fig. 15. Temperature measurements records obtained from the proposed data logger.

Fig. 14 shows a prototype of the ADL board with attached probe containing five DS18B20 thermometers ready for practical deployment.

The measured data from the vertical temperature probe for one month are shown in Fig. 15.

VI. Discussion

This article presented the ADL, a device designed for long-term autonomous data collection. In the design and implementation of the ADL solution, an effort was made to prove the hypotheses defined in Section I. Hypothesis 1 concerned ways to reduce the overall power consumption through appropriate choice of HW components, optimal microcontroller operating frequency setting, and application firmware optimization.

In the development of the presented solution, several methods of saving energy consumption of the ADL module were applied.

1) When choosing a microcontroller, we focused on microcontrollers of the STM32 family. Since this is a simple device, the target microcontrollers were identified from the low-end family containing the Cortex M0 core. There are two series: the STM32F0 and the STM32L0, with the F0 series referred to as the Value-Line and the L0 series as the "Low-Power" solution. These microcontrollers have similar power levels in RUN mode, but in STANDBY mode, the power consumption of the L0 microcontroller is an order of magnitude lower (1.22 versus 0.53 μA).
2) Utilization of power-saving modes of the microcontroller used. In the inactive state, the power consumption is reduced to 8.8% (0.22 mA) of the power consumption in the active state (4 mA) (see Table VII). The consumption of 0.22 mA includes the consumption of auxiliary circuitry. This finding confirms Hypothesis 1.A.
3) At the time of measurement (RUN mode), the frequency of the microcontroller core varied adaptively (see Table V and Fig. 12)—Hypothesis 1.B. In validating this hypothesis, we found that the RANGE_3 working setup, which refers to the voltage level of the logic signals at the microcontroller core level, is the most suitable for this task.
4) For the measurements on the connected one-wire sensors, the time for converting the measurement values was minimized from the original 750 to 100 ms (see Fig. 9, time t_4)—Hypothesis 1.C. By reducing the time required for the conversion, we have reduced the accuracy of the measurement from a resolution of 2^{-5} °C–2^{-1} °C, which is acceptable given the nature of the measurement.

The design of the HW part was focused on minimizing the power consumption. According to the measured consumption values in the RUN and the STANDBY modes, we estimated the runtime depending on the measurement interval in Section V. For a measurement interval greater than 5 min, the running time is 320 days, which is almost 11 months under the given conditions. To verify this hypothesis, we changed the parameter t_{STANDBY} to 3 s, and we used a Li-ion 18 650 type battery with an assumed capacity of 2000 mAh. With this configuration, the device's operation time was approximately 65 days. Converting the values in Table VIII using an assumed battery capacity of 2000 mAh yields a value of 79 days. The estimated and calculated runtime values are similar, which leads us to conclude that the values will be similar for a more extensive t_{STANDBY} time.

Hypothesis 2 was verified by designing a simplified storage system (see Section III-B). In this approach, a data storage method was designed where the operation of overwriting the data on the physical medium was eliminated. When NOR FLASH memory is used, reading from memory on a byte-by-byte basis or in block mode is possible. However, a write operation is a block operation that must be preceded by erasing a memory block (4-kB sector, 32-kB block, or 64-kB block). A write operation follows it. According to the datasheet for the memory used, the write time is defined as $\mathrm{tw} = t_{\mathrm{init}} + N \cdot t_{1\mathrm{B}}$, where $t_{\mathrm{init}} = 30$ μs, $t_{1\mathrm{B}} = 2.5$ μs, N = number of bytes to write. The penalty for utilizing this solution is a property of

the proposed FS—the proposed simpleFS storage system is a read-only system. This property is not a disadvantage for the target application.

The main benefit of the proposed solution is to optimize the ADL consumption in the working mode and at the same time to shorten this time. Consequently, this leads to achieve an extension of the effective operating time.

The experimental installation in order to verify the functionality of the ADL device was carried out at the Research Centre of the University of Zilina, where one ADL module with nine DS18B20 temperature sensors was installed. The data were collected with a period of 10 min for two months. A specially designed mechanical sensor system [40] referred to as DBAR was used for the installation. This is an original design of the sensor arrangement and its mechanical design to measure the vertical temperature profile of the soil cross section [41].

The correctness of the measured data from the ADL module was confirmed by a test measurement, where the same types of sensors (DS18B20) were used, and the measurements were taken at similar time intervals. The sensors were connected to the one wire booster (OWB) measurement module described in [38].

An alternative to the proposed solution is to use an existing FS, e.g., FAT32. This system is used by default for memory storage smaller than 4 GB. It is mainly used in memory cards. In terms of usability, it was unsuitable in the ADL project because the goal of designing simpleFS was to minimize the time required to communicate with memory. When using FAT32, overwriting the data in the FAT table is required, which can significantly increase the overhead of writing data. In the presented simpleFS system, to write data to a file, two actions are required before the actual write: open the last file in simpleFS (complexity O(1)), find the end of the file to be written to [complexity O(log(n), where n is the space reserved for the file in bytes].

A. Future Work

The ADL HW module in the current version is capable of long-term autonomous operation as a logger for data collection in remote locations. When designing the HW part, an extension of the functionality was foreseen adding a module for wireless communication. In terms of further optimization of the solution, we are currently focusing on improving key parameters.

1) Extending the autonomous operation time by partially recharging the used batteries with solar cells.
2) Optimization of the measurement cycle time when using multiple temperature sensors ($n > 5$). In this case, the total measurement time is increased due to the communication speed of the one-wire bus. It can be achieved by increasing the standard communication speed from the standard 16.3 to 114 kb/s in overdrive mode. However, when increasing the communication speed, the length of the bus, its topology, and shape must be considered.
3) Integration of IoT submodule for wireless batch communication. The goal is to leverage existing IoT infrastructure. The ADL module will contain a LoRa module, which will be used to send data to a prepared storage continuously.

VII. Conclusion

In this article, we proposed a novel FS intending to reduce the energy consumption of data loggers with the expectation of extending operating time. Overall, our results demonstrate a strong effect of utilizing our approach. Future research should consider the simpleFS in different environments and various application abilities.

Further modifications of the ADL module are planned to increase the energy independence, e.g., solar energy power for partial recharging of the batteries or other means of energy harvesting. Due to the original intention of the ADL module—use in remote areas—the assumption of direct solar radiation at the installation site may not always be guaranteed. Therefore, the PV panel solution hand in hand with other methods has to be analyzed in terms of the minimum harvested energy that the solution has to deliver for the overall energy balance to be positive.

Patents

SK122021U1 Temperature probe with adjustable position of the installed sensors.

References

[1] N. M. Kumar, K. Atluri, and S. Palaparthi, "Internet of Things (IoT) in photovoltaic systems," in *Proc. Nat. Power Eng. Conf. (NPEC)*, Mar. 2018, pp. 1–4, doi: 10.1109/NPEC.2018.8476807.

[2] K. Zhou, T. Liu, and L. Zhou, "Industry 4.0: Towards future industrial opportunities and challenges," in *Proc. 12th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, Aug. 2015, pp. 2147–2152, doi: 10.1109/FSKD.2015.7382284.

[3] T. Baranwal, Nitika, and P. K. Pateriya, "Development of IoT based smart security and monitoring devices for agriculture," in *Proc. 6th Int. Conf. Cloud Syst. Big Data Eng. (Confluence)*, Jan. 2016, pp. 597–602, doi: 10.1109/CONFLUENCE.2016.7508189.

[4] Z. H. Ali, H. A. Ali, and M. M. Badawy, "Internet of Things (IoT): Definitions, challenges and recent research directions," *Int. J. Comput. Appl.*, vol. 128, no. 1, pp. 37–47, Oct. 2015, doi: 10.5120/ijca2015906430.

[5] P. Y. Dibal et al., "Processor power and energy consumption estimation techniques in IoT applications: A review," *Internet Things*, vol. 21, Apr. 2023, Art. no. 100655, doi: 10.1016/j.iot.2022.100655.

[6] M. N. Hassan, M. R. Islam, F. Faisal, F. H. Semantha, A. H. Siddique, and M. Hasan, "An IoT based environment monitoring system," in *Proc. 3rd Int. Conf. Intell. Sustain. Syst. (ICISS)*, Dec. 2020, pp. 1119–1124, doi: 10.1109/ICISS49785.2020.9316050.

[7] M. Vagaš et al., "Wireless data acquisition from automated workplaces based on RFID technology," *IFAC-PapersOnLine*, vol. 52, no. 27, pp. 299–304, 2019, doi: 10.1016/j.ifacol.2019.12.677.

[8] G. F. L. R. Bernardes, R. Ishibashi, A. A. S. Ivo, V. Rosset, and B. Y. L. Kimura, "Prototyping low-cost automatic weather stations for natural disaster monitoring," *Digit. Commun. Netw.*, vol. 9, no. 4, pp. 941–956, Aug. 2023, doi: 10.1016/j.dcan.2022.05.002.

[9] A. Asaduzzaman, K. K. Chidella, and F. N. Sibai, "A smart data logger for enhancing data communication in Wi-Fi based mobile systems," in *Proc. SoutheastCon*, 2015, pp. 1–6, doi: 10.1109/SECON.2015.7132925.

[10] Q. Liu, M. Kenny, R. Nilforooshan, and P. Barnaghi, "An intelligent bed sensor system for non-contact respiratory rate monitoring," 2021, *arXiv:2103.13792*.

[11] C. Miozzi, S. Nappi, S. Amendola, and G. Marrocco, "A general-purpose small RFID epidermal datalogger for continuous human skin monitoring in mobility," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2018, pp. 371–373, doi: 10.1109/MWSYM.2018.8439572.

[12] D. Srisuchinwong, B. Sukhachewanon, and T. Chanwimalueang, "Acquiring unobtrusive sleep-related signals through an ESP32-based data logger," in *Proc. 13th Int. Conf. Knowl. Smart Technol. (KST)*, Jan. 2021, pp. 38–42, doi: 10.1109/KST51265.2021.9415820.

[13] C. Del-Valle-Soto, R. Velázquez, L. J. Valdivia, N. I. Giannoccaro, and P. Visconti, "An energy model using sleeping algorithms for wireless sensor networks under proactive and reactive protocols: A performance evaluation," *Energies*, vol. 13, no. 11, p. 3024, Jun. 2020, doi: 10.3390/en13113024.

[14] E. Mulyana, A. E. Setiawan, S. Sumaryo, and A. Munir, "Data monitoring system of solar module with data logger for public street lighting application," in *Proc. 26th Int. Conf. Telecommun. (ICT)*, Apr. 2019, pp. 280–283, doi: 10.1109/ICT.2019.8798777.

[15] Q. Li, Z. Liu, and J. Xiao, "A data collection collar for vital signs of cows on the grassland based on LoRA," in *Proc. IEEE 15th Int. Conf. e-Business Eng. (ICEBE)*, Oct. 2018, pp. 213–217, doi: 10.1109/ICEBE.2018.00041.

[16] J. R. Shipley, J. Kapoor, R. A. Dreelin, and D. W. Winkler, "An open-source sensor-logger for recording vertical movement in free-living organisms," *Methods Ecology Evol.*, vol. 9, no. 3, pp. 465–471, Mar. 2018, [Online]. Available: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.12893, doi: 10.1111/2041-210X.12893.

[17] H. Bassil, H. Moubarak, A. Kassem, M. Hamad, and C. El-Moucary, "A smart real time portable multichannel data logger system," in *Proc. 29th Int. Conf. Microelectron. (ICM)*, Dec. 2017, pp. 1–5, doi: 10.1109/ICM.2017.8268873.

[18] M. Compare, P. Baraldi, and E. Zio, "Challenges to IoT-enabled predictive maintenance for industry 4.0," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4585–4597, May 2020, doi: 10.1109/JIOT.2019.2957029.

[19] M. Kalliris, S. Kanarachos, R. Kotsakis, O. Haas, and M. Blundell, "Machine learning algorithms for wet road surface detection using acoustic measurements," in *Proc. IEEE Int. Conf. Mechatronics (ICM)*, vol. 1, Mar. 2019, pp. 265–270, doi: 10.1109/ICMECH.2019.8722834.

[20] J. Mabrouki, M. Azrour, D. Dhiba, Y. Farhaoui, and S. E. Hajjaji, "IoT-based data logger for weather monitoring using arduino-based wireless sensor networks with remote graphical application and alerts," *Big Data Mining and Analytics*, vol. 4, no. 1, pp. 25–32, 2021, doi: 10.26599/BDMA.2020.9020018.

[21] R. Luharuka, R. Gao, and S. Krishnamurty, "Design and realization of a portable data logger for physiological sensing [GSR]," *IEEE Trans. Instrum. Meas.*, vol. 52, no. 4, pp. 1289–1295, Aug. 2003, doi: 10.1109/TIM.2003.816808.

[22] Z. Jinhai, "Research of embedded FAT file system," in *Proc. Int. Conf. Uncertainty Reasoning Knowl. Eng.*, vol. 2, Aug. 2011, pp. 44–47, doi: 10.1109/URKE.2011.6007903.

[23] J. Zhang, W. Zhu, G. Ma, T. Tong, J. Zhong, and L. Wang, "Design of a portable physiological signal data storage system," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Oct. 2018, pp. 128–132, doi: 10.1109/SmartWorld.2018.00057.

[24] X. He, H. Deng, J. Ma, and H. Sun, "Research on information storing technology based on dual-TF card interface technology of MCU," in *Proc. Int. Conf. Qual., Rel., Risk, Maintenance, Saf. Eng.*, Jun. 2012, pp. 1464–1466, doi: 10.1109/ICQR2MSE.2012.6246500.

[25] J. Xu et al., "NOVA-fortis: A fault-tolerant non-volatile main memory file system," in *Proc. 26th Symp. Operating Syst. Princ.* New York, NY, USA: Association for Computing Machinery, Oct. 2017, p. 478, doi: 10.1145/3132747.3132761.

[26] STMicroelectronics. (2021). *Microcontrollers & Microprocessors*. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors.html

[27] B. Mazumder and J. O. Hallstrom, "A fast, lightweight, and reliable file system for wireless sensor networks," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2016, pp. 1–10, doi: 10.1145/2968478.2968486.

[28] A. Othman and D. Maga, "Relation between security and energy consumption in wireless sensor network (WSN)," in *Proc. New Trends Signal Process. (NTSP)*, 2018, pp. 1–8, doi: 10.23919/NTSP.2018.8524094.

[29] A. Othman and D. Maga, "Indoor photovoltaic energy harvester with rechargeable battery for wireless sensor node," in *Proc. 18th Int. Conf. Mechatronics Mechatronika (ME)*, Dec. 2018, pp. 1–6, doi: 10.1016/j.egypro.2012.01.164.

[30] C. Guo, S. Ci, Y. Zhou, and Y. Yang, "A survey of energy consumption measurement in embedded systems," *IEEE Access*, vol. 9, pp. 60516–60530, 2021, doi: 10.1109/ACCESS.2021.3074070.

[31] H. Wu, C. Chen, and K. Weng, "An energy-efficient strategy for microcontrollers," *Appl. Sci.*, vol. 11, no. 6, p. 2581, 2021, doi: 10.3390/app11062581.

[32] L. J. Bradley and N. G. Wright, "Optimising SD saving events to maximise battery lifetime for Arduino^{U}/Atmega328P data loggers," *IEEE Access*, vol. 8, pp. 214832–214841, 2020, doi: 10.1109/ACCESS.2020.3041373.

[33] J. Lambert, R. Monahan, and K. Casey, "Power consumption profiling of a lightweight development board: Sensing with the INA219 and teensy 4.0 microcontroller," *Electronics*, vol. 10, no. 7, p. 775, Mar. 2021, doi: 10.3390/electronics10070775.

[34] J. Gakkestad et al., "A 2.5-D integrated data logger for measuring extreme accelerations," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 7, no. 12, pp. 1930–1939, Nov. 2017, [Online]. Available: https://ieeexplore.ieee.org/document/8113561, doi: 10.1109/TCPMT.2017.2766263.

[35] J. Dud'ák, M. Skovajsa, and I. Sládek, "Proposal of a communication protocol for smart sensory systems," in *Proc. 16th Int. Conf. Mechatronics Mechatronika*, Dec. 2014, pp. 107–112, doi: 10.1109/MECHATRONIKA.2014.7018243.

[36] J. Dudak, G. Gaspar, S. Sedivy, P. Fabo, L. Pepucha, and P. Tanuska, "Serial communication protocol with enhanced properties–securing communication layer for smart sensors applications," *IEEE Sensors J.*, vol. 19, no. 1, pp. 378–390, Jan. 2019, doi: 10.1109/JSEN.2018.2874898.

[37] *Reference Manual RM0376—Ultra-Low-Power STM32L0x2 Advanced Arm®–Based 32-Bit MCUs*, STMicroelectronics, Geneva, Switzerland, 2021.

[38] J. Dudák and G. Gašpar, *Design and Implementation of Sensory Solutions for Industrial Environment: Utilizing 1-Wire® Technology in Industrial Solutions* (Signals and Communication Technology). Cham, Switzerland: Springer, 2023, doi: 10.1007/978-3-031-30152-0.

[39] R. D. Lorenz, "Observing desert dust devils with a pressure logger," *Geoscientific Instrum., Methods Data Syst.*, vol. 1, no. 2, pp. 209–220, Dec. 2012, doi: 10.5194/gi-1-209-2012.

[40] G. Gaspar et al., "IoT-ready temperature probe for smart monitoring of forest roads," *Appl. Sci.*, vol. 12, no. 2, p. 743, Jan. 2022, doi: 10.3390/app12020743. https://www.mdpi.com/2076-3417/12/2/743

[41] S. Sedivy, G. Gaspar, M. Farbak, and P. Fabo, "Temperature probe with adjustable position of the installed sensors," Žilinská univerzita v Žiline, Žilina, Slovakia, Tech. Rep. 9323, 2021. [Online]. Available: https://worldwide.espacenet.com/patent/search/family/075979890/publication/SK122021U1?q=pn:SK122021U1

Juraj Ďud'ák received the Ph.D. degree from the Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava, Slovakia, in 2011.

He is dedicated to low-level programming for sensors. He is an Expert in the design and implementation of software solutions. His specialization is code creation for single-chip microprocessors, design and creation of desktop multiplatform applications, and design of database solutions. He has authored scientific and technical papers in national and international conference proceedings and journals.

Dr. Ďud'ák is a member of the International Association of Engineers (IAENG) and a member of the Editorial Board of *International Journal of Sensors and Sensor Networks* (IJSSN).

Gabriel Gašpar received the Ph.D. degree in automation from the Faculty of Materials Science and Technology in Trnava, Slovak University of Technology, Bratislava, Slovakia, in 2016.

Since 2020, he has been the Managing Director of the Transport Infrastructure Research Division, Research Centre, University of Žilina, Žilina, Slovakia. He has authored scientific and technical papers in national and international conference proceedings and journals. He is a sole author or a coauthor of several IPRs. His research and development activities include the field of low-level microprocessor programming, PCB design, software applications in the field of sensory systems, and preparation of physical experiments to verify the proposed sensors.

Dr. Gašpar is a member of the International Association of Engineers (IAENG) and a member of the Editorial Board of *International Journal of Sensors and Sensor Networks* (IJSSN).

Roman Budjač received the master's degree from the Slovak University of Technology, Bratislava, Slovakia, in 2018, where he is currently pursuing the Ph.D. degree in the study program Automation and Informatization of Processes in cybernetics.

At present, he is in the final year of study. The doctoral thesis deals with deep learning compression and methods to minimize neural models and the possibilities of using deep learning on power-constrained devices. He is also a Researcher at the Research Centre, University of Žilina, Žilina, Slovakia.

Mr. Budjač is a member of the International Association of Engineers (IAENG). He was a successful recipient of two grants as the principal investigator in the years 2021 and 2022 to support young researchers at Slovak Technical University in Bratislava.

Ivan Sládek received the M.Sc. degree in electronics engineering from the Faculty of Electrical Engineering, University of Žilina, Žilina, Slovakia, in 1991.

Since 2021, he has been a Researcher with the Research Centre, University of Žilina. He has authored scientific and technical papers in national and international conference proceedings and journals. He is a coauthor of several IPRs. His research and development activities include the field of electronics and PCB design, and software applications in the field of control engineering.

Peter Husár has been a Full Professor with the Institute of Biomedical Engineering and Informatics (BMTI), Technical University of Ilmenau, Ilmenau, Germany, since 2006, where he is currently the Head of BMTI and works in the field of research and development of technique-oriented methods and systems for early detection, diagnosis, therapy, and rehabilitation in medicine. He aims mainly at sensors for human applications in optics and applications and realization of sensors for electrical bio signals as well as sensors for measurements of relevant vital parameters.

Dr. Husár is a member for several national and international professional organizations.