Hindawi
Journal of Sensors
Volume 2018, Article ID 5268247, 16 pages
https://doi.org/10.1155/2018/5268247

Research Article

ARM-Based Universal 1-Wire Module Solution

Juraj Dudak[1], **Pavol Tanuska**[1], **Gabriel Gaspar**[2] **and Peter Fabo**[3]

[1]*Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, Jána Bottu 25, 917 24 Trnava, Slovakia*
[2]*TNtech, s.r.o., Rybárska 20, 911 01 Trenčín, Slovakia*
[3]*Research Centre, University of Žilina, Univerzitná 1, 010 26 Žilina, Slovakia*

Correspondence should be addressed to Juraj Dudak; jdudak@gmail.com

Received 10 November 2017; Revised 19 January 2018; Accepted 30 January 2018; Published 21 March 2018

Academic Editor: Eduard Llobet

Copyright © 2018 Juraj Dudak et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Sensory networks are part of a solution to monitor the required physical variables in the area of interest. Their type, the used communication protocol, plays an important role in the parameter of their complexity. One of the economical solutions is the usage of a 1-wire communication network that requires only 2 physical wires. The individual sensors or the nodes of the communication network are connected in parallel. The goal was to design and implement a universal low-power 1-wire bus module with a fully implemented 1-wire standard. As a platform for the development of such module, STM32-based microcontroller was chosen. The main advantage of this solution is the ability to utilize a sensor from a large variety of available sensors with a standardized communication interface. Our solution of the universal 1-wire module provides a single interface for sensors with different communication interfaces, while it still communicates with the standard 1-wire bus controller.

1. Introduction

Utilization of various sensors for data collection for immediate or later evaluation is the current trend in the area of industrial technology. Each sensor must belong to a specific sensory network and communicate using a specific communication protocol. A special case is sensory networks for smart sensors [1, 2] which are implemented in several different communication technologies, such as Wi-Fi, Bluetooth, and ZigBee. Not unusual are sensory nodes with multiple communication interfaces, for example, wired interface for PC connection and wireless interface for smartphone connection [3]. Currently used communication protocols include M-Bus, Profibus, Modbus, Interbus, and 1-wire. According to the ISO/OSI model, these are the application layer protocols. These protocols differ in the physical layer by the type of communication cable, the voltage levels, and the mechanical design of the used connectors. They are also distinguished by using different addressing methods in the link layer. The most noticeable differences are in the application layer, in which the communication protocol is applied. An important factor in selecting the sensory network type is the set of supported sensors, maximum communication speed, addressing method of individual sensors, the cost, and complexity of installing the entire sensory network.

Text below will deal with a sensory network based on the 1-wire standard. The principle of communication on this bus is based on the time intervals, during which the bus is in the active state. There are two basic modes: standard mode and overdrive mode [4]. Overdrive mode defines shorter time intervals, resulting in higher transmission speeds. Bus transfer rate of 1-wire communication protocol is 16.3 kbps in standard mode and 114 kbps in overdrive mode. Each sensor contains a unique 64-bit identifier, which also serves as its address. Whole bus uses only 2 wires. Communication line also serves as a power supply for individual sensors. Bus length can reach more than 500 m [5]. Main drawback of this sensory network is a small set of supported sensors. There is a large set of identifier nodes and different types of memory chips for the 1-wire bus. Only the temperature sensor DS18B20 [6] is available from the set of physical quantity sensors. Absence of various types of physical quantity sensors brings a major limitation into deploying this type of sensory networks. The fact that for the 1-wire bus there are no other

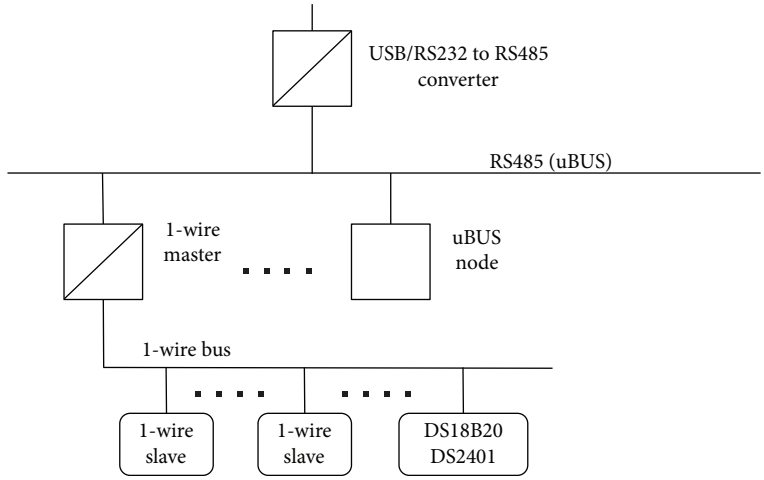


Figure 1: Sensory network example.

sensors than temperature sensor was used to design our universal sensory module solution. The following assumptions, limitations, design, implementation, and testing of universal sensory module for 1-wire bus will be described.

Part of the overall design (Figure 1) is a 1-wire master bus module that fully implements the 1-wire standard [4]. Support for bus communication over 500 m was achieved by implementing the application note [5]. The 1-wire bus master is controlled by a host computer using the uBUS [7–9] communication protocol based on the MODBUS protocol. One of the unique features of the uBUS protocol is the fact that the communication between the computer and the 1-wire bus master can be encrypted using the RSA algorithm [10]. The aforementioned master module is used for testing in Performed Tests. This paper focuses on the design and implementation of a universal slave module supporting 1-wire protocol.

2. Related Works

There are very few relevant publications regarding the problem of creating a slave module of 1-wire bus. In the practical application of the slave module, there are several publications where authors used FPGA technology. Perera et al. [11] present their "FPGA based single chip solution with 1-Wire protocol for design of smart sensor nodes," where they focus on the implementation of a smart sensor using FPGA technology. When implementing a 1-wire protocol, reference is made to status diagrams describing a method to communicate on a 1-wire bus. In the paper [12], the authors created emulation of the temperature sensor DS18B20 using FPGA technology. This sensor model was then used in "hardware in the loop" tests, where multiple instances of emulated sensors were used. Usage of FPGA technology is also discussed in Piedra's publication [13], which describes the parameters of different FPGA platforms and their capabilities to implement specific sensors depending on their parameters, such as RAM and power consumption. The issue of the smart sensor communications network is addressed by Moreno-Tapia et al. in paper [1], where they suggest the communication of reconfigurable smart sensors in industrial networks—specifically in CNC machine monitoring applications. In [14], which deals with the sensory network with communication layer using radio signals, the individual sensory nodes (sensing node) are implemented on the microprocessor STM32L1. The problem of the sensor physical connection to the microcontroller is discussed in [15], where Reverter describes the physical effects that can occur during the measurement itself. The author also suggests connections which eliminate undesired transition effects. The issue of sensor-microcontroller communication is also described by Tu in [16], from the perspective of reading and processing data on the microcontroller side according to the IEEE 1451.2 specification [17].

In several works, the authors deal with the actual use of 1-wire sensors. In papers [18, 19], they independently propose a simple wireless temperature measurement system based on 1-wire technology. Peiffer and Kruger are working [20] on an efficient communication method based on 1-wire protocol for embedded solutions. They propose the use of the CMDA addressing method and define the theoretical protocol description and error estimate and present the test results for this addressing method.

An interesting contribution is presented by Magre Colorado and Martíinez-Santos in [21], where they propose to add a cryptographic layer to the 1-wire communication protocol to increase security of smart home applications.

The 1-wire protocol is described in detail in the documentation [5, 6, 22]. These documents are the basis for all further work using 1-wire technology. Reference [22] defined a method of addressing slaves on a bus. Because each 1-wire slave contains a unique identifier, this identifier is used as the node address. Reference [6] is a description of a specific 1-wire bus slave-DS18B20 digital thermometer including timing, method of communication, supported functions, and their usage. Each 1-wire unique identifier consists of the sensor type (family code), the identifier itself, and the checksum.

According to One-Wire Bus Architecture [23] owned by Maxim Integrated, it is not allowed to create new codes for sensor types nor sensor identifiers. Similarly, existing identifiers are not allowed to be modified. Due to the conditions of

Table 1: Supported 1-wire bus slave commands.

State	Allowed operations	OWS command set
Function CMD	READ_ROM	CMDSET 1
	SEARCH_ROM	CMDSET 1
	MATCH_ROM	CMDSET 1
	SKIP_ROM	CMDSET 1
Control CMD	COPY_SCRATCHPAD	CMDSET 2
	READ_SCRATCHPAD	CMDSET 2
	READ_MEMORY	CMDSET 2
	CONVERT	CMDSET 2

the 1-wire standard patent protection for creating new slaves as defined in the document [23], only a few publications dealing with this issue have recently been published. However, most of the authors have dealt with a 1-wire [11, 18–20] master or slave simulation/emulation [12].

In spite of the strict limitations in the 1-wire standard patent protection, there have been no violations of the 1-wire standard patent in this work. The proposed module contains a standard DS2401 identifier with a unique 64-bit identification number that provides basic functionality related to addressing. This identifier also serves as the address of the entire universal module.

The basic feature of our proposal is that we do not define any new type of universal 1-wire slave that would violate the legal requirements of Maxim Integrated. According to [23], it is not allowed to create a new family code nor identifier. The 1-wire slave identifier must be defined by the manufacturer only. The proposed 1-wire bus slave module fully implements the basic 1-wire (Table 1) protocol commands, which are sufficient to implement elementary functions such as triggering the measurement on the connected sensor or reading the measured value.

3. 1-Wire Protocol Implementation

The primary basis for designing a universal 1-wire module solution was to design a universal module (hereinafter OWS—one-wire slave) allowing the use of several types of sensors, eliminating the need for external power to the module, minimizing the power consumption, and fully complying with 1-wire protocol specification and patent protection requirements. From the point of view of implementing the 1-wire protocol, it is advantageous to use a suitable microcontroller with sufficient computational power and low power consumption.

Each 1-wire bus device must implement basic commands to address and read/write data. These commands consist of a 1-byte value and are denoted by a symbolic entry. For addressing purposes, these are READ_ROM (0x33), SEARCH_ROM (0x55), MATCH_ROM (0xF0), and SKIP_ROM (0xCC) commands. For data handling, these are COPY_SCRATCHPAD (0x48), READ_SCRATCHPAD (0xBE), READ_MEMORY (0xF0), and CONVERT (0x44) commands. The functionality of these commands is explained in Section 3.1.

In Figure 2, the principal design of the OWS functionality is shown. The OWS contains the DS2401—a Maxim

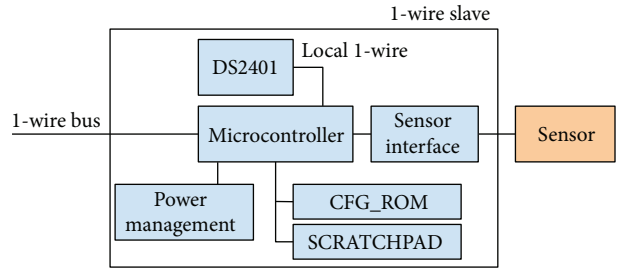


Figure 2: OWS internal structure.

Integrated 64-bit identifier. OWS acts as a standard DS2401 identifier with the base set of commands. Advanced OWS properties for communicating with the connected sensor and reading and transferring the measured value are implemented by extending the set of supported functions to standard features such as READ_SCRATCHPAD or CONVERT [6] that the original DS2401 identifier does not recognize.

3.1. 1-Wire. The 1-wire technology is based on a serial protocol that uses one data wire and one zero potential wire to transmit data. Communication with one or more slaves is always initiated by the master. The bus is inactive in state logical 1, which is provided by pull-up resistor. All slaves have an open collector driver that can change the bus status to logical 0. The communication is divided into time slots with a standard length of 60 μs, during which just 1 bit is transferred. There are four basic operations defined on the bus: write logical 0, write logical 1, read 1 bit, and reset. Communication always starts by master initiating reset operation (see Section 5 for RESET pulse details) followed by a 1-wire bus command. Each 1-wire slave has a distinct 64-bit address that consists of three parts: 8-bit FC (family code) that specifies the slave type, 48-bit identifier, and 8-bit checksum CRC8. This address is recorded into the slave during production process and cannot be modified. According to the document [22], this address is unique.

The principle of 1-wire bus communication is master/slave. There is always one master and one or several slaves on the bus. Communication starts by the master initiating the RESET pulse. Connected slaves report their presence on the bus by holding the bus level at active value—logical 0. Master is then using the search algorithm (1-wire search algorithm) [22] to determine the addresses of slaves connected to the bus. Further communication proceeds directly by addressing a specific slave or in a broadcast mode [24].

The 1-wire search algorithm principle is as follows [22]: The device that receives the SEARCH command responds by sending the first (lowest) bit of its 64-bit code. If there are multiple devices on the bus, they all respond at once. As stated in the 1-wire bus specification (devices are connected parallel to the common open collector output), the result is the logical product (AND) of all bits. After sending this bit, the master requests another bit to be sent. The device responds to this request by negating the first sent bit. From these two received bits, the situation on the bus can be derived. There are four different options described in Table 2.

Table 2: Decision-making rules in 1-wire search algorithm.

First bit	Second bit	Situation
0	0	There are more devices on the bus. These devices have different values in their codes. There was a mismatch.
0	1	There are one or more devices on the bus. These devices have a bit with a value of 0 in their codes.
1	0	There are one or more devices on the bus. These devices have a bit of 1 in their codes.
1	1	There is no device on the bus that responds to the SEARCH command.

Master now sends one acknowledgment bit. Next, only devices that have the first bit of a value the same as that sent by the master will participate in the search.

If all devices have the same bit value at the given position, the master sends this value. If they have different values (i.e., the first option—the two read bits were zero), the master must note the position at which the mismatch occurred and send either 1 or 0. Whether the master sends 0 or 1 is determined based on previous searches, mainly by the position of the last mismatch. The algorithm described below transmits zero at the first pass and transmits one at the second pass. This procedure repeats until all 64 bits of identification are read.

OWS functions are distinguished by the function code that is part of each request. These codes are divided into 2 categories depending on the status of the slave. Distribution of these functions is into function commands (function CMD) and control commands (control CMD)—Table 1. Commands in the function CMD category provide addressing a specific slave. Commands in the control CMD category implement the required slave functionality.

3.2. Universal 1-Wire Bus Module: OWS. The OWS contains 2 basic sets of commands (Table 1). A set of CMDSET 1 copies a set of DS2401 commands. Since the DS2401 is just an identifier, it does not contain any comxmands in the control CMD category. CMDSET 2 commands for communicating with connected sensors are included, reading the measured value and transferring measured values on a 1-wire bus.

The 1-wire slave may be in one of 3 states (Figure 3):

(i) IDLE

(ii) Function CMD

(iii) Control CMD

The OWS does not perform any activity in the IDLE state and is waiting for incoming communication. In function CMD state, command related to addressing is processed—searching algorithm (SEARCH_ROM), address check (MATCH_ROM), or transition to broadcast mode (SKIP_ROM). In the control CMD state, the slave performs the requested command. According to conventions, the CONVERT command is issued to start the measurement, READ_SCRATCHPAD command to read the SCRATCHPAD area which stored the measured value, and READ_MEMORY command to read the additional memory contents.

The slave in the IDLE state is in a waiting loop for the RESET pulse. If an active value appears on the bus, the slave will start detecting the RESET pulse. If the RESET pulse is too short or too long, the slave remains in the IDLE state. If the RESET pulse is evaluated as correct, the slave switches to function CMD (Figure 3). The RESET pulse length is 480 μs in the standard mode [4]. After the RESET pulse, the bus returns to the inactive state for 70 μs. After this time, all connected slaves signal their presence on the bus by changing the bus state to logical 0. If the master detects logical 0 at this time, it means that there is at least 1 slave connected on the bus and the operation is followed by an enumeration of connected slaves using the "1-wire search algorithm." Upon completion of this algorithm, the master has received all slave identifiers on the 1-wire bus. Using this information, master can address a specific slave and request for measured data from the sensor.

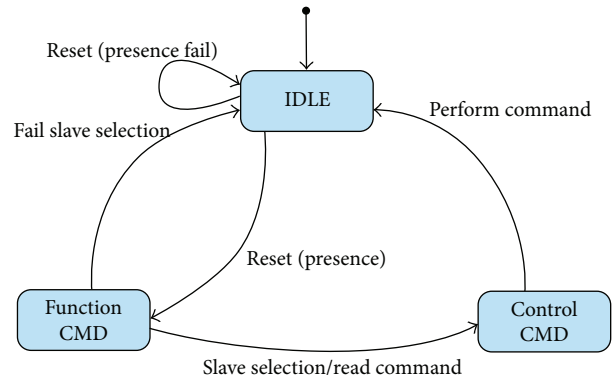


Figure 3: 1-wire bus slave state diagram.

For the OWS, flow charts were defined (Figure 4 [6], Figure 5) describing management of requests. These diagrams define the behavior of the OWS. The following text will be described processing of 1-wire commands in the OWS.

In Figure 4, a flowchart for processing commands in the function CMD state is shown. The READ_ROM command will request the slave address. Subsequently, the slave sends its address to the 1-wire bus. This command is used to verify that the slave is still present on the bus. The MATCH_ROM command is processed as follows: after issuing the command, master sends the individual bits of the slave node address to be addressed. If all sent bits match those of the slave address, the slave changes its status to the function command; otherwise, its status returns to the default IDLE state. The SEARCH_ROM command triggers the bus searching algorithm. The last command is SKIP_ROM which the slave transfers into the function CMD state without checking the address. This results in the situation, where the sent command will be executed by all connected slave nodes. It is possible to

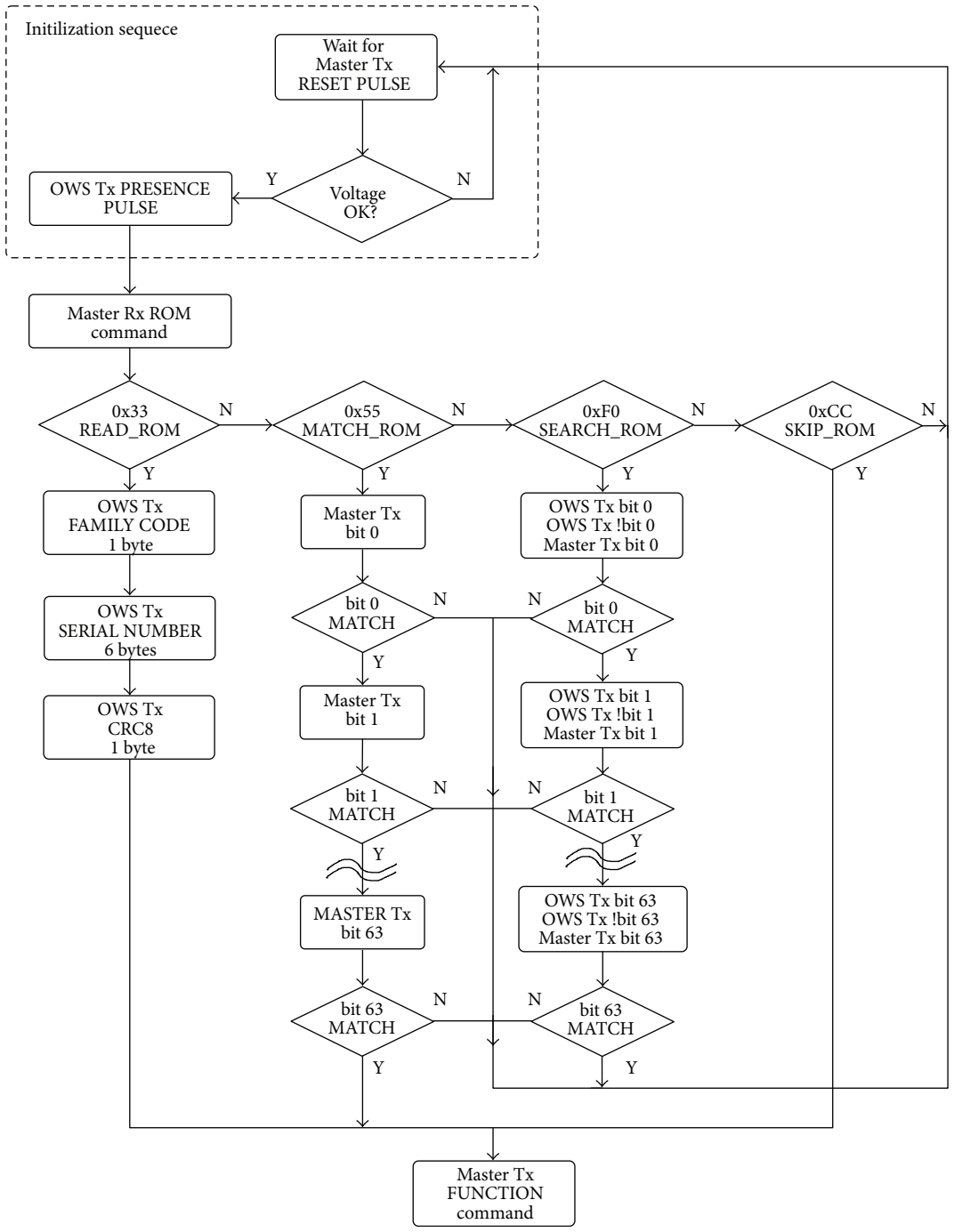


Figure 4: OWS—function commands processing algorithm [6].

switch the bus into a broadcast state when all connected 1-wire slaves are active.

In Figure 5, the command processing algorithm in the function CMD state is shown. The first command is CONVERT, which starts the measurement on the connected slave. In the case of using more power-demanding sensors, it is necessary to provide the OWS power circuits with enough power to ensure successful completion of the operation.

OWS operates in parasitic power mode, that is, it is powered directly from the 1-wire bus. OWS is during communication and measurement powered using power management circuitry including a supercapacitor. In order to secure a large enough power reserve, a 700 ms time period is used to recharge the supercapacitor serving as a temporary power source.

It is not possible to communicate over the 1-wire bus during the supercapacitor recharging period, as the bus is used to power up the supercapacitor. The DS18B20 module [6] defines the time required for the temperature conversion depending on resolution: from 94 ms for 9-bit resolution to 750 ms at 12-bit resolution. This time is therefore variable. For OWS module, the 700 ms time includes the

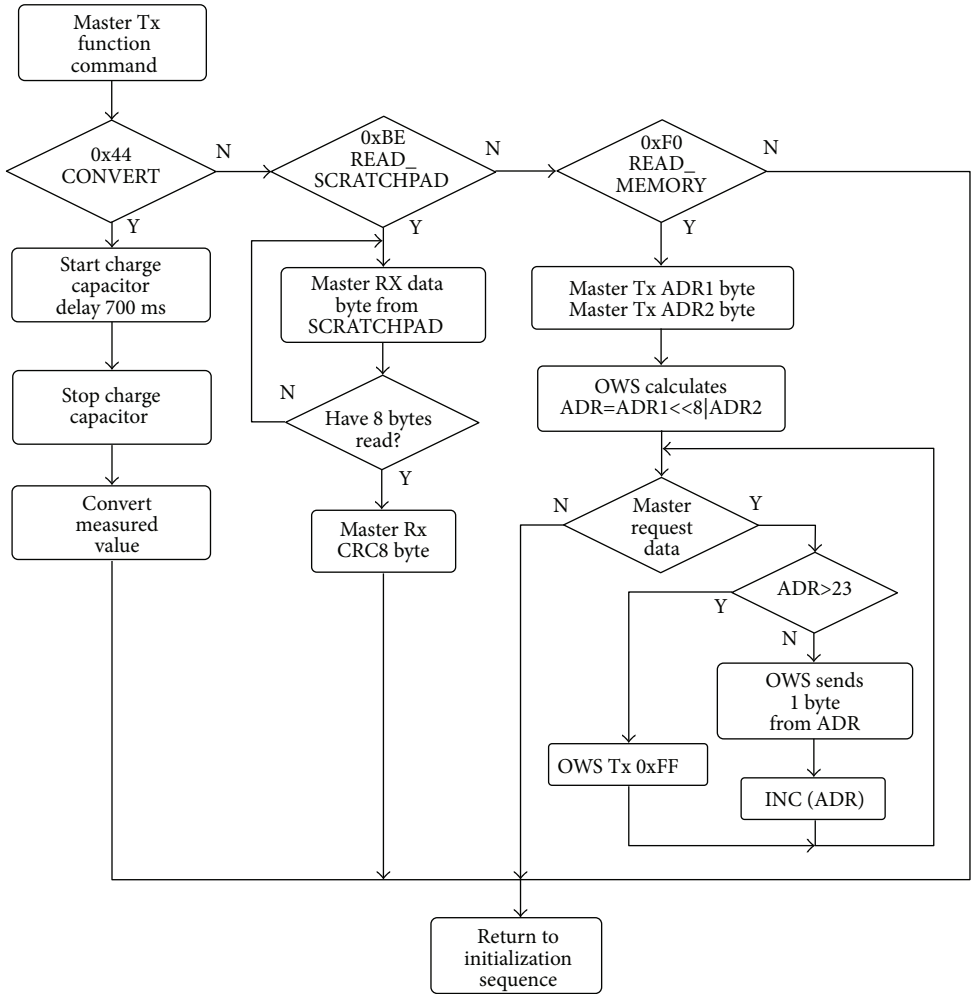


Figure 5: OWS—control commands processing algorithm.

supercapacitor charging time and the conversion of the required physical quantity itself. Charging is completed after 700 ms, and the conversion is started. The result of the conversion is stored in the SCRATCHPAD memory. The length of the conversion itself varies and depends on the type of connected sensor and its specifications. Typically, it is in the order of tens of milliseconds. To retrieve the measured value, the READ_SCRATCHPAD command is used to read 9 bytes of the measured value from the SCRATCHPAD area and send it to the 1-wire bus. Frame format for the OWS solution is described in Table 3. The length of the selected 9-byte data frame was based on the DS18B20 standard temperature sensor data frame [6].

Data frame formats for various measured physical quantities are displayed in Table 3: relative air humidity, atmospheric pressure, solar radiation intensity, electrical voltage, electrical current, electrical resistance, and binary inputs. Data frame for the relative humidity measurement contains the value of measured humidity in the first 2 bytes, and the other 2 bytes represent the temperature.

In the data frame for measuring atmospheric pressure and solar radiation intensity, the first 4 bytes represent the measured value. When using a sensor that measures electrical voltage, electrical current, and electrical resistance, it is possible to implement two 24-bit converters—the frame contains up to 2 values, each of 3B. The data frame for binary values allows reading 6 bytes and 48-bit information, respectively. The 6th byte in each frame sequence determines the type of data frame. V_bus item represents the value of the OWS power supply voltage. In specific cases, triggering of the measurement may be energy intensive, so it is necessary to check the power voltage level of the entire OWS. The last item is the CRC8 checksum.

Table 3: Format of data frame with measured values.

Type/byte	0	1	2	3	4	5	6	7	8
Humidity	H_1	H_0	T_1	T_0	X	X	0x1	V_bus	CRC8
Pressure	P_3	P_2	P_1	P_0	X	X	0x2	V_bus	CRC8
Sunlight	L_3	L_2	L_1	L_0	X	X	0x3	V_bus	CRC8
Voltage	$V1_2$	$V1_1$	$V1_0$	$V2_2$	$V2_1$	$V2_0$	0x4	V_bus	CRC8
Current	$I1_2$	$I1_1$	$I1_0$	$I2_2$	$I2_1$	$I2_0$	0x5	V_bus	CRC8
Binary	B_1	B_2	B_3	B_4	B_5	B_6	0x6	V_bus	CRC8
Resistance	$R1_2$	$R1_1$	$R1_0$	$R2_2$	$R2_1$	$R2_0$	0x7	V_bus	CRC8

The last OWS command is READ_MEMORY. The OWS has implemented ROM memory (Figure 2, CFG_ROM), where additional information about the type of the measured quantity, the physical unit of the measured quantity, the sensor name, and the limits of the measured physical quantity is stored. The size of this memory is 24 bytes, and its contents are defined in the firmware and cannot be changed later. This additional information is important from the point of view of implementing the IEEE 1451.0 [25] and IEEE Std 1451.2 [17] standards in the 1-wire bus master. The 1-wire bus master automatically detects the presence of slaves on the bus and provides information about the number of sensors, their type, measured physical quantities, bus state status information (active/inactive), and others to the host computer. OWS contains this information in a dedicated part of the memory and can provide it to the master.

4. Universal 1-Wire Module Design

When designing the OWS, the following solution requirements were defined:

(1) Fully implemented 1-wire standard

(2) Power independent solution

(3) Minimal power consumption

(4) Solution universality—capability of connecting devices communicating through known standards such as UART, I2C, and SPI to the slave

(5) The possibility of implementing advanced processing methods on the data acquired from a sensor (filtering, transformation, etc.)

Requirement 1 was described in more detail in Section 3 of this paper.

4.1. Power Independent Solution [24]. All existing 1-wire slaves are able to operate in parasitic power mode, in which they in the IDLE state use the bus potential to charge the supercapacitors, which then serve as a power supply during communication. Supercapacitor refers to a capacitor with sufficient capacity, which can supply power to the slave for a sufficiently long time period. It is when the slave proceeds with communication or measurement. In Figure 6, it is labeled as *SuperCap*. The same principle is used in the design of the OWS. From a power point of view, the OWS has to meet the following prerequisites:

(i) If the capacitor is discharged and the OWS is connected to a 1-wire bus, it has to be charged without the intervention of the control logic and the control microcontroller, respectively.

(ii) Once the minimum supply voltage necessary to run the control microcontroller is reached, the process of charging is taken over by the microcontroller logic.

(iii) If an active level (logical 0) is detected on the bus, the charging of the capacitor is immediately terminated and the OWS switches to the active mode in which it communicates with the 1-wire bus master.

(iv) When the communication is over, the OWS switches to the reduced power mode and activates the charging of the capacitor.

4.2. Minimal Power Consumption[24]. The proposed OWS consists of a sensor that provides measurement of the desired quantity (humidity, pressure, light intensity, etc.), a control microcontroller, 1-wire bus controller, power supply management of the entire OWS, and a capacitor in the role of a power source. To ensure the greatest possible reliability, it is necessary to minimize the power consumption of the entire equipment. This is achieved by using the power-saving modes of the control microcontroller.

The OWS may be found in two modes: active mode and standby mode. In active mode, the current consumption can be reduced by decreasing the clock frequency of the control microcontroller. In passive mode, the consumption is minimized by turning off unnecessary peripherals and switching the microcontroller into standby mode.

4.3. Solution Universality [24]. The next requirement was to design the OWS allowing connection of standardized sensors. Under the "standardized," the sensor communication interface is understood. It is possible to connect the following to the OWS:

(i) Sensors communicating using the I2C interface, such as sensor of visible and UV light Si1145 and atmospheric pressure sensor BMP180

(ii) Sensors communicating using the SPI interface, such as a 24-bit A/D converter AD7195

(iii) Sensors communicating using the I/O interface of the microcontroller ports, such as AM2302 humidity sensor, which uses its own serial communication protocol

Design versatility is ensured already in firmware design. When adding a new sensor, it is necessary to define the following functions: Interface_Init()—initialization of the communication interface and pins of the used microcontroller and Sensor_Init()—initialization procedure for a particular sensor. The measurement procedure itself may be more complicated because in several cases it included a few readings of sensor values among which there must be a certain delay. To ensure a minimum power consumption, the LP_Delay() (low power delay) function that is used at these waiting delays was created. The Sensor_Measure() function must be implemented at the level of the OWS application itself. This function is used to stop the supercapacitor charging and measure on the sensor at the correct time points. The last feature is Sensor Read(), which ensures the correct formatting of the response data following Table 3.

4.4. Implementation of Advanced Data Processing Methods. The 1-wire bus transmission rate (16.3 kbps) is significantly

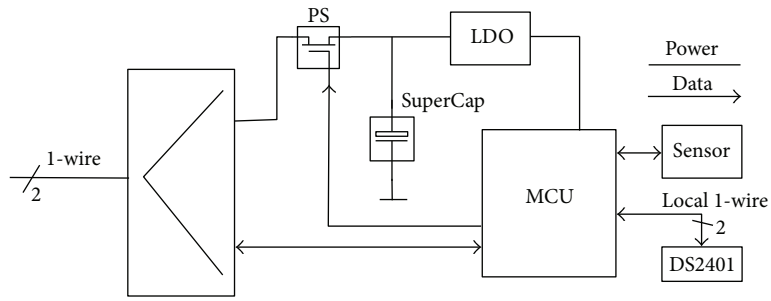


Figure 6: Principal diagram of hardware design.

lower than the sensor transmission rate (ranging from 100 kbps to 10 Mbps). This allows the measured data to be processed before sending them to the 1-wire bus. Basic operations for data handling include moving average and low-pass and high-pass filters. Due to the way the OWS operates, these operations will be performed in a batch at defined intervals or over a set of data of the specified size, as the active sensor increases the current consumption of the entire OWS.

4.5. Hardware Design. In Figure 6, a principled diagram of OWS is shown. The first part of the diagram includes a data and power distributor. To ensure continuous power supply, a controlled power switch (PS—power switch), a high-capacity electrolytic capacitor (supercapacitor), and a linear voltage regulator (LDO—low dropout) powering the microcontroller and the connected sensors were used. The microcontroller controls the supercapacitor charging at a time when no communication is in progress.

According to the specification, the OWS has to begin charging the capacitor after being connected to the 1-wire bus without any intervention of the control electronics. This condition occurs when the capacitor is completely discharged. The microcontroller is inactive after connecting the OWS module with discharged supercapacitor to the bus. After reaching the minimum voltage on the supercapacitor required for the control microcontroller, the controlled charging is activated. The PS switch (Figure 6) is switched on, allowing charging at the maximum speed.

In order to reduce the total power consumption of the entire device, microcontroller's internal A/D converter will be used to monitor supercapacitor voltage levels. At the time when the voltage on supercapacitor is not high enough to start the microcontroller, supercapacitor charging is controlled by an analog circuit based on the voltage divider principle.

5. Universal Module Implementation on the STM32 Platform

During OWS implementation, we used the ARM microcontroller of the Cortex M0 and Cortex M0+ family, respectively. The reasons for this selection are as follows: sufficient performance, wide range of microcontroller clock settings, multiple microcontroller power-saving modes, miniature design, and low cost. The first solution was the OWS implementation on STM32F030 microcontroller. It is a 32-bit microcontroller based on the Cortex M0 architecture. The maximum clock rate is 48 MHz, 4 kB RAM, and 16 kB FLASH. Microcontroller supports SLEEP, STOP, and STANDBY modes. The second solution was the implementation on STM32L031 microcontroller with the Cortex M0+ architecture. This microcontroller belongs into the ultra-low power category, which reduces the energy consumption of the whole solution. It has similar parameters, with additional power modes: LOW POWER RUN and LOW POWER SLEEP.

The parameter most affecting the OWS energy consumption is the choice of the microcontroller clock frequency. The goal is to reduce the microcontroller clock frequency to the lowest frequency suitable for OWS reliable operation. There are 2 limitations when searching for the lowest limit of the clock frequency:

(1) 1-wire bus timing

(2) Speed of transition from energy-saving mode

When communicating, the slave has to measure the exact time interval in the 1 μs range. Due to the architecture used (Cortex M0, M0+) and the libraries used for the OWS firmware programming, the clock frequency was set to 16 MHz. There is no action conducted in the standby mode, and therefore, the microcontroller can be put into the minimum power mode. When initiating a communication, that is, when detecting a RESET pulse that lasts 480 μs, it is necessary to wake the microcontroller as quickly as possible so that the correct RESET pulse length can be detected.

The time of transition from standby to the standard RUN mode consists of the t_WUSTOP wake-up time and the t_CLOCKCFG, which is time required to reconfigure the RCC (reset and clock control) block, that ensures that the frequency is set correctly for all parts of the microcontroller, initialization of the selected peripherals, and check of the voltage level on 1-wire bus. According to [26] t_WUSTOP is 5 μs, and the measured time t_CLOCKCFG is 200 μs at the 16 MHz clock frequency. Time t_CLOCKCFG is always the same at a given microcontroller frequency. In order to correctly detect the RESET pulse, it is therefore necessary to measure the remaining time t_RESET' = 480 μs - (t_WUSTOP +t_CLOCKCFG) = 275 μs. Due to the additional processing overhead (system interrupts of the microcontroller core, calling functions in the processing itself), it implemented a 70 μs tolerance for the time t_RESET'. When using a clock frequency less than 16 MHz, it was not possible to detect the correct RESET pulse length because the time

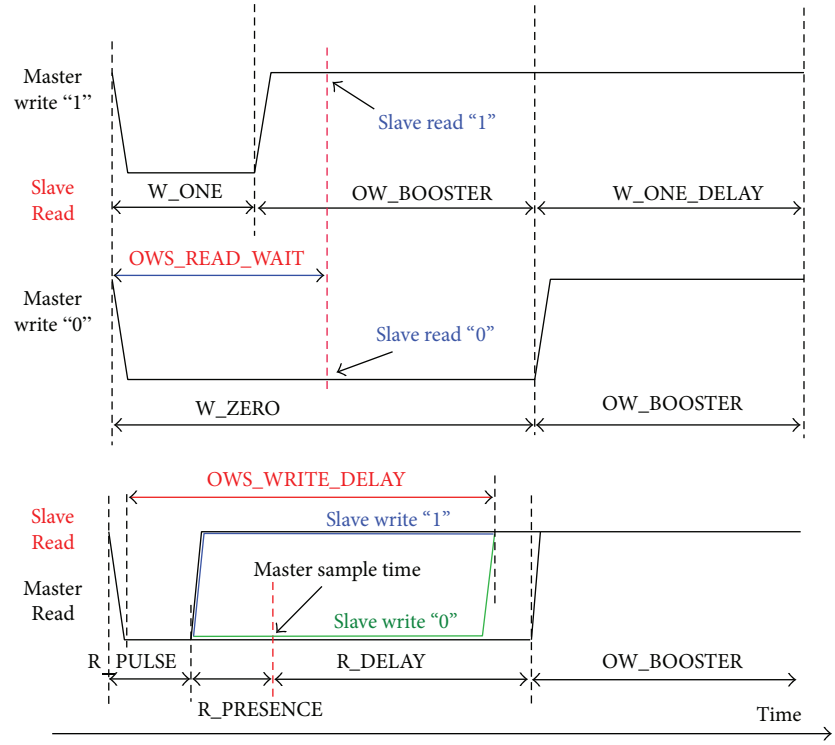


Figure 7: 1-wire protocol elementary operations.

t_CLOCKCFG would approach the 480 μs time, which is the RESET pulse duration.

5.1. Using Low-Power Mode. When OWS communicates with the 1-wire bus master, the RUN mode is used. The microcontroller is clocked at 16 MHz and has all the necessary peripherals for its operation connected. When the communication terminates, the microcontroller switches into standby mode. The STM32F0xx microcontroller has 3 power-saving modes: SLEEP, STOP, and STANDBY. The power consumption in these modes is 7.3 mA, 19 μA, and 2 μA, respectively. Only the core of the microcontroller is stopped in SLEEP mode, and in the STOP mode, the core is powered off, the peripheral clock is stopped, and the RAM remains active. In STANDBY mode, only the module responsible for the awakening of the microcontroller remains active. Consumption in SLEEP mode is too high, so this mode will not be used. Consumption in STANDBY mode is the lowest, but it takes too long to wake the microcontroller from this mode. The suitable power-saving mode is STOP mode.

5.2. Communication Implementation. The principle of 1-wire bus communication is master/slave. There is always just one master and one or several slaves on the bus. Communication starts by the master issuing the RESET pulse. There are 4 basic operations in the communication: write logical 0, write logical 1, read one bit, and reset (reset detectionfor slaves). In Figure 7, these operations from the point of view both the master and slave are shown. Typical values of the timing parameters in Figure 7 are shown in Table 4 [4].

The 1-wire standard in the document [4] specifies bus timing for writing and reading 1 bit from a master view. To implement a 1-wire slave, it is essential to define timing for inverse operations. Writing 1 bit on the bus by master represents a 1-bit reading for the slave.

For the 1-wire slave node, we defined the following time constants: OWS_READ_WAIT and OWS_WRITE_DELAY (Figure 7).

The algorithm of 1-bit reading and writing is in Figure 8. The OW abbreviation stands for the logical level of the 1-wire bus. One-bit reading begins by detecting the logical 0 level on the 1-wire bus. If this level is not detected in 400 μs, the ERROR flag is set and the reading is terminated. Otherwise, the detection continues: the bus remains in the logical 0 state for time W_ONE or W_ZERO (Figure 7), depending on whether the master sends 1 or 0. After detecting the logical 0 level, the OWS is waiting for time OWS_READ_WAIT with condition: W_ONE < OSW_READ_WAIT < W_ZERO. At the end, it is waiting for the logic level of the 1-wire bus change to inactive level logical 1. If the value does not change to logical 1 in 160 μs, the ERROR flag is generated and the reading is invalid.

The bit writing algorithm is less complicated: when writing the logical 0, the slave node holds the bus level at log. 0 (DRIVE_OW_LOW) for time OWS_WRITE_DELAY; when writing logical 1, the level is increased to the value of logical 0—Figure 7. The OWS_WRITE_DELAY time must meet the following condition:

$$\begin{aligned} &\text{MASTER\_SAMPLE} < \text{OWS\_WRITE\_DELAY} \\ &\qquad < (\text{R\_PULSE} + \text{R\_PRESENCE} + \text{R\_DELAY}), \end{aligned} \tag{1}$$

Table 4: Parameters of 1-wire communication timing.

Parameter	Description	Time (μs)
W_ONE	Master write log. 1—drive bus low.	6
OW_BOOSTER	After releasing bus, master activate force "1."	10
W_ONE_DELAY	After this time, master release bus.	54
W_ZERO	Master write log. 0—drive bus low.	60
R_PULSE	Master send read pulse—drive bus low, then release bus.	6
R_PRESENCE	Time when master sample level of 1-wire bus.	9
R_DELAY	Time to activate force "1" by the master.	45

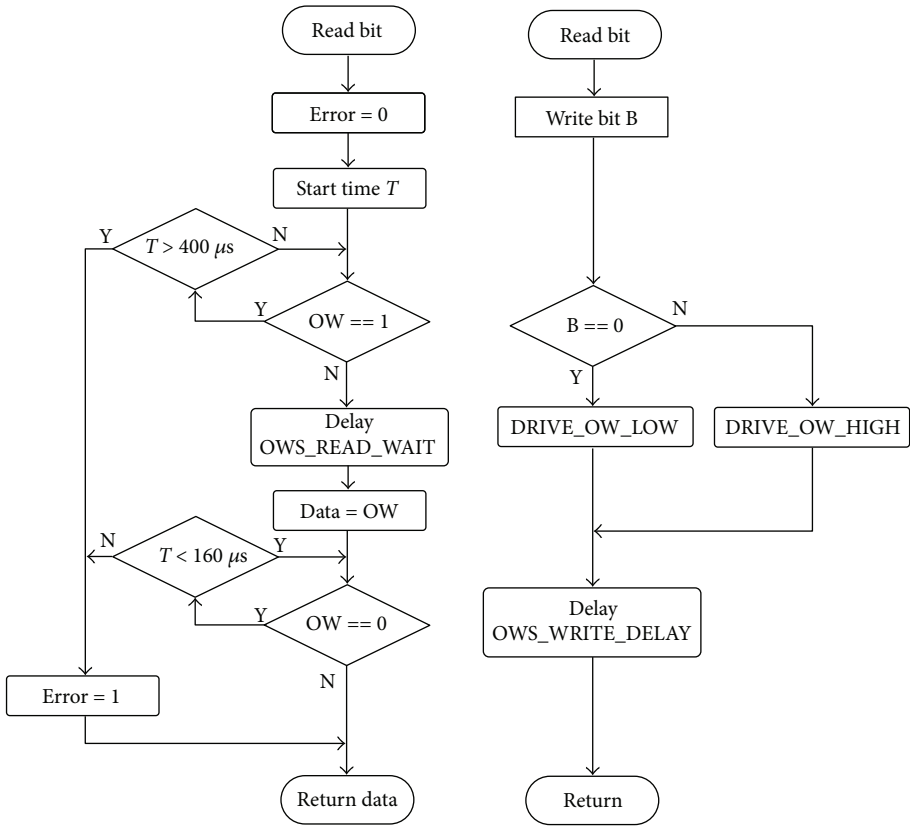


Figure 8: 1-wire bus algorithm for reading and writing 1 bit.

where MASTER_SAMPLE is the time period, when the master reads the value from the bus. OW_BOOSTER is a special interval when the master activates the 1-wire bus controller, which instantly raises the level to logical 1. During OW_BOOSTER time period, slaves are not allowed to change the 1-wire bus level.

The following text is the main source code snippet in C language for the STM32F030x microcontroller (Sourcecode 1). The microcontroller is set that after detecting the logical 0 level on the 1-wire bus switches from the STOP mode to the RUN mode. Subsequently, the supercapacitor charging (CAPACITOR_CHARGE_STOP) is deactivated, followed by the communication on the 1-wire bus (OW_slave). When the communication operation is completed, the OWS returns to STOP mode (OWS_Stop).

5.3. Power and OWS Start-Up. The OWS does not need any external power source for its operation. As a power supply in the IDLE mode, it used the communication bus voltage in parasitic power mode (parasite power). An important part of the OWS hardware solution was to design a way to control supercapacitor charging. With fully discharged supercapacitor, the OWS has to start charging automatically until the voltage reaches the level at which the microcontroller will start. After reaching this voltage level, the microcontroller takes over the charging process. In Figure 9, the charging current is 12 mA in the first approximately 17 s. If the supercapacitor voltage exceeds the threshold level at which the microcontroller starts operating, the controlled supercapacitor charging is switched on. At this point, the charging current reaches 80 mA, which is caused by switching on the PS

```
while (1) {
        if( OW == 0 ) {                          // detection of RESET pulse
            CAPACITOR_CHARGE_STOP;    // stop charging of supercap
            OW_slave();                          // processing of Function CMD
            if(ow_status != OW_STATE_IDLE)   {
                OW_slave();                  //processing of Control CMD
            }
            // determine sleep time
            last_tick = HAL_GetTick() + CHARGING_DELAY;
        }
        current_tick = HAL_GetTick();
        if(current_tick > last_tick) {
            OWS_Stop();                          // go to STOP mode
        }
  }
```

Sourcecode 1: Snippet in C language for the STM32F030x microcontroller.

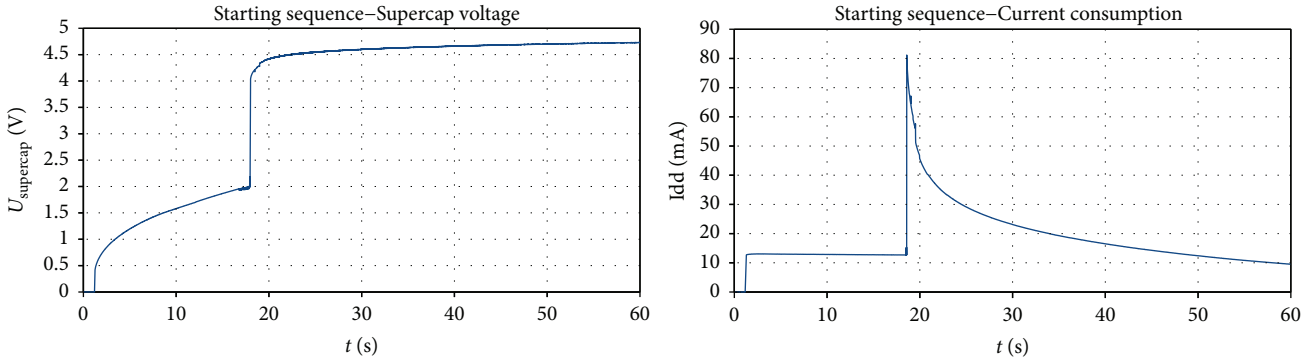


Figure 9: Current consumption and voltage on supercapacitor during starting sequence. STM32F030 was used.

switch itself. The current drops with exponential dependence and stays at 330 μA, which is the consumption of the microcontroller and additional circuitry in STOP mode. The value of the supercapacitor was 330 mF. The graphs (Figure 9) are for the OWS module with the STM32F030 microcontroller. When using a low-power microcontroller, the graph differs by significantly shortening the first phase of the supercapacitor charging (part with the constant current consumption) to approximately 1 s.

The full charge time of the supercapacitor is approximately 3 minutes. This time depends on the capacity of the supercapacitor. The charging current is defined by

$$I_{cc} = I_0 e^{(-t/RC)}. \quad (2)$$

In implementation, 2 microcontrollers were selected: STM32F030 belonging to general-purpose microcontrollers and STM32L031, which belongs to the low-power category. The charging process is similar for both microcontroller, except that the STM32L031 can boot at a lower supply voltage than the STM32F030. In Figure 10, a comparison of the selected microcontrollers' power consumption is shown. As a test, command used the CONVERT command, which starts the sensor measurement and the READ_SCRATCHPAD command, which reads the measured value. At the beginning, the microcontroller is in STOP mode and the communication begins at 0.5 s. The OWS switches to RUN mode. Processing CONVERT command is as follows: since this command may also come at a time when supercapacitor does not have enough energy to provide a reliable power supply for a period of 600 ms, the microcontroller switches to LP_RUN and activates supercapacitor charging. After this time, it switches to RUN mode, following by reading the sensor value. This is represented by the first peak in Figure 10. The second peak is the processing of the READ_SCRATCHPAD command when the measured value is sent to the bus. This is followed by the transition to the STOP mode.

The power consumption of used microcontrollers is in the IDLE state (Figure 3, IDLE) in STOP mode at the same level. Differences are in RUN mode when the STM32L031 microcontroller has a lower power consumption by approximately 500 μA to 800 μA.

The amount of used energy can be expressed as an integral:

$$E = \int_0^{t_0} U(t)I(t)dt. \quad (3)$$

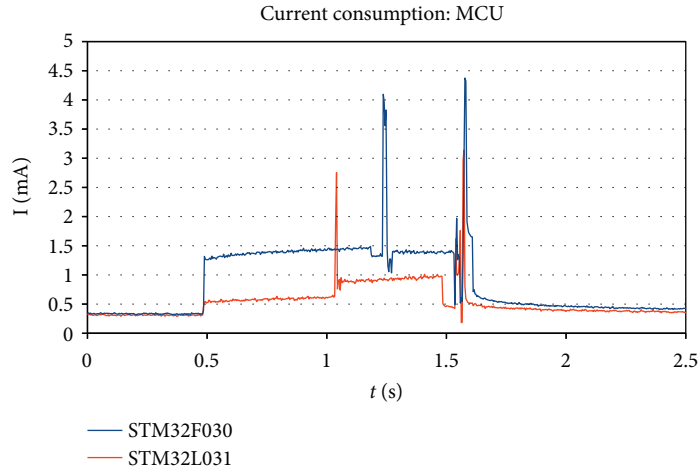


Figure 10: Current consumption comparison in RUN mode.

For constant supply voltage, energy demand can be expressed as power consumption over time, that is, as an electrical charge:

$$Q = \int_0^{t_0} I(t)dt. \quad (4)$$

Applying the relation (4) to the measured values from the graph in Figure 10, we get Q_F = 2.733 mAs, Q_L = 1.81 mAs. The Q_F or Q_L values are the electrical charge for STM32F0 and STM32L0 microcontrollers, respectively. Therefore, the power consumption with the STM32L031 microcontroller is 66.38% of the consumption with the solution using the STM32F030 microcontroller.

6. Performed Tests

The proposed solution of the OWS has been tested in a real 1-wire sensory micronetwork. The following parameters were set as a condition for the successful evaluation of the tests:

(i) Communication without errors

(ii) Compatibility with standard 1-wire slaves

(iii) Correct functionality even when using long bus

6.1. Test A. Tests were carried out using different 1-wire bus lengths: 150 m, 300 m, 450 m, and 500 m. 40 DS18B20 sensors and 7 OWSs were connected to the bus. Deployment of DS18B20 sensors on the bus was in blocks of 10, 20, and 40 units directly on the bus or using a 3 m branch. OWSs were placed in blocks of 3, 4, or 7 units.

In Figure 11, the 1-wire bus communication course using a bus with length of 150 m (Figure 11(a)) and 500 m (Figure 11(b)) is displayed. The measurements were made on the side of the 1-wire bus master. Since the signal on the 1-wire bus is distorted depending on its length, a logic level comparator was added in the master module for correct signal processing. The green color represents signal from the 1-wire master comparator. This signal was measured directly on the printed circuit board of the master. The blue color shows the signal level on the 1-wire bus. This value was measured on a 1-wire bus data line of a 1-wire master. The first pulse shown in the graph in Figure 11 is the last bit of the SEARCH_ROM command sent by the master node, followed with a request to read a bit that ends at about 150 μs (Figure 11). Subsequently, the slave node writes 1 bit to the bus. In this case, it is log. 1. Writing bit 1 is implemented so that the OWS releases the 1-wire bus and the bus voltage increases to the maximum value of 5 V, representing log. 1. In the case of a long bus test (Figure 11(b)), the reflection of the signal (blue waveform) from the open, electrically unmatched end of the bus is clearly seen.

6.2. Test B. The power consumption of the OWS in the IDLE state is 330 μA (Figure 10). This current is supplied from the 1-wire bus in regular operation. During communication, this current rises to 1 mA with STM32L0 to 1.5 mA with the STM32F0 microcontroller. A supercapacitor with the capacity of 330 mF is used as a power source. The following test was aimed at measuring the consumption of the OWS in its IDLE and ACTIVE states.

The maximum current that can be used is derived from the 1-wire bus controller on the master side. The 1-wire master bus can provide short-term 200 mA current. The master module cyclically measures the 1-wire bus current, and no communication is allowed when the 25 mA is exceeded. It is supposed that one of the connected supercapacitors is recharging.

The capacitor value was derived from the following assumption. OWS module could also use sensors with higher current consumption (e.g., strain gauges and external A/D converter). When multiple OWS modules are connected to a 1-wire bus, all modules are active during communication and supercapacitor recharging is greatly limited. From the view of power consumption, it is most difficult to apply 1-wire search algorithm. Supercapacitor charging is triggered for 700 ms before measuring on energy-demanding sensors, which is not possible with the 1-wire search algorithm.

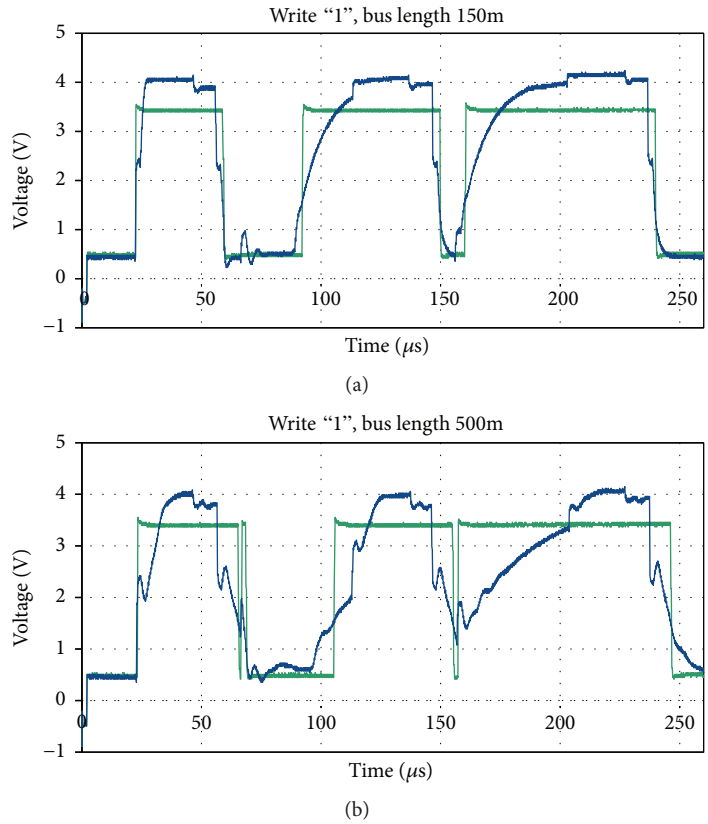


Figure 11: Communication test with different 1-wire bus lengths: (a) 150 m and (b) 500 m.

The 1-wire search algorithm itself as well as the reading of the measured values can take up to several seconds. Supposed charging of super capacitors will not be possible for 10 s. The estimated OWS load is 100 Ω. The voltage rating of the supercharger is 4.8 V. The required minimum voltage is 3.6 V. To calculate the value of supercapacitor, we used

$$U_c = U_{\text{supercap}}\, e^{-(t/\text{RC})}. \tag{5}$$

From this formula, we will output capacity C:

$$C = -\frac{t}{R \ln U_c / U_{\text{supercap}}}. \tag{6}$$

After setting the values to (6), we get $C = 347$ mF. The specific value of the 330 mF capacity was chosen as the closest available value.

6.3. Test B.1. The tests were always based on an assumption that the supercapacitor is fully charged. The test begun after disconnecting the OWS of the 1-wire bus. Figure 12 shows the voltage state of the OWS supercapacitor (Vcap) in STOP mode that is at the time when there is no communication on the bus. The initial voltage value was Vcap = 4.8 V. The test was performed for 45 minutes.

Using a STM32F0 microcontroller, the Vcap voltage drops rapidly after 40 minutes. According to [26], the minimal supply voltage of this microcontroller is Vdd = 2.4 V, and below this level, the microcontroller goes into a permanent reset state. The STM32L0 microcontroller belongs to the "ultra-low power" category. The minimal supply voltage of this microcontroller is Vdd = 1.8 V [27], which allows it to operate reliably even at lower voltages. This test was focused on the ability to maintain the required voltage on the supercapacitor, even when disconnected from the 1-wire bus. After disconnecting the OWS from the bus, the module is in the STOP state. If the OWS reconnects to the 1-wire bus with a nonzero voltage level, its charging will be much faster.

6.4. Test B.2. Another OWS test aimed at power consumption was monitoring the voltage level of Vcap during communication without the possibility of controlled charging of the supercapacitor. Initial test conditions were set as follows: fully charged supercapacitor, test duration 10 min.

To keep the OWS in RUN mode during the test, a RESET pulse was sent to the 1-wire bus every 10 ms, followed by the SEARCH_ROM command to ensure that the STOP standby mode was not activated. When using the STM32F0 microcontroller, the Vcap voltage dropped below 2.4 V, which is the limit value of the correct functioning of the microcontroller core after 4 min (240 seconds). After 5 minutes (300 seconds), the voltage level Vcap dropped to 2 V, the microcontroller was reset state and the OWS lost its functionality. Because the STM32L0 microcontroller belongs to the ultra-low power category, the power supplied from the 1-wire bus is sufficient to operate. The result of the test is in Figure 13.

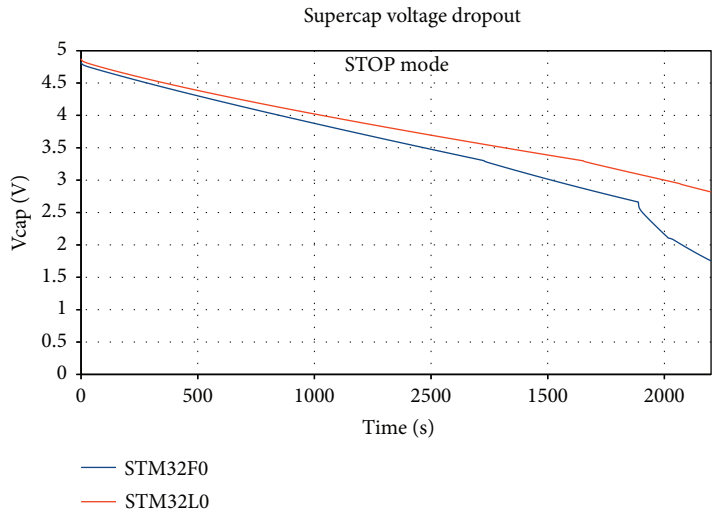


Figure 12: Supercap voltage dropout in STOP mode.

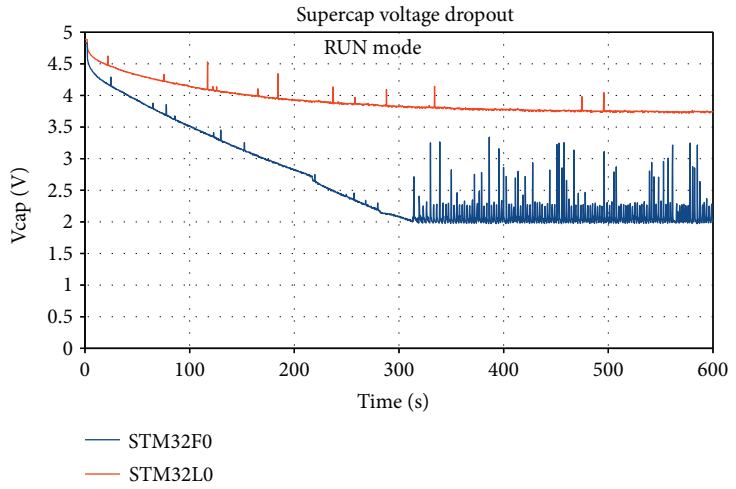


Figure 13: Supercap voltage dropout in RUN mode.

Approximately 7 minutes later (420 s), the voltage at the supercapacitor remained at Vcap = 3.7 V. Furthermore, the Vcap value does not decrease anymore. This is due to property of the supercapacitor charging circuitry. In case, the controlled charging is not activated, the supercapacitor is charged with a current that is proportional to the voltage difference on the 1-wire bus and the voltage on the supercapacitor. This property is used when OWS starts. Due to the low power consumption, this charging current is sufficient to keep OWS functional.

7. Discussion

From the energy efficiency tests, we can clearly state that the use of the STM32L031 microcontroller eliminates the condition, where the OWS will not function due to the time-consuming communication (presence of several dozen sensors) on the 1-wire bus and the low voltage level on the supercapacitor. As demonstrated in the last test, the OWS can be partially recharged even during communication when the controlled charging is deactivated. This is accomplished by designing a charging circuit that, when the controlled charging is deactivated, delivers supercapacitor current proportional to the voltage difference on the supercapacitor and the 1-wire bus.

Test B.2 conditions can only rarely occur in real-time operation, because only RESET and SEARCH_ROM commands were used in this test. In practice, the SEARCH_ROM command is followed with a command from the control CMD (Table 2), for example, CONVERT. This defines that prior to measuring on the sensor, controlled supercharger charge is activated for 700 ms (Figure 5), which is sufficient to fully charge the supercapacitor. Test B.2 was run repeatedly with modified conditions: RESET, SEARCH_ROM, CONVERT, and READ_SCRATCHPAD were cyclically sent to the 1-wire bus. Part of the CONVERT command is a

700 ms pause when the supercapacitor recharges. In this modified test, the voltage drop on the supercapacitor was $\Delta V = 0.05$ V and the voltage did not drop any further.

8. Conclusions

The paper presented the design and implementation of a universal 1-wire module. This module is specific because it works in parasitic power mode and therefore does not require external power. Proposed solution consists of the control microcontroller STM32F030F4 or STM32L031F6, a supercapacitor that serves as a power supply, a 1-wire bus controller, and a sensor that measures the required physical quantity. During design and implementation phase, a great emphasis was placed on the low power consumption of the entire module. The need of low power consumption is due to the way the modules are used: dozens of sensors can be connected to the 1-wire bus. The more energy-efficient solution is one using the STM32L031 microcontroller, because in RUN mode it has 33.62% (based on (4), Section 5) lower power consumption than the one with STM32F030 microcontroller.

The uniqueness of this solution lies in the very implementation of the universal 1-wire module of 1-wire bus. For the use of different sensor types, other than a temperature sensor, there is no solution like a 1-wire slave, available. There was no 1-wire protocol modification in the implementation nor any new family code or new 1-wire function codes were added. The solution is fully compatible with the 1-wire standard. The proposed OWS contains the standard DS2401 identifier, which is a unique identifier with a set of READ_ROM, SEARCH_ROM, and MATCH_ROM commands. OWS implementation extends this set with the SKIP_ROM, COPY_SCRATCHPAD, READ_SCRATCHPAD, READ_MEMORY, and CONVERT commands. OWS has standard interfaces for connecting external sensors. Measurements on these sensors are triggered using the standard CONVERT command. The OWS contains a ROM that stores additional information about the type of the measurement, the physical unit of the measurement, the name of the sensor, and the limits of the measured physical quantity. This information is used by the 1-wire master to provide a detailed overview of 1-wire bus sensors. The OWS was tested in real-time operation along with other sensors on a 500-meter 1-wire bus. After successful tests, OWSs were permanently put into operation. There are 15 OWSs with relative air humidity sensors with another 100 DS18B20 sensors installed in the monitoring system of temperature and relative air humidity of production halls and office spaces. This solution works flawlessly in continuous operation, where data are read every 10 minutes for more than 1 year.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This publication is the result of implementation of the project: "UNIVERSITY SCIENTIFIC PARK: CAMPUS MTF STU - CAMBO" (ITMS: 26220220179) supported by the Research & Development Operational Program funded by the EFRR. This publication is the result of the project implementation: "Priemyselné výskumno-vývojové centrum "TRENZA"" (ITMS: 313011B622) supported by the Research & Innovation Operational Programme funded by the ERDF. This publication is the result of the project implementation: "Výskum a vývoj novej generácie ekologických veľkokapacitných systémov výroby a skladovania obnoviteľnej energie" (ITMS: 313011B778) supported by the Research & Innovation Operational Programme funded by the ERDF.

References

[1] S. V. Moreno-Tapia, L. A. Vera-Salas, R. A. Osornio-Rios, A. Dominguez-Gonzalez, I. Stiharu, and R. D. J. Romero-Troncoso, "A field programmable gate array-based reconfigurable smart-sensor network for wireless monitoring of new generation computer numerically controlled machines," *Sensors*, vol. 10, no. 8, pp. 7263–7286, 2010.

[2] L. Ruiz-Garcia, L. Lunadei, P. Barreiro, and I. Robla, "A review of wireless sensor technologies and applications in agriculture and food industry: state of the art and current trends," *Sensors*, vol. 9, no. 6, pp. 4728–4750, 2009.

[3] J. Chen, Y. Lin, R. Gong, J. Guo, X. Zeng, and J. Liu, "Embedded based miniaturized universal electrochemical sensing platform," *Journal of Sensors*, vol. 2016, Article ID 1254368, 8 pages, 2016.

[4] "1-wire communication through software: APPLICATION NOTE 126. Maxim integrated. Maxim Integrated Products, Inc.," [cited 2015-11-09], 2002, https://www.maximintegrated.com/en/app-notes/index.mvp/id/126.

[5] "Guidelines for reliable long line 1-wire networks: Appnote148. Maxim Integrated. Maxim Integrated Products," [cited 2017-04-11], 2008, https://www.maximintegrated.com/en/app-notes/index.mvp/id/148.

[6] "DS18B20: programmable resolution 1-wire digital thermometer," [cited 2017-04-11], 2015, https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[7] J. Dudak, M. Skovajsa, and I. Sladek, "Proposal of a communication protocol for smart sensory systems," in *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*, pp. 107–112, Brno, Czech Republic, 2014.

[8] J. Dudak, G. Gaspar, and G. Michalconok, "Extension of 1-wire measuring system SenSys," in *Proceedings of 15th International Conference MECHATRONIKA*, pp. 1–4, Prague, Czech Republic, 2012.

[9] J. Dudak, *Príspevok k priemyselným komunikačným štandardom, [Ph.D. thesis]*, FIIT STUBA, Bratislava, Slovakia, 2010.

[10] J. Dudak, G. Gaspar, and S. Sedivy, "Securing communication layer of uBUS protocol," in *Advanced Mechatronics Solutions*, R. Jabłoński and T. Brezina, Eds., vol. 393 of Advances in Intelligent Systems and Computing, Springer, Cham, 2016.

[11] M. D. R. Perera, R. G. N. Meegama, and M. K. Jayananda, "FPGA based single chip solution with 1-wire protocol for the design of smart sensor nodes," *Journal of Sensors*, vol. 2014, Article ID 125874, 11 pages, 2014.

[12] M. Eider, S. Kunze, and R. Poeschl, "FPGA based emulation of multiple 1-wire sensors for hardware in the loop tests," in *2016 IEEE Sensors Applications Symposium (SAS)*, pp. 1–6, Catania, Italy, 2016.

[13] A. de la Piedra, A. Braeken, and A. Touhafi, "Sensor systems based on FPGAs and their applications: a survey," *Sensors*, vol. 12, no. 9, pp. 12235–12264, 2012.

[14] M. Pizzotti, L. Perilli, M. del Prete et al., "A long-distance RF-powered sensor node with adaptive power management for IoT applications," *Sensors*, vol. 17, no. 8, p. 1732, 2017.

[15] F. Reverter, "The art of directly interfacing sensors to microcontrollers," *Journal of Low Power Electronics and Applications*, vol. 2, no. 4, pp. 265–281, 2012.

[16] J.-F. Tu, "Utilizing uart method for realizing the tii of ieee 1451," *International Journal of Electronics Engineering*, vol. 1, no. 2, pp. 133–139, 2009.

[17] "IEEE standard for a smart transducer interface for sensors and actuators - transducer to microprocessor communication protocols and transducer electronic data sheet (TEDS) formats," in *IEEE Std 1451.2-1997*, 1998.

[18] L. Cuifen, J. Ma, F. Hua, L. Ce, S. Lei, and J. Yu, "Wireless monitoring system for granary based on 1-wire," in *2010 International Conference On Computer Design and Applications*, pp. V4-496–V4-499, Qinhuangdao, China, 2010.

[19] L. Zhang, M. Yuan, D. Tai et al., "Design and implementation of granary monitoring system based on wireless sensor network node," in *2010 International Conference on Measuring Technology and Mechatronics Automation*, pp. 950–953, Changsha City, China, 2010.

[20] B. Peiffer and A. Kruger, "Physical layer architecture for 1-wire sensor communication bus: binary channel code division multiple access," in *2011 in 2011 IEEE Sensors Applications Symposium*, pp. 100–105, San Antonio, TX, USA, 2011.

[21] L. A. Magre Colorado and J. C. Martíinez-Santos, "Leveraging 1-wire communication bus system for secure home automation," in *Advances in Computing. CCC 2017*, A. Solano and H. Ordoñez, Eds., vol. 735 of Communications in Computer and Information Science, Springer, Cham, 2017.

[22] "APPLICATION NOTE 187: 1-wire search algorithm. Maxim Integrated," [cited 2016-07-08], 2002, https://www.maximintegrated.com/en/app-notes/index.mvp/id/187.

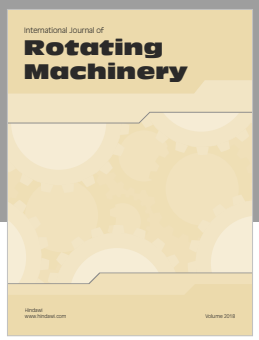
[23] R. D. LEE, "One-wire bus architecture," US Patent 5210846A, 1993.

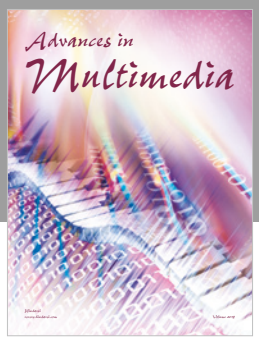
[24] J. Dudak, I. Sladek, and G. Gaspar, "Proposal and implementation of universal and non-volatile 1-wire module," in *2016 17th International Conference on Mechatronics - Mechatronika (ME)*, pp. 1–5, Prague, Czech Republic, 2016.

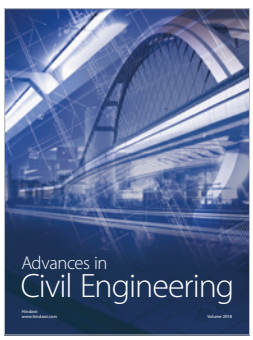
[25] "IEEE standard for a smart transducer interface for sensors and actuators - common functions, communication protocols, and transducer electronic data sheet (TEDS) formats," in *IEEE Std 1451.0-2007*, pp. 1–335, New York, NY, USA, 2007.

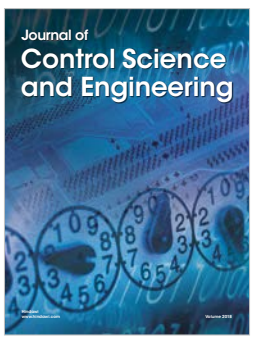
[26] *Value-line ARM®-based 32-bit MCU with up to 256-KB Flash, timers, ADC, communication interfaces, 2.4-3.6 V operation*, STMicroelectronics, Geneva, Switzerland, 2015, Datasheet, [cited 2017-05-24], http://www.st.com/resource/en/datasheet/stm32f030f4.pdf.

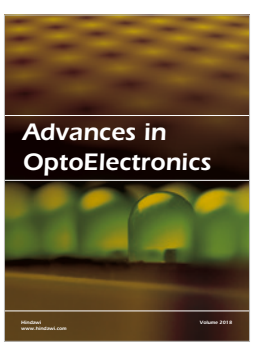
[27] *Access line ultra-low-power 32-bit MCU ARM ®-based Cortex ® -M0+, up to 32KB Flash, 8KB SRAM, 1KB EEPROM, ADC*, STMicroelectronics, Geneva, Switzerland, 2015, Datasheet, [cited 2017-05-24], http://www.st.com/resource/en/datasheet/stm32l031f4.pdf.

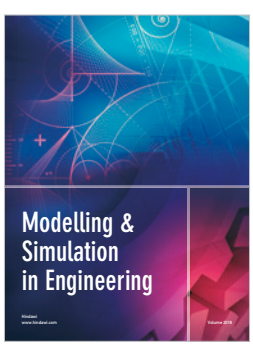


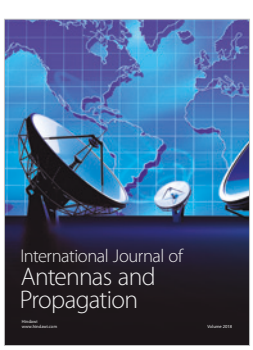


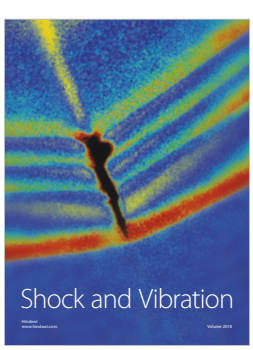


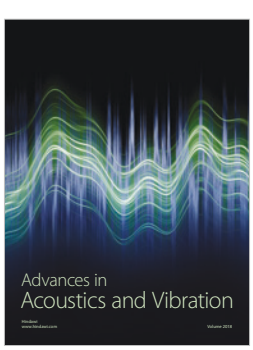












Hindawi

Submit your manuscripts at
www.hindawi.com