Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Model Driven Developed Terminal for Remote Control of Renewable Energy Sources Powered Station for Electric Vehicles Charging

Jovan N. Vujasinović, Goran S. Savić, and Milan D. Prokin, Member, IEEE

University of Belgrade - School of Electrical Engineering, Belgrade, Serbia

Corresponding author: Jovan N. Vujasinović (e-mail: jovan.vujasinovic@vfholding.rs).

This work is partially supported by National Association of Autonomous and Electric Vehicles, as European Institute of Innovation and Technology, Urban Mobility, Regional Innovation Scheme Serbia Hub.

ABSTRACT Terminal for remote control of renewable energy sources powered station for electric vehicles charging has been presented in this paper. This terminal enables remote control of electric vehicle chargers, smart storage batteries, smart electricity meters, cash registers, as well as, remote control of renewable energy sources and other devices within the station. This terminal also makes stations for electric vehicles charging powered by renewable energy sources more accessible to electric vehicles users, to electricity distribution system operators, to electricity supplier operators, to tax administration operators, and finally to users and owners of station. Therefore, communication and control with all these devices and systems is integrated in one device. Realization of hardware and software of such terminal has been also described in this paper. Its development and commercialization would encourage an increase in the use of electric vehicles powered by energy from renewable sources, which would cause the decreasing of the level of air pollution and all negative effects it brings. Different categories of this device are considered. Also, although it is a device with embedded software, a very advanced method was used, that is model driven development method, which enables fast and more efficient development and maintenance of the device.

INDEX TERMS Electric vehicles chargers, model driven development, remote control, renewable energy sources

1. INTRODUCTION

Production and using of electric vehicles is constantly being increased. The benefits of that trend in terms of air pollution reduction can be fully achieved only if electric vehicles are powered by energy obtained from renewable sources. Further development of the infrastructure for electric vehicles charging becomes more and more important, especially in terms of increasing the number of stations for electric vehicles charging powered by renewable energy sources [1].

Stations for electric vehicles charging powered by renewable energy sources are being integrated into one larger system which increases their accessibility to electric vehicles users, electricity distribution operators, suppliers and tax administration, as well as, to the owners and users of the electric vehicles charging stations. Such integration increases the efficiency of using the electricity distribution network and saves time and money. One of the most important features of such system is remote control of station for electric vehicles charging. The key device in this system is terminal for remote control of stations for electric vehicles charging powered by renewable energy sources. The realization of this terminal is presented in this paper.

This paper has a following structure. In Section 2 the architecture of the system for remote control of renewable energy sources powered station for electric vehicles charging has been described. Section 3 contains description of subsystems of the terminal for remote control of the charging station for electric vehicles. Hardware of the terminal for remote control of the charging station for electric vehicles has been described in Section 4. Different variants of terminal hardware implementation are presented in Section 5. Section 6 contains description of the software of the terminal for remote control of electric vehicles charging station.



1. Block diagram of the architecture of the system for remote control of renewable energy sources powered station for electric vehicles charging.

In Section 7, drivers for the terminal have been described, while the subroutines for executing the processes are disclosed in Section 8. The related work is presented in Section 9, and brief conclusion has been given in Section 10.

1. SYSTEM ARCHITECTURE

In Fig. 1 block diagram of the architecture of the system for remote control of renewable energy sources powered station for electric vehicles charging [2] is shown. Terminal for remote control of renewable energy sources powered station for electric vehicles charging is the main component of the system. By Internet of things (IoT) network infrastructure it is connected with renewable energy sources, smart storage battery, electric vehicles chargers, cash registers, smart electricity meter, user devices and other devices. This terminal is also connected to the cloud via the internet. That fact enables many functionalities, like monitoring, processing, setting and storage of data received from renewable energy sources, smart battery for energy storage, electric vehicles chargers, smart electricity meter and cash registers. The access to the mentioned data in the cloud is possible through several different platforms: platform for users of electric vehicle chargers, electricity trading platform, information system of the electricity distribution system operators and tax administration information system. The owners of electric vehicles receive all information about electric vehicle chargers through the platform for users of electric vehicle chargers, while the electricity available in the system is traded through the electricity trading platform. The owners of electric vehicles charging stations also have the access to the data stored in the cloud. In order to process this data, advanced algorithms have been used. That increases the efficiency of distribution network use, as well as, allows the significant savings in the system and enables the implementation of innovative smart energy services.

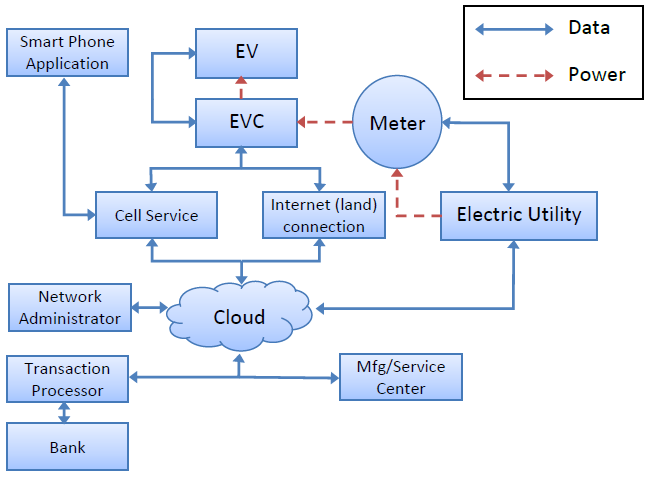
1. DESCRIPTION OF SUBSYSTEMS

System for remote control of renewable energy sources powered station for electric vehicles (EVs) charging contains four subsystems: the subsystem for remote control of electric vehicle chargers, the subsystem for remote control of smart batteries, the subsystem for remote control of smart meters and optional subsystem for remote control of fiscal cash registers.

1. THE SUBSYSTEM FOR REMOTE CONTROL OF EVS CHARGERS

The subsystem for remote control of electric vehicle chargers [3] has three main components: chargers for electric vehicles, a terminal for remote control of chargers for electric vehicles and a platform for users of electric vehicle chargers. This subsystem is shown in Fig. 2. Electric vehicles are supplied by energy from smart batteries. Smart batteries, on the other side, obtain the energy from renewable sources, or from the electricity distribution network, when necessary.

The terminal for remote control of electric vehicle chargers has access to data from electric vehicle chargers about available power, charging programs and prices. These data are being sent via internet to the platform for electric vehicle charger users, and thus become available to end consumers. This subsystem uses software algorithms in order to manage the price of electricity at electric vehicle charging stations depending on their location and the amount of available energy. This approach makes possible decreasing the price of electricity at stations with more available power (so that customers can be motivated to supply their electrical vehicles with electricity at these stations), while the price of electricity may be increased at stations with less available power (so that customers would be discouraged from supplying their electrical vehicles with electricity at these stations). Decisions about price corrections are made by the owner of the charger station in case this subsystem is a separate subsystem, or by artificial intelligence technologies in a unified terminal that has access to all devices (electric vehicle chargers, smart battery, smart meter and cash registers), appropriate information systems and software platforms.

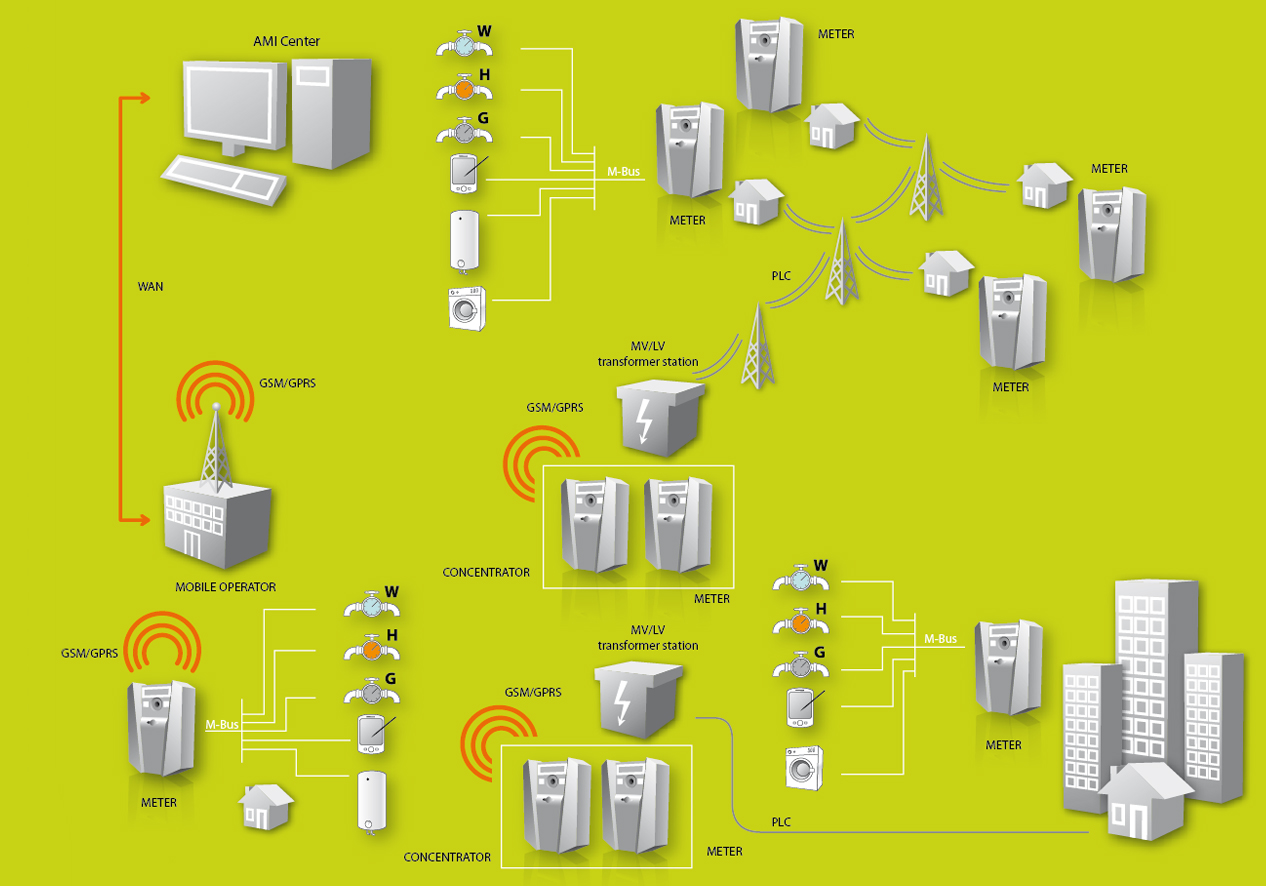


1. The subsystem for remote control of electric vehicle chargers.
2. THE SUBSYSTEM FOR REMOTE CONTROL OF SMART BATTERIES

The subsystem for remote control of smart batteries [4] has three main components: smart batteries for energy storage, terminals for remote control of smart batteries and electricity trading platform. This subsystem is shown in Fig. 3. Smart batteries for electricity storage obtain energy from renewable sources, or from the electricity distribution network, when necessary. The terminal for remote control of smart batteries provides the information about available energy in the smart batteries to the electricity trading platform. This allows the owners of electric vehicles charging stations who have a shortage of available energy in particular moment to buy energy (obtained from renewable sources) from producers which have a surplus of available energy. Similarly, the owners of electric vehicles charging stations who have a surplus of available energy in particular moment, can sell the energy to consumers which have a shortage of available energy in that moment. This also allows the owners of electric vehicles charging stations to buy surplus energy (obtained from renewable sources) at reasonable prices (due to increased production of energy from renewable sources at particular time) and to store the purchased energy in smart batteries for energy storage, which can be later sold at higher prices in moments when there is a shortage of energy in the system (due to reduced production of energy from renewable sources at particular time). The terminal for remote control of smart batteries is also responsible for controlling the process of smart batteries charging, after the purchase of electricity obtained from renewable sources.



1. The subsystem for remote control of smart batteries.



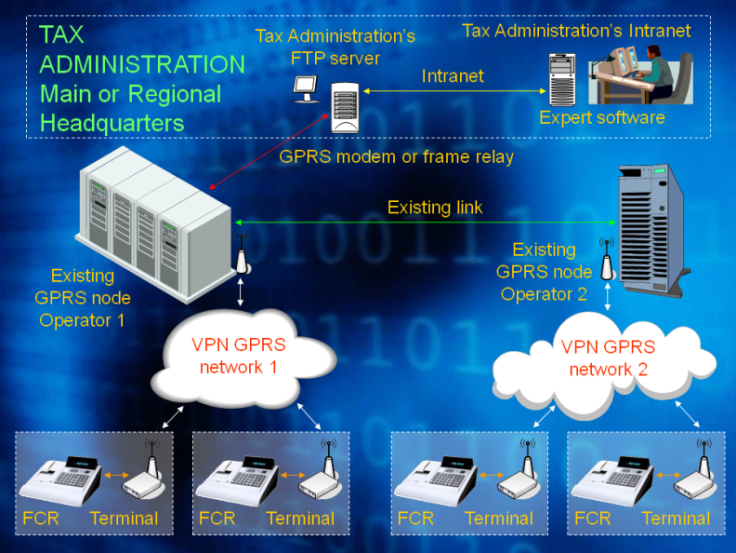
1. The subsystem for remote control of smart meters.
2. THE SUBSYSTEM FOR REMOTE CONTROL OF SMART METERS

The subsystem for remote control of smart meters [5] has three main components: smart meters, remote control terminal for smart meters and information system of the electricity distribution system operator.

This subsystem is shown in Fig. 4. Smart meters have plenty of functionalities. They measure active and reactive energy, register the average maximum power within a programmable period, measure the quality of electricity and display the appropriate data on the display. Also, smart meters support a flexible tariff policy and ensure the integrity of metering. Smart meters have the ability to record the profiles of the corresponding measured quantities, to record the event logs, to limit the consumed power and to remotely connect consumers to the power grid or to remotely disconnect the consumers from the power grid. The terminal for remote control of smart meters communicates with smart meters in order to collect the data from meters, to configure the meters, to set the meter parameters and to manage consumption. On the other side, the terminal for remote control of smart meters communicates with the information system of the electricity distribution system operator. This information system performs the functions of administration of components of the subsystem for remote control of smart meters, collects the data, performs smart meter settings and report creation functions.

1. THE SUBSYSTEM FOR REMOTE CONTROL OF FISCAL CASH REGISTERS

The subsystem for remote control of fiscal cash registers [6] has three main components: fiscal cash registers, terminals for remote management of fiscal cash registers and information system of the Tax Administration. This subsystem is shown in Fig. 5. Fiscal cash registers are able to receive commands from the cashier via the keyboard with visual monitoring of data on recorded transactions. They are also able to store the data in operational memory and fiscal memory and to print the data on the fiscal account. Fiscal cash registers have ability to group, to summarize and to present the data on the realized recorded turnover and the realized refunded turnover by tax rates, by articles and by cashiers. They can also download all relevant data in electronic form via appropriate input-output port. They satisfy the relevant security criteria as well. The terminal for remote control of fiscal cash registers allows remote reading and management of fiscal cash registers and the programming of the fiscal cash registers with data on article structures and data on article prices. It also downloads data from the fiscal cash registers and forms appropriate reports which are then transmitted to the information system of a Tax Administration. The information system of a Tax Administration receives reports on transactions at tax rates for a given period from the terminal for remote control of fiscal cash registers, data on resets and specifications of tax rates. The Tax Administration is responsible for monitoring of all transactions through its information system for ensuring of tax collection.



1. The subsystem for remote control of fiscal cash registers.
2. HARDWARE OF THE TERMINAL

Realizations of the first terminals for reading fiscal cash registers and fiscal printers are presented in [6-8]. Advanced fiscal cash registers with integrated terminal without and with additional services are presented in [9] and [10]. Terminals for wireless control of electric vehicle chargers are shown in [11].

Block diagram of the hardware of the terminal for remote control of renewable energy sources powered station for electric vehicles charging is shown in Fig. 1. The main part of the terminal is a microcontroller, which controls all operations. For reliable operation of the microcontroller, it is necessary to provide a reset circuit of the microcontroller in the terminal hardware, which enables the correct start of the microcontroller after the power up. Usually, with this type of device, a suitable connector is required, which can be accessed only after opening the device enclosure, and which is used for connection to a personal computer for in-circuit programming and debugging purposes. Also, if there is no special port on the microcontroller for this purpose, additional connectors are needed to which the serial port of the microcontroller will be connected either to the appropriate terminal port (in this case the cloud port) or to previously described connector for microcontroller programming by selecting the jumpers in the appropriate positions. The first position is used during regular operation of the terminal, while the second position is used for programming and debugging purposes during the production and servicing of the terminal. The voltage adapter contains the appropriate voltage regulators with accompanying capacitors and resistors. These voltage regulators adjust the input voltage to all required voltage levels, which are further used to power all components of the terminal. The voltage adapter also contains a corresponding rectifier of the AC voltage to DC voltage, in case the terminal is supplied by AC voltage. The real time clock provides accurate time information, regardless of power outages. For this reason, this integrated circuit should have a battery power backup. The parameters necessary for the proper operation of the microcontroller software are stored in non-volatile memory. These are parameters that must be permanently stored, and they ensure continuity in the operation of the terminal regardless of power outages. For these purposes EEPROM or FRAM memory can be used. Since exchange of data with the IoT infrastructure and exchange of data with the cloud are performed in different time slots, the terminal must store this data somewhere in the meantime. Due to the large amount of this data, it is expected that the internal operational memory of the microcontroller would not be sufficient for storage purposes. This is one of the reasons why the use of additional operational memory (RAM) in the terminal hardware is necessary. Another reason is that data contained in this memory should be saved regardless of possible external power outages. In the past, SRAM memory with implemented battery backup of the power supply (using the appropriate circuit with diodes or switches) was used. Nowadays, FRAM memory is used for these purposes.

The terminal has a minimum of two ports, one for communication with the cloud and the other for communication with the IoT network infrastructure [12]. Therefore, a microcontroller should contain at least two serial ports, or eventually, a microcontroller with one serial port can be used, while the other serial port can be realized with an additional integrated circuit. For a more advanced terminal, the microcontroller should have five serial ports. Two serial ports would be used for communication with the cloud: one for serial communication with the appropriate GSM/GPRS/3G/4G/5G modem and the other for the ETHERNET port. The realization of the ETHERNET port is possible directly if the microcontroller already has such port internally implemented, or indirectly by using the serial port of the microcontroller and an additional integrated circuit. The remaining three serial ports of the microcontroller would be used for communication with the IoT network infrastructure: one for wiring the devices within the station, another for wirelessly connecting the devices within the station and the third for wirelessly connecting the users of the station (port for local access, primarily for electric vehicle owners, who use the station to charge their vehicles, but it can also be used by a repairman and station owners/operators). The wired connection of the devices within the station should be realized via RS485 port, either directly if the microcontroller already has such an internally implemented port, or indirectly with an additional integrated circuit otherwise. For wirelessly connecting the devices within the station Zigbee or LoRa can be used, while for wirelessly connecting the users of the station WiFi would be the best choice. For the realization of each of these interfaces, it is necessary to provide an appropriate additional integrated circuit connected to the serial port of the microcontroller.

Depending on the functional requirements, it is usually necessary to provide a part for visual indication and a part with keys on the terminal. In general, these two parts allow the operator or a repairman to communicate directly with the terminal. Typically, the part with the keys can be implemented with several special keys or by using the appropriate keyboard. Through the part with the keys, the terminal or a microcontroller receives input information, such as commands or data from the operator or a repairman. The part for visual indication can be implemented by using light emitting diodes with accompanying resistors and inverters (in the basic variant), or by using display (in the more advanced variant). The light emitting diodes (LEDs) can be controlled by a microcontroller or by some peripherals, such as GSM/GPRS/3G/4G/5G modem (when LEDs indicate the presence of GSM/GPRS/3G/4G/5G network), while the display is controlled directly by the microcontroller. Using the part for visual indication, the microcontroller shows the operator or a repairman all the necessary information.

1. VARIANTS OF TERMINAL HARDWARE IMPLEMENTATION

Possible variants of terminal hardware implementation depend on the complexity of functional requirements. There is an obvious need for a device (Terminal) that would combine these aspects for described electric vehicles charging stations, as well as for households. However, households have different needs and capacities compared to commercial or industrial electric vehicles charging stations, so we propose the following three classes of terminals: light, standard and extended. The light class is intended for household use – residential charging stations. The light terminal should allow easy configuration, control and monitoring of appropriate devices in the home. First of all, it is expected that one or two electric vehicles chargers will be connected to the light terminal, as well as, the smart meters, smart electrical devices (smart water heaters, air conditioners, etc.) whose management can achieve economical management of electricity consumption in order to reduce costs, as well as the inclusion of users in the concept of a smart city (aspect of energy efficiency in the concept of a smart city) through communication with the electrical energy provider. In addition, the light terminal should enable the connection of a renewable energy source, such as solar panels on the roof of the house. While the standard class is intended for the needs of the most standard – commercial charging stations for electric vehicles, the extended class is intended for the needs of industrial charging stations for electric vehicles of enormous capacity. It is fundamentally identical to the standard class, with the difference that it supports much larger number of chargers and processes significantly larger amounts of data. It is utilized in large parking lots where a large number of parking spaces are equipped with chargers. In the future, such parking lots will be located in large shopping malls, vehicle manufacturers, etc.

Having in mind the optimal price-performance ratio, it is clear that the smallest budget is available for the realization of the light terminal, while the largest budget is available for the realization of the extended terminal. Therefore, the light terminal could be implemented with a corresponding 8051 microcontroller with two serial ports (or possibly one serial port and an additional integrated circuit for implementation of another serial port), EEPROM memory and SRAM memory with battery backup, a couple of buttons and a couple of LEDs. An ARM microcontroller with 5 serial ports, FRAM memory, a corresponding keyboard and a corresponding display can be used for implementation of the standard terminal. The extended terminal can be implemented using an industrial single board computer with 5 serial ports, large memory and ports for connecting a keyboard and a screen.

1. SOFTWARE OF THE TERMINAL

For the implementation of software that is executed in the microcontroller of the terminal, the structured programming is usually used. For such implementation of the software, it is necessary to define the algorithm of the main program, and then to develop the algorithms of all subroutines, as well as the description of the memory organization for data storage. The details of these algorithms depend on the technical specifications of the terminal.

At the beginning of the main program, shown in Fig. 6, a group of subroutines is executed in order to start the terminal. After that, the program enters into the infinite loop, where it checks the corresponding flags, and whenever these flags are activated (set), it calls the execution of the corresponding subroutines. These subroutines can be divided into 3 groups: subroutines for time-dependent tasks, subroutines for processing of the received messages, and subroutines for executing the processes. While performing the infinite loop, the strobe signal is also sent to the built-in watchdog timer of the microcontroller. This ensures that the watchdog timer resets the microcontroller in case the microcontroller software is blocked. If necessary, the execution of the main program is interrupted, in order to execute the appropriate interrupt service routines. There are at least serial interrupt service routines and timer interrupt service routines.

The group of subroutines for starting the terminal includes: *Configuration of the microcontroller*, *Initialization of global variables*, *Configuration of other components*, and *Recovering from a power outage*. The subroutine *Configuration of the microcontroller* configures all important microcontroller registers that determine: timer operation permission, timer operation mode, timer trigger frequency, UART operation permission, UART operation mode, interrupt permission, interrupt priorities, etc. The subroutine *Initialization of global variables* sets the initial values of all global variables. This is necessary since these variables, which are stored in the RAM of the microcontroller, have some arbitrary values after the microcontroller is powered up. The subroutine *Configuration of other components* configures all other terminal components except the microcontroller. This includes setting the display and/or LEDs, configuring ports, reading the real time clock and initializing the GSM/GPRS/3G/4G/5G modem by activating the flag *activatedModemInitialization* and by initialization of variables important for modem operation (appropriate setting of state variable and status variable, initialization of other control variables, initialization of first state control variables) and by initiating the activity of the first state of the process. The subroutine *Recovering from a power outage* ensures continuity in the operation of the terminal after power is recovered and reading the global variables from a non-volatile memory.



1. Main program.

The group of subroutines for time-dependent tasks includes at least the subroutine *Execution of time-dependent tasks*. This subroutine is called in the main program after activating the flag *oneMinuteElapsed*. Within that subroutine, the time is read and the execution of the appropriate tasks is started at the appropriate moments. The flag *oneMinuteElapsed* is activated in the interrupt service routine of the corresponding timer each time after the appropriate time interval has elapsed.

The subroutines for executing the processes represent a set of activities which are undertaken in response to an event. During execution, the process goes through different states. Examples of more complex processes are the process of making decisions in demand side management [13] and the process of optimal control of electric vehicle chargers in order to reduce the load on the network [14], using different strategies [15-19] depending on the tariff.

A special set of issues that must be addressed is the security of IoT devices [20-22] and terminals [23, 24], based on a large number of strategic and tactical recommendations and best practices in successful and secure implementations [25, 26].

1. DRIVERS

For the proper functioning of the software described in the previous section, the light terminal also requires the implementation of the necessary drivers. The implementation of the standard terminal has two possible variants: the utilization of mentioned drivers, and the utilization of a specialized operating system (FreeRTOS). The extended terminal is run using the Linux or Windows operating system.

The key drivers are the following interrupt service routines: the serial interrupt service routine and the timer interrupt service routine, monitoring the expiration of the one-minute period, controlling the keys and processing the appropriate number of timers required for the operation of the entire microcontroller software. Serial communication (with IoT network infrastructure or an owner or a cloud) is performed in this subroutine, if additional serial ports are implemented with external integrated circuits. This subroutine is called an appropriate number of times per second, which is ensured by writing the appropriate time interval between the two interrupts in the timer registers during each execution of this subroutine. Each time a period of one minute expires, the timer interrupt service routine activates the *oneMinuteElapsed* flag.

A timer is used to detect the expiration of predefined period of time. Processing the timer in the timer interrupt service routine includes decrementing the timer value and performing the appropriate action when the timer value equals zero. This timer operation algorithm is shown in Fig. 7. Timers have a very important role in preventing the blockade of the execution of microcontroller program due to waiting for a corresponding event from the external environment. The microcontroller can have several independent timer interrupts, with different interrupt periods.

The serial interrupt service routine is responsible for receiving and sending the messages in serial communication, via microcontroller's UART. The serial interrupt service routine is called when one byte of the message has been received, or when one byte of the message has been sent and the next byte of the message has to be sent. Also, the serial interrupt service routine detects the end of the received message. As previously described, the microcontroller's UART is used for a serial connection to a GSM/GPRS modem, and through that serial connection the microcontroller realizes three different communications: communication with the modem itself, communication with the cloud and communication with the service technician. Therefore, the serial interrupt service routine is responsible for receiving and sending the messages in these communications. The microcontroller's UART can be used for a serial connection to a connector (described in Section III), and through that serial connection the microcontroller can communicate with a service technician. The microcontroller communicates with the modem using the AT protocol, i.e. using the AT commands. Communication between the microcontroller and the cloud is realized via modem within the established GSM/GPRS/3G/4G/5G data connection, using the protocol for communication with the cloud. The microcontroller communicates with the repairman either directly by a cable connection via the connector, or indirectly by wireless connection via a modem within the established GSM/GPRS/3G/4G/5G data connection.



1. Timer interrupt service routine.



1. Serial interrupt service routine.

Therefore, due to different packet structures used in serial communication, it is necessary to have a variable for storage of information about the current serial communication mode, i.e. the variable *serialCommunicationMode*. This variable has standard value of the AT\_COMMAND constant, which means that the structure of the current communication packet is consistent with the packet structure of the AT protocol. The only deviation from its standard value, this variable has during communication with the cloud from the moment of establishment to the moment of termination of GSM/GPRS/3G/4G/5G data connection, and then its value is equal to the CLOUD\_DATA constant. The algorithm of the serial interrupt service routine is shown in Fig. 8. The serial interrupt service routine checks the variable *serialCommunicationMode*, and depending on its value calls one of the following subroutines: *AT communication* or *Cloud communication*.

1. THE SUBROUTINES FOR EXECUTING THE PROCESSES

In order to more clearly explain the subprograms from this group, it is worth first explaining the process execution mechanism itself. A process is a closed set of activities undertaken in response to an event to generate an output. During execution, a process goes through various states. Process state changes are conditioned by corresponding events. In each state, appropriate action is taken.

The method of realization of the process is as follows. Each process is assigned: activation flag, state variable, status variable, startup routine, execution routine, state routine, process control variables, and state control variables.

The process can be found in different states. Information about the state of the process is stored in the corresponding state variable.

Corresponding events cause the state of the process to change. Information about these events is stored in appropriate flags or variables.

The process is executed in a number of cycles. In one cycle of the process, the activity of the state in which the process is at that moment is executed once.

In each state of the process, a corresponding activity is executed. The number of different activities is equal to the number of different process states, so one activity corresponds to each process state.

Repetition of the same activity occurs in the case of keeping the process in the corresponding state for more than one cycle.

The process is executed within the appropriate execution subroutine, which is executed in the main program each time it passes through the infinite loop as long as the activation flag is active.

One cycle of the process corresponds to one pass through the infinite loop of the main program.

Depending on the value of the state variable, the execution routine calls the corresponding state routine.

Within the status subprogram, the appropriate activity is executed, checking whether the appropriate events have occurred and, depending on that, the process remaining in the same state or the appropriate change of the process state.

Starting the process, that is changing the state of the process from the state of inactivity to the first state of execution, is done by calling and executing the start subroutine. As part of the startup subroutine, the corresponding activation flag is activated, the state variable and status variable are set appropriately, other process control variables are initialized, the control variables of the first state are initialized, and the activity of the first state of the process is started.

Changing the state of the process from the current state to the next state is done by setting the state variable accordingly, initializing the control variables of the next state and starting the next state activity.

Stopping the process, that is changing the state of the process from the current state to the state of inactivity is done by deactivating the activation flag, setting the state variable and the status variable accordingly and, if necessary, setting the appropriate parameters that affect the next start of the process, the start of other subprograms or processes, etc. If necessary, stopping the process can be done within a separate subroutine, i.e., within the stop subroutine.

Putting the process into the waiting state, i.e. changing the process state from the current state of execution to the state of waiting is done by deactivating the activation flag, setting the state variable accordingly and, if necessary, starting the detection of appropriate events that will return the process from the state of waiting to the state of execution.

The status of the process, that is the information whether the process has successfully generated a result, is stored in the corresponding status variable. In the same variable, information about the reason for the failure of the process is stored, in case of unsuccessful completion of the process.

This description of the process is a model driven development, i.e. a model for object-oriented programming. The advantage of this way of programming is that it enables much faster and better quality development and maintenance of software. On the other hand, the disadvantage is that the programmer has less control over the speed of execution and the amount of program and working memory required for the operation of the resulting software. This can be limiting for embedded software applications. However, in the following text, we will describe how the process model thus obtained is translated into program code, in such a way as to achieve all the advantages and neutralize the mentioned disadvantages on the example of complex subprograms.

In the further text of this chapter, a detailed explanation of one of the more complex subprograms from this group follows, that is, the subprogram of creating a complete daily (Z) report. The explanation is given for the fiscal cash register CR401 manufactured by Intracom, and the same can be applied to other fiscal cash registers. The purpose of the CreateCompleteZReport subprogram is to execute the process of creating a complete Z report. Fig. 9 shows a state diagram of the complete Z report workflow.

The result of the process, that is the purpose of the process of creating a complete Z report, is to prepare a daily turnover report by items, a daily turnover report by operators (cashiers) and a daily report for sending to the PO server, that is to ensure the reading of data from the fiscal cash register, the formatting of such read data into reports and the placement of such generated reports into the terminal's memory. The PO server is the taxpayer's server, which allows a taxpayer to monitor all relevant data and possibly adjust the parameters available to that taxpayer.

The process can be found in five states. The names of the states are determined in accordance with the procedure that is carried out in that state in the fiscal cash register. The following is an overview of the status and corresponding activities, events and status changes:

State of inactivity. The process is initially in this state. From this state, the process can move to the state of reading the PLU (Price Look-Up) structure of the article, if a key activation of less than 5 seconds occurs or a RequestZReport command is received from the PO server.

Reading of PLU article structures state. The purpose of this state is to prepare a daily turnover report by item to be sent to the PO server. Therefore, the activity of the process in this state is reading the PLU structure of items from the fiscal register, formatting the thus obtained data and placing it in the memory of the terminal in the data block seriesKZReports in the PLUReport field. In the regular execution of the process, this activity is executed 10,000 times, because the fiscal cash register CR401 manufactured by Intracom has the ability to work with 10,000 items. The process remains in the same state as long as the PLU structure of the item with sequence number from the set of numbers 0 to 9998 happens to be read. When the PLU structure of the item with sequence number 9999 happens to be read, the process goes into the state of reading of operator statistics. The process remains in the same state until it happens that some PLU structure is not read in the first or second attempt. If it happens that some PLU structure is not read in the third attempt, the process goes into the inactive state and places the information about the unsuccessful execution of the process because the cash register does not respond to the command to read the PLU structure in the status variable of statusZReport.

Reading of operator statistics state. The purpose of this state is to prepare a daily turnover report by cashiers. Therefore, the activity process in this state is the reading of the operator's statistical data (hereinafter referred to as the operator structure) from the fiscal cash register, formatting the thus obtained data and placing it in the terminal's memory in the data block seriesKZReports in the operatorReport field. In the regular execution of the process, this activity is executed 15 times, because the cash register CR401 manufactured by Intracom has the ability to work with 15 cashiers. The process remains in the same state as long as the operator structure with the sequence number from the set of numbers 1 to 14 happens to be read. When the operator structure with the sequence number 15 happens to be read, the process goes into the execution of Z report state. The process remains in the same state until it happens that some operator structure is not read in the first or second attempt. If it happens that some operator structure is not read in the third attempt, the process goes into an inactive state and places information about the failed execution of the process because the cash register does not respond to the command to read operator statistics in the status variable statusZReport.



1. A state diagram of the complete Z report workflow.

Execution of Z report state. The purpose of this state is the creation of a daily report, that is, the execution of the Z report in the fiscal cash register in the stipulated time period. Therefore, the activity of the process in this state is the execution of the Z report in the fiscal cash register, the detection of the event that the Z report execution procedure in the fiscal cash register has been completed, and the initiation of the event detection procedure that the scheduled time period for the execution of the Z report has expired. Detection of the event that the Z report execution procedure in the fiscal cash register has been completed is done by asking whether the cash register responds to the time reading command (time reading was chosen as one of the simpler commands). Such detection is necessary for the following reasons: the cash register does not respond to the command to execute the Z report, the cash register does not respond to any command during the execution of the Z report, and the time required by the cash register to execute the Z report is not constant but depends on the number of items sold in that day and other factors. Initiation of the event detection procedure that the scheduled time period for execution of the Z report has expired is done by activating the appropriate timer. In regular process execution, this activity is executed once. The time reading is performed several times, which depends on the ratio of the time required by the cash register to execute the Z report and the time waiting for the response from the cash register. If the event occurs that the time has not been read, the process remains in the same state. If the event occurs that the time has been read (the event that the Z report execution procedure in the fiscal cash register is completed) or that the scheduled time period for the Z report execution has expired, the process switches to the reading of Z report state.

Reading of Z report state. The purpose of this state is to prepare a daily report for sending to the PO server. Therefore, the activity of the process in this state is the reading of the last Z data (hereinafter the last Z report) from the fiscal cash register, formatting the thus obtained data, placing it in the memory of the terminal in the data block seriesKZReports in the field descriptionZReport and detecting the event that the Z report execution procedure is successfully completed in the fiscal cash register. This detection is performed by checking whether the number of the last Z report read from the cash register is different from the number of the last Z report stored in the terminal. In regular process execution, this activity is executed once. If events occur that the last Z report was read and that the Z report was successfully executed, the process goes into the inactive state and places the information about the successful execution of the process in the status variable status. The process remains in the same state until it happens that the last Z report is not read in the first or second attempt. If it happens that the last Z report is not read in the third attempt, the process goes into the inactive state and places the information about the unsuccessful execution of the process because the cash register does not respond to the command to read the last Z report in the status variable of statusZReport. If the events occur that the last Z report was read and the Z report failed to execute on the first or second attempt, the process enters the execution of Z report state. If it happens that the last Z report was read and the Z report was executed unsuccessfully in the third attempt, the process goes into the inactive state and places the information about the failed execution of the process because the cash register does not execute the Z report in the status variable of statusZReport.

The flag of the activation of this process is the workCompleteZReport flag. The execution subroutine of this process is the CreateCompleteZReport subroutine. The state variable of this process is the status variable, that is, the same variable that is used to store information about the busy status of communication with the fiscal cash register. The status variable of this process is the variable statusZReport within the field descriptionZReport of the pointed member of the block stringKZReport in the terminal memory. The process startup subroutine is the InitializationWorkCompleteZReport subroutine.

The process is started by the InitializationRunCompleteZReport within the timer 0 interrupt service routine or the ModemMessageProcessing subroutine or the PowerLossRecovery subroutine. The process is executed in the CreateCompleteZReport subroutine.

The CreateCompleteZReport subroutine is executed in the main program as long as the workCompleteZReport flag is active. All the time during the process of creating a complete Z report, information about the state of the process is stored in the status variable. For all time during and after the process of creating a complete Z report, the status information is stored in the variable statusZReport within the field descriptionZReport of the pointed member of the block stringKZReport in the terminal memory.

Fig. 10. shows the algorithm of the CreateCompleteZReport subprogram. The main purpose of the CreateCompleteZReport subprogram is to ensure the execution of the process of creating a complete Z report, that is, the execution of the corresponding activity, that is, the corresponding state subprogram, depending on the state of the process.

Therefore, at the beginning of the CreateCompleteZReport subprogram, the status of the process is checked, that is, the status variable is checked. Depending on the state of the process, the corresponding previously described activity is executed by calling and executing the corresponding state subroutine.

If the process is in the state of reading of PLU structures, the ReadingOfDailyPLUReport subroutine is called. If the process is in the state of reading of operator statistics, the ReadingOfOperatorStatistics subroutine is called. If the process is in execution of Z report state, the ExecutionOfZReport subroutine is called. If the process is in the state of reading of Z report, the subroutine ReadingOfZReport is called. If the process is not in any of the mentioned 5 states, the variable status is set to the value of the constant INACTIVE.



1. CreateCompletZReport subroutine algorithm.
2. RELATED WORK

Papers dealing with individual subsystems can be found. Thus, papers [27-28] show the analysis of terminal hardware and software for the system for remote control of meters, while papers [6-10] show the analysis of hardware and software for the system for remote control of fiscal cash registers, while papers [11, 14-19, 29] provide an analysis of a system for remote control of electric vehicle chargers. Paper [3] provides an overview of works dealing with systems and terminals for remote control of charging stations for electric vehicles powered by solar power plants. Reference [4] shows a system for remote control of smart batteries.

One can find works that deal with model design development, that is, object-oriented way of programming embedded software, although they are not common. And the application in practice is still not common due to less control of the programmer over the speed of execution and the amount of program and working memory required for the operation of the software obtained in this way, which is very important for embedded software. Paper [30] provides a model for object-oriented programming that is applied in the automotive industry. Paper [31] talks about the trend that embedded software development is shifting from manual programming to model-driven development (MDD), and why it is important to assure the quality of embedded software. Work [32] elaborates verifying protocol conformance using software model checking for the model-driven development of embedded systems.

Compared to the abbreviated version [33], this paper is much more detailed (i.e., contains a general definition of the model, i.e. the process), and gives an overall overview of the hardware and software, and in particular adds a detailed explanation of the procedures for executing the process using model design development, that is the object-oriented way programming. In comparison with other works, none of the previously mentioned approaches offers such system, i.e. a terminal with unified control of chargers, renewable sources, storage, meters and fiscal cash registers, as well as categorization of such a terminal. Additionally, in addition to the realization of the terminal hardware that enables communication with all of these devices, this paper also provides a design development model, that is a model for both structured and object-oriented implementation of embedded software, which can be used for application in smart cities systems, smart metering, smart home, smart grid and smart energy management in general. This design development model gave excellent results in practice when applied to terminals for remote control of fiscal cash registers, enabling at the same time much faster and better software development and maintenance, while maintaining a sufficient level of control over execution speed and the amount of required working and program memory.

1. CONCLUSION

The realization of the terminal for remote control of renewable energy sources powered station for electric vehicles charging has been considered in this paper. The architecture of the system for remote control of the station has been presented, followed by a description of a block diagram of the terminal hardware. It has been concluded that the realization of the terminal hardware can be based on a microcontroller, with appropriate memory, communication ports, keys, visual indicators, a real time clock and a power adapter. Three variants of hardware implementation and required drivers are proposed. The terminal software operation is based on the initialization of all necessary registers, variables, components and processes, and further execution of the infinite main loop of the program, within which all necessary processes are executed, occasionally interrupted by interrupt service routines. An overview of the complete terminal software is given. The implementation of very complex subprograms for process execution is also explained in detail. The model-driven development was used, which enables the application of object-oriented programming. This is a very advanced method for embedded software. In this way, much faster and better quality development and maintenance of software is possible while maintaining control over the speed of execution and consumption of program and working memory. The practical application of this model in practice during the implementation of terminals for remote control of fiscal cash registers gave satisfactory results. In further work, the development of a compiler that would automatically generate program code based on the model can be considered.

REFERENCES

1. P. Arunkumar, K. Vijith, “IOT enabled smart charging stations for electric vehicle,” *International Journal of Pure and Applied Mathematics*, vol. 119, no. 7, pp. 247-252, 2018.
2. J. Vujasinović, G. Savić, M. Prokin, “Terminal za daljinsko upravljanje stanice za punjenje električnih vozila koja se napaja iz obnovljivih izvora električne energije,” in *Proceedings of the 20. intenational symposium INFOTEH-JAHORINA*, Istočno Sarajevo, Bosna i Hercegovina, Mart 2021, pp. 1-5.
3. S. Khan, A. Ahmad, F. Ahmad, M.S. Shemami, M.S. Alam, S. Khateeb, “A comprehensive review on solar powered electric vehicle charging system,” *Smart Science*, vol. 6, no. 1, pp. 54-79, 2017.
4. https://sonnengroup.com/sonnenbatterie/
5. S. Marjanovic, J. Vujasinovic, G. Savic, S. Manojlovic, B. Ninkovic, P. Medjedovic, “System for electric power remote reading and load management,” in *Proceedings of the VI Advising the Bosnia and Herzegovina Committee CIGRE*, Neum, Bosnia and Herzegovina, November 2003.
6. J. Vujasinovic, “Terminal for Remote Reading and Management of Fiscal Cash Registers,” M. S. thesis, University of Belgrade - School of Electrical Engineering, Belgrade, Serbia, 2013.
7. M. Prokin, D. Prokin, “First generation of turnover control devices,” in *Proceedings of the 19th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 2011, pp. 888-891.
8. M. Prokin, D. Prokin, “GPRS terminals for reading fiscal registers,” in *Proceedings of the 2nd Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 15-20 June 2013, pp. 259-262.
9. M. Prokin, D. Prokin, “Improved fiscal devices without additional services,” in *Proceedings of the 5th Mediterranean Conference on Embedded Computing (MECO)*, Bar, Montenegro, 12-16 June 2016, pp. 273-276.
10. M. Prokin, D. Prokin, “Improved fiscal devices with additional services,” in *Proceedings of the 5th Mediterranean Conference on Embedded Computing (MECO)*, Bar, Montenegro, 12-16 June 2016, pp. 277-280.
11. M. Prokin, M. Čabarkapa, J. Stojković, D. Prokin, “Wireless control of chargers for electric vehicles,” in *Proceedings of the 8th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 10-14 June 2019, pp. 502-505.
12. J. Vujasinović, G. Savić, Z. Čiča, “Uvođenje IoT u stanicu za punjenje električnih vozila koja se napaja iz obnovljivih izvora energije,” in *Proceedings of the 28th Telecommunication Forum TELFOR*, Belgrade, Serbia, November 2020.
13. J. Vujasinović, G. Savić, A.Rakić, “Upravljanje potražnjom kod stanice za punjenje električnih vozila koja se napaja iz obnovljivih izvora energije,” in *Proceedings of the 28th Telecommunication Forum TELFOR*, Belgrade, Serbia, November 2020.
14. M. Prokin, J. Stojković, M. Čabarkapa, D. Prokin, “Optimal control of chargers for electric vehicles,” in *Proceedings of the 8th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 10-14 June 2019, pp. 557-560.
15. C. Zhang, Q. Huang, J. Tian, L. Chen, Y. Cao, R. Zhang, “Smart grid facing the new challenge: the management of electric vehile charging loads,” *Energy Procedia*, vol. 12, pp. 98-103, 2011.
16. G. Xu, L. Bai, “Heuristic methods for optimal electric vehicle charging scheduling in smart grid,” *Int. J. Automation and Logistics*, vol. 1, no. 1, pp. 22-46, 2013.
17. E. Crisostomi, R. Shorten, S. Stuedli, F. Wirth, “Electric and plug-in hybrid vehicle networks: Optimization and control,” *CRC Press*, Boca Raton, FL, USA, 2018.
18. W. Tang, Y. J. (A.) Zhang, “Optimal charging control of electric vehicles in smart grids,” *Springer Briefs in Electrical and Computer Engineering*, Springer, Cham, Switzerland, 2017.
19. S. Rajakaruna, F. Shahnia, A. Ghosh, “Plug in electric vehicles in smart grids – Charging strategies,” *Power Systems*, Springer, Singapore, 2015.
20. M. Čabarkapa, M. Prokin, G. Šimić, N. Nešković, Đ. Budimir, “Internet of insecure things,” in *Proceedings of “Archibald Reiss Days”*, Belgrade, Serbia, 7-9 Nov. 2017, vol. III, pp. 101-109.
21. A. Miljković, M. Čabarkapa, M. Prokin, Đ. Budimir, “The importance of IoT and IoT forensics,” in *Proceedings of “Archibald Reiss Days”*, Belgrade, Serbia, 2-3 Oct. 2018, vol. II, pp. 395-404.
22. “Internet of things poses opportunities for cyber crime, ” *Federal Bureau of Investigation*, Washington DC, USA, Sep. 10, 2015.
23. M. Prokin, D. Prokin, A. Nešković, N. Nešković, “Cybersecurity of fiscal devices with GPRS terminals,” in *Proceedings of the 7th Mediterranean Conference on Embedded Computing (MECO)*, 10-14 June 2018, Budva, Montenegro, pp. 171-174.
24. M. Prokin, D. Prokin, A. Nešković, N. Nešković, “Cybersecurity of improved fiscal devices,” in *Proceedings of the 7th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 10-14 June 2018, pp. 175-178.
25. J.A. Eisenach, C. Barfield, J.K. Glassman, M. Loyola, S. Tews, “An American strategy for cyberspace – Advancing freedom, security and prosperity,” *American Enterprise Institute*, June 2016.
26. Y.M.Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, C. Rossow, “IoTPOT: A novel honeypot for revealing current IoT threats,” *J. Information Processing*, vol. 24, no. 3, pp. 522-533, 2016.
27. V. Lapcevic, S. Manojlovic, J. Vujasinovic, S. Marjanovic; “Experimental results on the load management system and remote meter reading system,” in *Proceedings of the XI International Electrotechnical and Computer Science Conference ERK 2002*, Portoroz, Slovenia, September 2002.
28. B. S. England, A. T. Alouani, “Multiple Loads-Single Smart Meter for Measurement and Control of Smart Grid,” in *Proceedings of the 2019 IEEE Innovative Smart Grid Technologies* - Asia (ISGT Asia), 2019, pp. 2440-2444.
29. P. Arunkumar, K. Vijit, “IOT Enabled smart charging stations for Electric Vehicle,” *International Journal of Pure and Applied Mathematics*, vol. 119, no. 7, pp. 247-252, 2018.
30. S. Konrad, B. H.C. Cheng, L. A. Campbell, Member, “Object Analysis Patterns for Embedded Systems,” *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 970-992, 2004.
31. P., M. Trapp, “Trends in Embedded Software Engineering,” *IEEE Software*, vol. 26, no 3, pp. 19-25, 2009.
32. Y. Moffett, J. Dingel, A. Beaulieu, “Verifying Protocol Conformance Using Software Model Checking for the Model-Driven Development of Embedded Systems,” *IEEE Transactions on Software engineering*, vol. 39, no. 9, pp. 1307-1325, 2013.
33. J. Vujasinovic, G. Savic, M. Prokin, “Terminal for Remote Control of Renewable Energy Sources Powered Station for Electric Vehicles Charging,” in *Proceedings of the 10th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, June 2021.

JOVAN N. VUJASINOVIĆ was born in Serbia in 1977. He received the B.S.E.E., M.S.E.E. in 2001 and 2013, respectively, from School of Electrical Engineering, the University of Belgrade, Serbia. He joined the Department of Electronics at the same institution in 2001, as a Teaching and Research Assistant. Since 2007, he has been an entrepreneur in electronic and IT industry. He is currently also Ph.D. student. His research interests include the field of smart energy management, smart cities, smart metering systems, as well as remote management of other measuring and industrial devices.



**GORAN S. SAVIĆ** received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Belgrade, Belgrade, Serbia, in 2002, 2009, and 2017, respectively.

In 2005, he joined the School of Electrical Engineering, University of Belgrade, as a Teaching and Research Assistant. He is currently an Assistant Professor with the School of Electrical Engineering, University of Belgrade. His research interests include image and video compression, signal processing, microcontroller and FPGA applications.

**MILAN D. PROKIN** was born in Serbia in 1963. He received the B.S.E.E., M.S.E.E. and Ph.D. degree in 1986, 1988 and 1990, respectively, from School of Electrical Engineering, the University of Belgrade, Serbia. He joined the Department of Electronics at the same institution in 1986, where he is presently a full professor. His research areas are image and video compression, signal processing, switching amplifiers, real-time control, microcontroller applications and advanced I/O.