Seeeduinoboot: an Advanced Bootloader for Seeeduino Stalker v2 Platforms, Allowing Remote Firmware Update

Catalin Bujdei, Sorin-Aurel Moraru

Department of Computer Science and Electrical Engineering, Transilvania University of Brasov, Mihai Viteazul street, no. 5, V III 8, Brasov, Romania 500174, e-mails: catalin.bujdei@unitbv.ro, smoraru@unitbv.ro

Abstract: For Wireless Sensor Networks (WSNs), where the network devices could be deployed into a large space area, updating directly the firmware application becomes usually complicated or even impossible. A remote solution is absolutely necessary. It decreases the maintenance time and it reduces the cost of the implementation. We have already proposed an implementation of the network device using the Seeeduino Stalker v2.1 platform, as processing unit, and XBee Series 2 module, as wireless communication unit. To implement the remote firmware update functionality it was necessary to modify the existent bootloader application, by adding the possibility of processing the XBee communication telegrams and direct operations on device memory. The new bootloader application, called Seeeduinoboot, has been developed starting from an existing open source version, Optiboot 4.4.

Keywords: remote firmware update; bootloader application; wireless sensor network; Seeeduino Stalker platform; XBee Series 2 module

1 Introduction

The microcontrollers are developed for a longer period of time and new versions are designed and implemented continuously, providing faster computing speed, more functionalities, more possibilities to configure them, less energy consumption and smaller size. To use one or another type of a microcontroller depends directly on the application type and requirements. The most of electronic devices include a microcontroller component, as a processing unit, with the role of monitoring and controlling the device's functionalities. It is useless without proper software included.

Usually in the microcontroller's memory there are installed 2 different types of software applications:

- *bootloader application* which is executed immediately after the microcontroller is powered on or when a reset operation occurred. This application initializes the electronic device.
- *firmware application* which is executed after the bootloader application ends its execution. The firmware application will run continuously until the microcontroller is powered off or reset. Usually, into the structure of this application, it is integrated an infinite software cycle, where the events, occurred during the device functioning, are detected and treated accordingly.

In generally the update of the bootloader and firmware applications is done using specific programming devices (programmers), which support different types of communications protocols and are dedicated for a specific set of microcontrollers. As programming methods could be mentioned: JTAG (Joint Test Action Group), SPI (Serial Peripheral Interface), PDI (Program and Debug Interface) and ISP (In-System Programming). The main advantage of the ISP method is that the microcontroller could be programmed as it is installed into the system, and it is not required to be detached.

For the most of the electronic devices it is never necessary to modify any of these applications, but in case that the device is under testing, like in our system implementation, it is important to have this ability. The update of the firmware application is required mostly when there are software bugs to be corrected, new functionalities have to be added or it is just necessary to modify the existent functionalities [1].

When we have a programmer device we just have to connect it directly to the microcontroller, according to the specifications, and execute the memory update procedure with the help of dedicated software tools. This is the easiest way of updating a firmware application, but it supposes to have the programmer device available and easy access to connect it to the microcontroller. In the situation when we don't have direct access to the microcontroller a remote firmware update procedure is required.

The paper is organized in 5 sections. Following the introduction, Section II presents some essential details related to the already constructed network device and enumerates a few existent projects of implementing the remote firmware update procedure. Section III presents information about In-Application Programming procedure and the Intel HEX file format. Section IV presents details about the implemented procedure and Section V the results obtained. At the end are presented the conclusion of the work.

2 Related Work

The research project, under development by our research team, supposes to create a system for monitoring of the indoor ambient conditions. The necessary data is recorded by a Wireless Sensor Network (WSN) and it is used to improve the occupancies comfort and decrease the energy consumption.

It is known that the main constraints of the WSNs are related to: energy consumption, cost, modularity, general character, data processing and security. Considering these aspects it has been established already to construct the network devices using the Seeeduino Stalker v2.1 platform, as processing unit, and XBee Series 2 module, as wireless communication unit (Fig. 1). This solution is simple, low cost, modular and easily adaptable for different types of applications - general character. Different types of extension shields, including sensors and/or actuators, have been constructed following predefined rules. The extension shields could be attached easily to the main board through the existent extension pins, using a simple plug & play operation [2, 3].

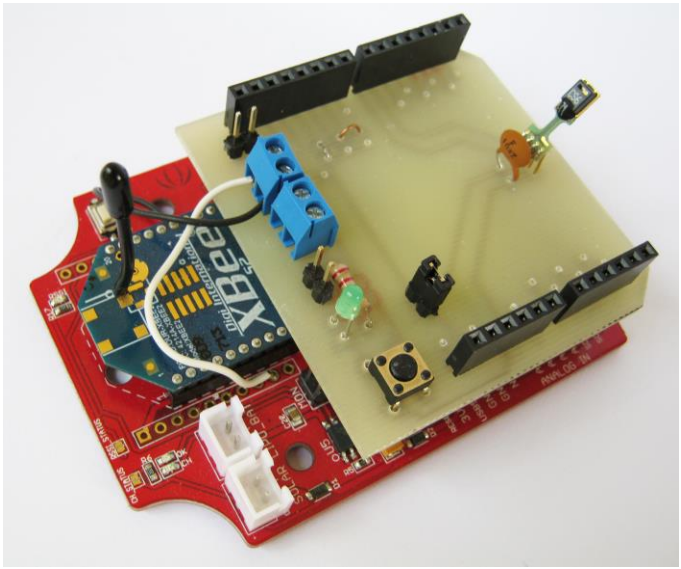

Figure 1
Network device implementation

In Fig. 2 are presented the main interconnections between the device components, most important for the remote firmware update functionality. The 3rd digital output (DO3) of the XBee module commands the reset operation on the microcontroller. The serial communication (RX and TX pins), linked to the XBee module, will allow the microcontroller to exchange data through the wireless environment.

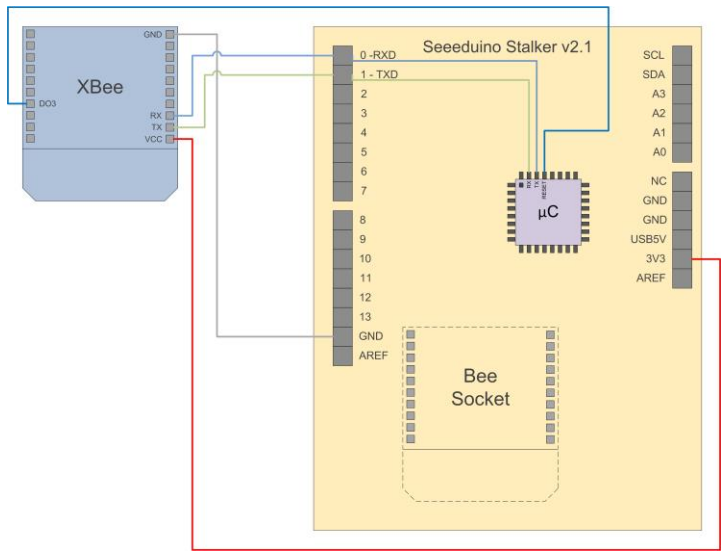

Figure 2
The main interconnections between network device components

The microcontroller integrated on Seeeduino Stalker v2.1 board is ATmega 328P. It includes 32K Bytes of In-System Self-Programmable Flash program memory, 1K Byte of EEPROM memory and 2K Bytes of SRAM memory. The Flash memory is organized as 16K x 16 (1 word) and for security reasons it is divided in 2 sections: Boot Loader Section (dedicated for bootloader application) and Application Program Section (dedicated for firmware application), as presented in Fig. 3 [11].

Figure 3
ATmega 328p Flash memory organization

The XBee Series 2 modules support 2 modes of communication: an AT (transparent) mode and an API mode. In transparent mode the data is sent further as it is received (the same content but different format). It is the simplest mode of communication but it has lower capabilities. The API mode is more complex and properly for the most of applications, all the wireless communication between devices is done using predefined telegram formats. The general format of a telegram in API mode is presented in Fig. 4.

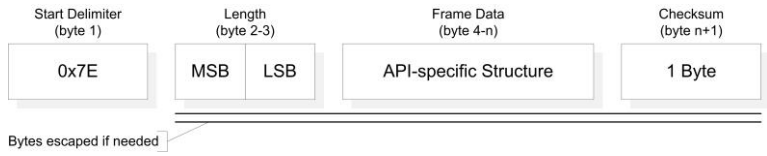

Figure 4
XBee API telegram format

XBee Series 2 modules support the ZigBee protocol of communication, the wireless communication between devices being accomplished using frame data in ZigBee format. There are different formats of ZigBee frame data for different purposes. The most important and used at the implementation of the update procedure are: ZigBee Transmit frame and ZigBee Receive frame.

To visit each network device for executing the firmware update procedure would require time and would also increase the total cost of the system [4, 5].

The developer Ladyada has presented a very simple solution of wireless programming, dedicated to Arduino platforms used together with XBee modules [6]. The solution could be easily implemented by configuring accordingly the XBee modules and modify some electrical connections. The single problem of this solution is the fact that the XBee modules can be used only in AT mode of communication and API mode, more complex and more useful for the most of the applications.

A solution of firmwares update, but not dedicated for Arduino platforms, is described by [7] and it supposes to use a supplementary microcontroller for programming the main microcontroller. The cost of the device will become higher, if additional components are necessary for this task, and for a platform already implemented will be difficult to connect a new microcontroller to it. Another solution referred to modular devices considers replacing the processing unit with another one having already installed the new firmware. The old processing unit would be send to the maintenance team for update and used then for other devices [8]. This is not so easily or impossible to be done when the device is not by far reachable.

Other solutions have been provided for different types of end nodes, but these are not suitable for our requirements. We have taken in consideration and use the advantages offered by our platform. Since we didn't found an already implemented solution of remote firmware update to be used with our network devices, we designed and implemented a new solution based on In-Application Programming (IAP). This type of programming has been considered as future update also by [7]. The IAP solution doesn't require any additional components and existing communication protocols could be used for this purpose. It is a simple way of reprogramming a device and the single condition of using IAP is to be supported by the microcontroller.

3 Methodology

3.1 In-Application Programming

The In-Application Programming (IAP) refers that the update of the microcontrollers' firmware application is done using the existent communication interfaces (e.g. UART, SPI, USB, Ethernet), without removing the microcontroller from the board where it is attached. When it is powered on, the bootloader application takes initially the control, do the necessary initializations and when it could execute directly the firmware application or enter the IAP code. The necessary data, representing the content of the new firmware application, is sent to the bootloader application using one of the available communication interfaces. The bootloader will take the data from the communication channel, will process it and will write it into the microcontroller memory. The IAP general flow format is presented in Fig. 5 [9].

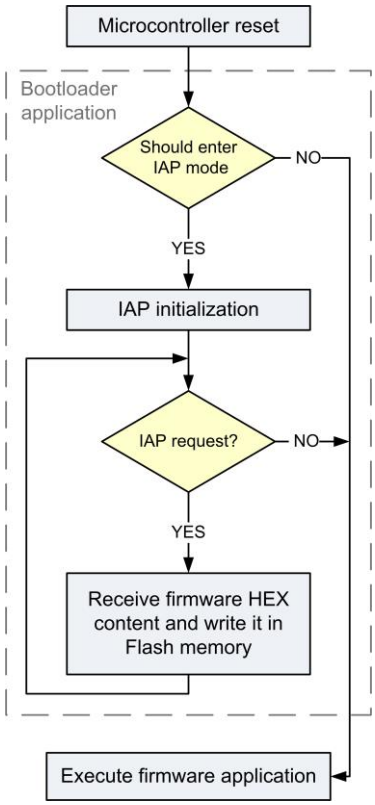

Figure 5
Implementation of In-Application Programming (IAP)

3.2 Firmware Application Format

Since the Seeeduino Stalker platform is compatible Arduino, the applications (sketchs) could be easily implemented using the Arduino programming environment. In the simplest way an application is compiled and uploaded on the platform using the options offered by the Arduino environment. The direct upload procedure is possible when the platform is connected to the computer. This method is not functional for remote firmware update.

When the compilation of the Arduino application is complete, it is created an associated HEX file in the temporary directories of the operating system. Other way to obtain the HEX file supposes to compile the application from the command line, but this procedure becomes complicated when we have applications with more files (classes) attached to the same sketch. The easiest way is to copy the HEX file already created from the temporary folder and use it further where it is necessary. The HEX file respects the Intel HEX file format and it contains all the information about the firmware application.

The Intel HEX format supposes that a file contains text disposed on lines and each line of text represents one HEX record. Each record starts with the ':' character followed by hexadecimal values. The general format of a HEX record is presented in Fig. 6.

Figure 6
HEX record format

Each group of letters represents a specific field of the HEX record and each letter represents a single hexadecimal digit. The description of the fields is:

- '**:**' - character which marks the begin of a record;
- **ll** - the number of data bytes (**dd**) of the record;
- **aaaa** - the memory address where the data bytes to be written;
- **tt** - the record type. The most important types are: 0x00 for data record and 0x01 for end-of-file record;
- **dd** - one byte of data. A record could include multiple data bytes, according to the length specified into the field **ll**;
- **cc** - checksum of the record [10].

4 Remote Firmware Update - Seeeduinoboot

The remote firmware update procedure supposes that the firmware application installed on a network device could be easily modified from distance, using different communication protocols.

There are 3 requirements to be considered before starting to implement the procedure:

- the execution environment capabilities: the destination/target device on which the update should occur;
- protocols which could be used for the transfer of the new firmware application content from the source to the destination (e.g. ZigBee protocol, through the API communication mode supported by XBee modules);
- and possibilities of minimizing the quantity of data transferred between device. The size reduction should occur on the computer connected to the network and it should not complicate the processing of data on the target device [12].

Normally, the maintenance computer (Driver) is connected to the wireless sensor network at the coordinator node level. The coordinator node ensures the link between WSN and the distribution and storage subsystem. The Driver application, installed on the Driver computer permits to configure the network and to transfer data in both directions. It also implements the functionality of remote firmware update: it takes the useful information from a specified HEX file and sends it to the target node. Fig. 7 presents, in a simplified mode, the connection between the maintenance computer and a target device (end node).

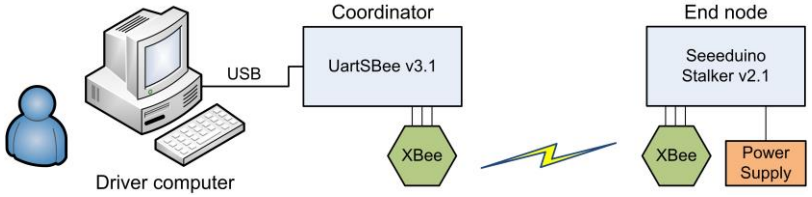

Figure 7
Connection between the maintenance computer and target device

In the current implementation of the Driver application there is a dialog, presented in Fig. 8, dedicated for the update functionality. It is working for a single network device at a time. The user has the possibility to select the HEX file corresponding to the new firmware application and to view the file content. During the update procedure, it is displayed the total number of bytes updated at the destination device, the time elapsed from the beginning of the update procedure and the estimated time, necessary for the procedure to complete.

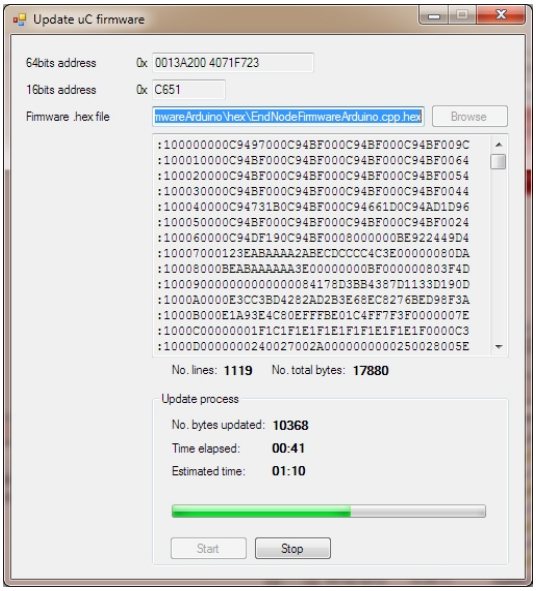

Figure 8
Remote firmware update dialog

The procedure of remote firmware update involves the next steps:

- send a remote command to the destination device for resetting the microcontroller. Since the execution of the bootloader application doesn't take too long time to complete (the watchdog is set on a short time - about 2 seconds), it is necessary that the update telegrams to arrive as soon as possible to the destination. Any delay into the communication procedure could lead to an incomplete update.
- retrieve the next data bytes to be sent and calculate the address from where they will be written into the memory;
- create a proper ZigBee Transmit frame, with the address and data bytes determined at previous step. The frame is included into a telegram which is then send to the destination device;
- at the destination device parse the ZigBee Receive telegram for getting the useful information, and update the Flash memory accordingly;
- send back a ZigBee telegram as a response if the update of the memory has been successfully or not.

Except the first step, all the other steps are repeated until all data from the new firmware content is send successfully to the destination node.

By default, on the Seeeduino Stalker v2 platform, it is installed a bootloader application available also for Arduino platforms. It implements the possibility of writing the firmware application using the Arduino environment and it doesn't implement anything about wireless communication.

Some developers have been optimized the bootloader application, created initially by the Arduino team, adding support for STK500 programmer and other improvements. Their new version of bootloader was called Optiboot [13]. Neither this version support wireless communication through the XBee modules. To simplify the work of updating the bootloader application, for adding the wireless communication functionality, we started by modifying the sources of Optiboot version 4.4.

The new bootloader application, Seeeduinoboot[1], includes a few essential steps, as it is also presented in Fig. 9:

- device initialization (watchdog timer and serial communication). When a timeout of the watchdog occurs the firmware application will be started.
- into an infinite loop the application will wait for any incoming telegrams on the serial port. The serial communication is directly linked to the wireless communication. The watchdog timer is reset each time a new telegram is received.
- when a new telegram is received it is verified to be in ZigBee Receive frame format and the integrity of the data is also checked. If everything is correct the telegram is parsed for retrieving the useful information.
- the frame type is checked, to be sure that the telegram represents a request for updating the firmware application.
- if it is an updating request it is determined the memory addres and the data bytes to be written into the memory;
- the corresponding memory page has to be read and stored into the programming buffer (data space dedicated for memory update). The unmodified part of the memory page will be preserved.
- the memory page is erased before writing, for preventing any data corruption.
- update the programming buffer with the information received on the serial telegram;
- write the memory page with the new data from the programming buffer. The erase and write operations have to refer to the same page [11].

[1] The Seeeduinoboot application is available in two versions, for downloading and testing, at the web address: http://code.google.com/p/seeeduinoboot.

- create a ZigBee Transmit telegram with the result of the update operation and transmit it back to the source device.

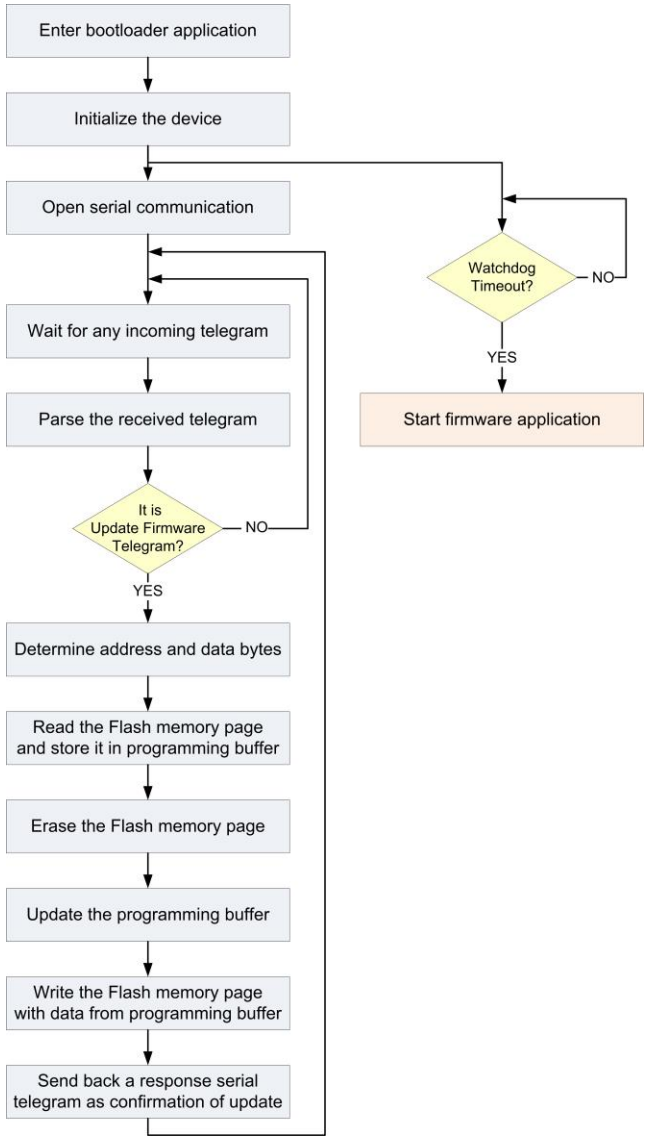

Figure 9
The main steps of the Seeeduinoboot application

5 Results

Each telegram (package) used for wireless communication is constructed by 2 sections: a header which contains the information necessary for routing the telegram through the network, and a payload section which contains the useful data (the data necessary to be transferred). The payload size is variable, according to necessities, but it should not exceed a maximum predefined length.

For the analysis presented further a firmware application of about 35 Kbytes long has been considered. The content of the application has been split in small telegrams and transferred over a single hop of the network, to the target device.

The direct firmware update of the network device, using a programmer device, required about 3 seconds. The remote firmware update procedure requires much longer to accomplish, but we have the advantage of mobility and easy maintenance. The most part of the time is spent for transferring the data between the devices (data dissemination).

According to the measurements, performed on our system, we have concluded that it takes about 250 ms to send a single telegram (package) of data over a single hop of the network. There is no visible influence of the telegram length on the transfer time and the most of the time is spent waiting for the destination device to get ready for receiving data. As it is presented in Fig. 10 the dissemination time decreases as long as larger payload is used, since the total number of telegrams necessary to send the same quantity of data is reduced.

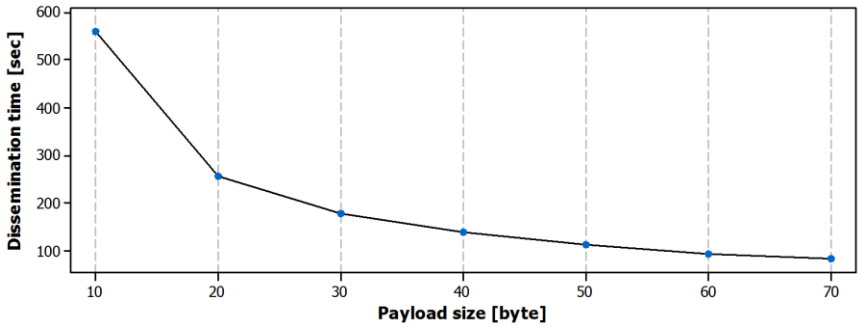

Figure 10
Variation of the dissemination time according to payload size

Fig. 11 presents the total quantity of data necessary to be transferred, considering also the data transferred as telegram headers. For larger payload the overhead is much reduced.

For more complex analysis the communication errors effect should be also considered, the larger telegrams have increased chances to be received with errors than the smaller telegrams.

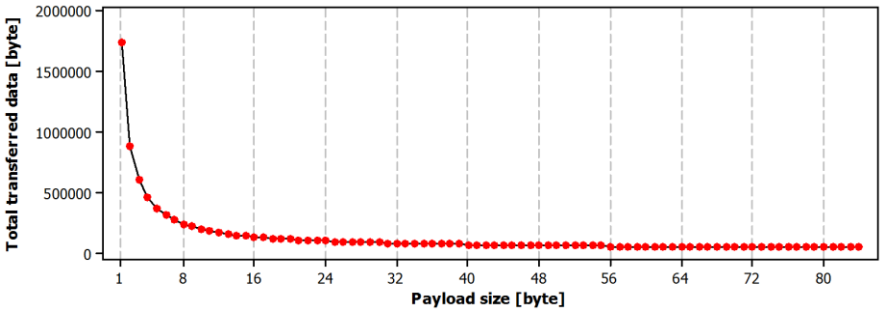

Figure 7
Variation of the total transferred data according to payload size

Conclusions

The Seeeduinoboot is a new bootloader application, designed and developed to be used together with the network devices constructed with Seeeduino Stalker v2 platform, as main processing unit, and XBee Series 2 module, as wireless communication unit.

Since into a WSN the end nodes could be deployed into a large space area, where the access to each device could be limited, it is very important to have the possibility of remote firmware update. The firmware application monitors and controls that the device is working properly, and as the bootloader application, it is stored into the microcontroller's Flash memory. The remote firmware update functionality is included in Seeeduinoboot application and it was the main reason of modifying the original bootloader application. In many situations of testing the network devices it appears the necessity of bugs correction, for adding new functionalities to the existing firmware application of just for modifying the existing functionalities. The remote firmware update simplifies the maintenance process, reducing in the same time the cost of the system.

The solution proposed is using In-Application Programming (the update of the firmware is done from the bootloader application) and it could be implemented in the same way for other devices with similar characteristics. New optimizations to the Seeeduinoboot are planned for the future, mainly for minimizing the quantity of data transferred during the wireless communication, minimizing in this way the energy consumption and extending the life time of the devices.

Acknowledgement

This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD) financed from the European Social Fund and by the Romanian Government under the project number POSDRU / 89 / 1.5 / S / 59323.

References

[1] R. Zurawski: Embedded Systems Design and Verification, CRC Press, Taylor & Francis Group, ISBN: 978-1-4398-0761-3, 2009

[2] C. Bujdei and S.-A. Moraru: A Low Cost Framework Designed For Monitoring Applications, Based On Wireless Sensor Networks, Proceedings of the 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) 2012, Brasov, Romania, 2012, pp. 1211–1220

[3] R. Faludi: Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing, O'Reilly Media, ISBN: 978-0-586-80773-3, 2010

[4] M. Overgaard: Remote, Reliable Firmware Upgrade on PICMG Board Management Controllers, CompactPCI and AdvancedTCA Systems, Vol. 9, No. 4, 2005, pp. 40-44

[5] T. Stathopoulos, J. Heidemann and D. Estrin: A Remote Code Update Mechanism for Wireless Sensor Networks, CENS Technical Report 30, 2003

[6] Ladyada: XBee radios - Wireless Arduino programming/serial link, http://www.ladyada.net/make/xbee/arduino.html, 2011

[7] K. Benkic, M. Malajner and Z. Cucej: AeWSN.NET: Framework for the Wireless Sensor Networks, Workshop - New Master Curricula and EU Practice, Nis, Serbia, 2008

[8] A. Ghobakhlou, S. Zandi and P. Sallis: Development of Environmental Monitoring System with Wireless Sensor Networks, International Congress on Modelling and Simulation (MODSIM2011) Perth, Australia, 2011, pp. 1125-1131

[9] STMicroelectronics: STM32F107 In-Application Programming (IAP) over Ethernet, Application Note, 2010

[10] Keil: Keil, Tools by ARM - Intel Hex File Format, http://www.keil.com/support/docs/1584, 2011

[11] Atmel: ATmega328P, http://www.atmel.com/images/doc7810.pdf, Technical Datasheet, 2011

[12] S. Brown and C. J. Sreenan: Updating Software inWireless Sensor Networks: A Survey, Technical Report UCC-CS-2006-13-07, 2006

[13] Optiboot: Optiboot - An optimised bootloader for Arduino platforms, http://code.google.com/p/optiboot, 2012