Standards for Low Range Communication in Wireless Sensor Nodes

Milen Todorov[1], Boyanka Nikolova[2], Georgi Nikolov[3], Milena Terzieva[4]
Technical University of Sofia
8 Kl. Ohridski Blvd
Sofia 1000, Bulgaria
{bnikol@tu-sofia.bg}

ABSTRACT: *In the wireless sensor networks, for short range communications, the most used standard is the Zigbee. This standard ensures mesh networking, low-power, low-complexity, reliability and operation on unlicensed frequency band. We through this work have introduced a model to produce a low-cost sensor node using open-source hardware and software platforms. We have used a specific platform to do this development.*

Keywords: Arduino, IEEE 802.15.4, Wireless Sensor Networks, XBee, XCTU, ZigBee

Received: 30 September 2020, Revised 9 January 2021, Accepted 20 January 2021

DOI: 10.6025/jitr/2021/12/2/41-48

1. Introduction

While numerous sensors are connected directly, through the use of existing local area networks, with controllers and data processing stations, the numbers of sensors that send information wirelessly are increasing. This is important because many network applications require hundreds or thousands of sensor nodes, often located in remote and inaccessible areas. Therefore the sensor node, except the sensor element should have processing and data storage capabilities. Sensor networks are ideal for all forms of environmental monitoring. Due to the sensors' small size, low energy requirements, and low-cost, implementers can install them at sites or at specific stations or machines for precise reporting. Sensors can be used to measure observations at key locations, the measured data is sent to a computer or even to a server in the cloud.

The microcontroller platform in this paper is Arduino. This is an open-source electronics platform based on hardware and software. The most significant advantages of Arduino over other systems are:

· Relatively inexpensive compared to other microcontroller platforms;

· Cross-platform – The Arduino Software (IDE) runs on different operating systems;

· Simple and clear programming environment;

· Open source and extensible software. The language can be expanded through C++ libraries;

· Open source and extensible hardware – Circuit designers can make, extend and improve the module.

ZigBee is a standard that defines a set of communication protocols for low-data-rate short-range wireless networking. The standard is developed by the ZigBee Alliance, which has hundreds of member companies, from the semiconductor industry and software developers to original equipment manufacturers (OEMs) and installers. The ZigBee standard uses IEEE 802.15.4 as its Physical Layer (PHY) and Medium Access Control (MAC) protocols. IEEE 802.15.4 uses spreading methods to improve the receiver sensitivity level, increase the jamming resistance, and reduce the effect of multipath. The signal spreading by the transmitter and despreading by the receiver reduce the effect of the interferers [1-3].

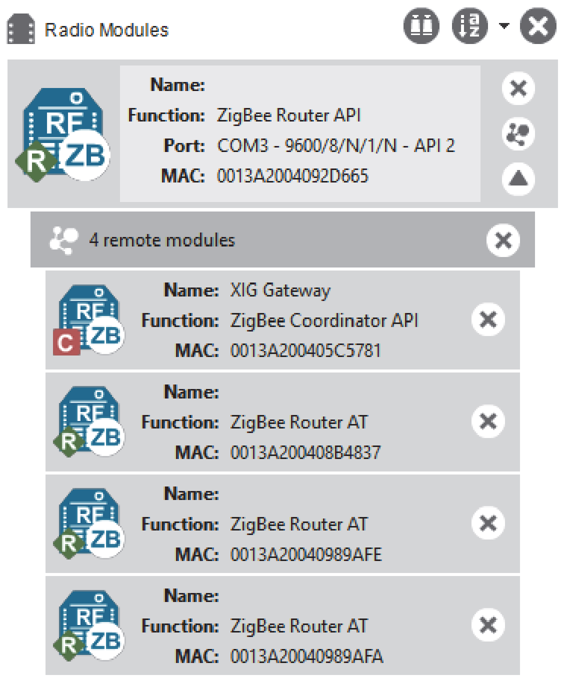
2. Sensor Node Configuration

Arduino boards do not come with ZigBee connectivity. The proposed option in this paper is to use a shield. A shield is basically an extension board that can be placed on top of the Arduino board. Shields are used to extend the hardware features of the Arduino. Another option is to use an external component, mounted on a breakout board, which are connected to Arduino. The used digital relative humidity and temperature sensor RHT03 measures the concentration of water (moisture) in the air. Humidity sensors react to these phenomena and generate a voltage that the microcontroller of Arduino read and calculate a value on a scale. A basic, lowcost humidity sensor is the DHT-22. The DHT-22 is designed to measure temperature as well as humidity. It generates a digital signal on the output (data pin). It should be used to track data at a reasonably slow rate (no more frequently than about once every 3 or 4 seconds). When this sensor generates data, that data is transmitted as a series of high and low voltages that the microcontroller reads and use to form a value. In this case, the microcontroller reads a value 40 bits in length. The first two bytes are the value for humidity, the second two are for temperature, and the fifth byte is the checksum value to ensure an accurate read [4]. DHT-22 temperature and humidity sensor requires pull up resistor to pull up the data value to the voltage level to ensure a valid logic on the wire. The sensor is temperature compensated and calibrated in accurate calibration chamber and the calibrationcoefficient is saved in type of programme in one-time programmable (OTP) EPROM. When the sensor is detecting, it will read this coefficient from memory [5]. The sensor uses its own protocol.

The XBee and XBee-PRO Radio Frequency modules meet IEEE 802.15.4 standards and support the needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices. The modules operate within ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other [6, 7]. The XBee/XBee-PRO modules interface to a host device through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART or through a level translator to any serial device (e.g. RS-232, USB interface board). By default modules operate in Transparent Mode. In this mode, the modules act as a serial line replacement – all UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin. If the module can not immediately transmit, the serial data is stored in the DI Buffer. The data is packetized and sent at any RO timeout or when maximum packet size are received. If the DI buffer becomes full, hardware or software flow control must be implemented in order to prevent overflow – loss of data between the host and module. Application Programming Interface (API) Operation is an alternative to the default Transparent Operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module. When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module. The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART or through a level translator to any serial device (e.g. RS-232, USB interface board). By default modules operate in Transparent Mode. In this mode, the modules act as a serial line replacement – all UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin. If the module can not immediately transmit, the serial data is stored in the DI Buffer. The data is packetized and sent at any RO timeout or when maximum packet size are received. If the DI buffer becomes full, hardware or software flow control must be implemented in order to prevent overflow – loss of data between the host and module. Application Programming Interface (API) Operation is an alternative to the default Transparent Operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module. When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module. The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and payload information instead of using command mode to modify addresses [8]. By default XBee modules are configured to operate within a peer-to-peer network topology, i.e. modules are synchronized without use of master/server configurations. A peer-to-peer network can be established by configuring each module to operate as an End Device, disabling End Device Association in all modules and setting ID and CH parameters to be identical across the network.

3. Network Deployment

Figure 1 shows the discovered radio module that is connected to the computer and remote radio modules in the same network as the local module.



Radio Modules

Name:
Function: ZigBee Router API
Port: COM3 - 9600/8/N/1/N - API 2
MAC: 0013A2004092D665

4 remote modules

Name: XIG Gateway
Function: ZigBee Coordinator API
MAC: 0013A200405C5781

Name:
Function: ZigBee Router AT
MAC: 0013A200408B4837

Name:
Function: ZigBee Router AT
MAC: 0013A20040989AFE

Name:
Function: ZigBee Router AT
MAC: 0013A20040989AFA

Figure 1. Detected local and remote modules

The detected network topology is shown of Figure 2. The network perspective is only available in API operating mode of detecting module. Radio modules in AT (transparent) mode do not support the network discovery process. Each module is labelled with its role (C – coordinator, R – router).

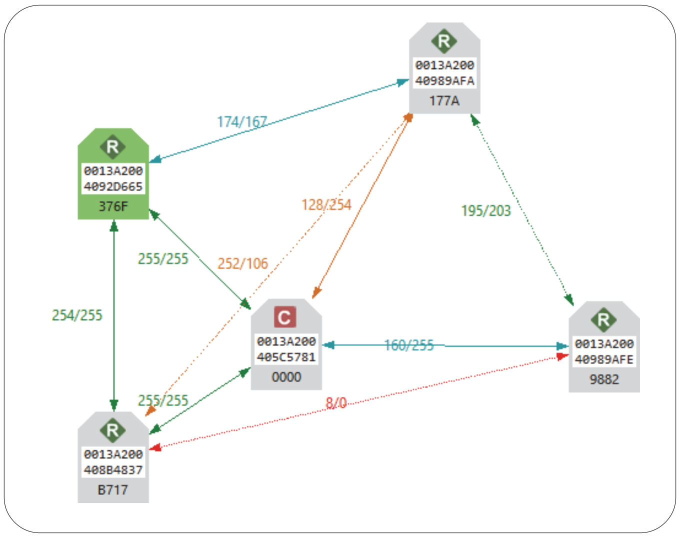
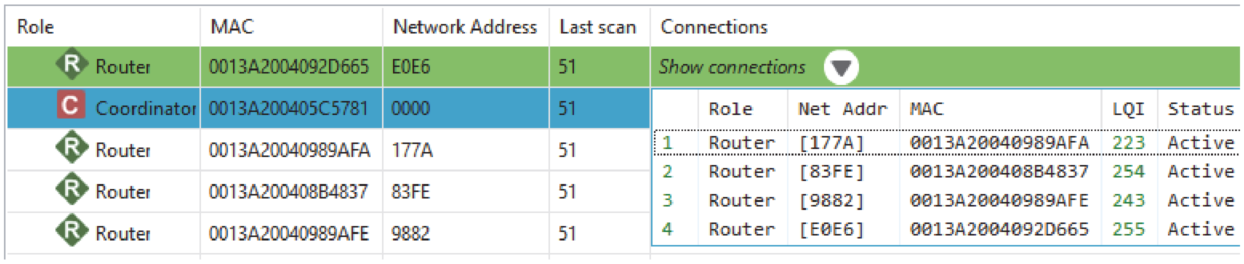


Figure 2. Network topology

As can be seen from the figure each node is connected to its neighbours with solid lines (active connections) or dotted lines (undiscovered connections), with arrows indicating the direction of communication. The bidirectional quality and status of the connection between two nodes are displayed next to the line that connects them. The link quality is represented by Link Quality Indication (LQI) with number between 0 and 255 where 0 is the weakest and 255 is the strongest. Fig. 3 shows table view of detected radio modules. The green background (first row) denotes the local radio module used for the detection. The blue background (second row) denotes the selected module. The table gives the role of each device, 64-bit address of the modules, network address (for ZigBee network) or node identifier (for networks working with other protocols), scan number when the devices were last discovered. Even when a radio module leaves the network, some devices continue to store information about their relationship. On Connection column can be seen the remote modules which are connected to the selected module. Also are displayed their: role in the network, 64-bit address, LQI and status.

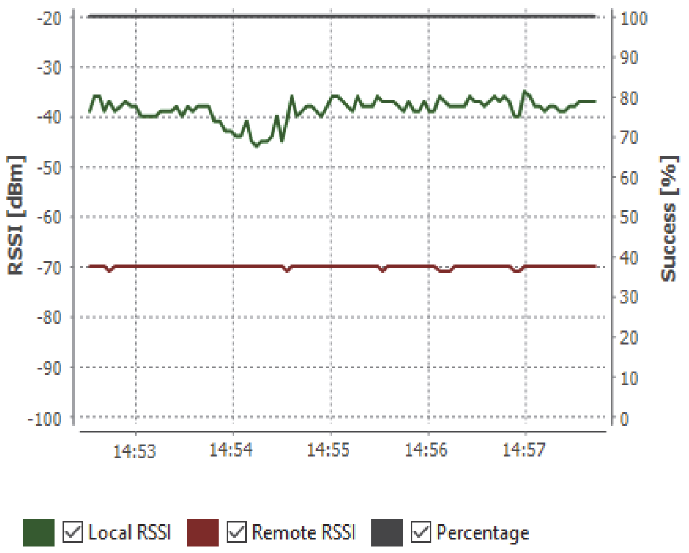


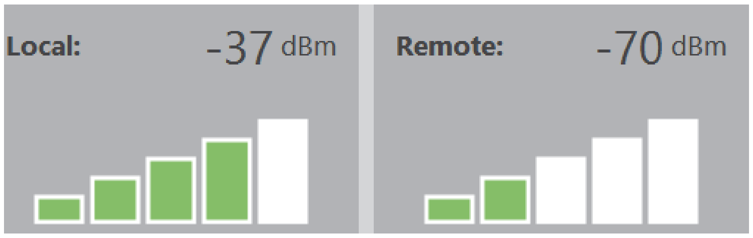
Role	MAC	Network Address	Last scan	Connections
Router	0013A2004092D665	E0E6	51	Show connections
Coordinator	0013A200405C5781	0000	51	
Router	0013A20040989AFA	177A	51	
Router	0013A200408B4837	83FE	51	
Router	0013A20040989AFE	9882	51	

	Role	Net Addr	MAC	LQI	Status
1	Router	[177A]	0013A20040989AFA	223	Active
2	Router	[83FE]	0013A200408B4837	254	Active
3	Router	[9882]	0013A20040989AFE	243	Active
4	Router	[E0E6]	0013A2004092D665	255	Active

Figure 3. Table view of detected nodes

The quality of the wireless signal can be affected by many factors, most important of which are absorption, reflection of waves, line of sight issues, antenna style and location. Therefore was carried out a range test, which shows the radio frequency (RF) range and link quality between two selected XBee modules. The LQI is an indication of the quality of the data packets received by the receiver and it is recorded for each received packet, indicating the signal energy or the signal-to-noise ratio. The LQI is only one of the decision factors in selecting a path to route a message. A method to calculate the link cost is to use a lookup table to map different levels of LQI directly to the link cost levels of 0 to 7. The table is created based on the average results of several experiments. Other factors such as routing energy efficiency considerations, can also influence the route selection [2]. Fig. 4 shows the results from the test. During the test the local module send data packets and waits for the echo from the remote module. The XCTU software tool counts the number of the packets sent and received and measures the signal strength of the both sides as a Received Signal Strength Indicator (RSSI) value. The performed test type is Cluster ID 0x12, which uses explicit addressing frames directed to the Cluster ID 0x12 on the data endpoint 0xE8, which returns the received data to the sender. As can be seen from the figure, during the test session were sent 100 packets with transmit interval of three seconds and response timeout before considering a packet to be lost also three seconds.





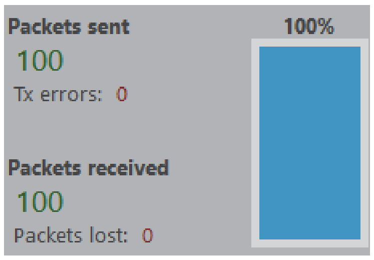
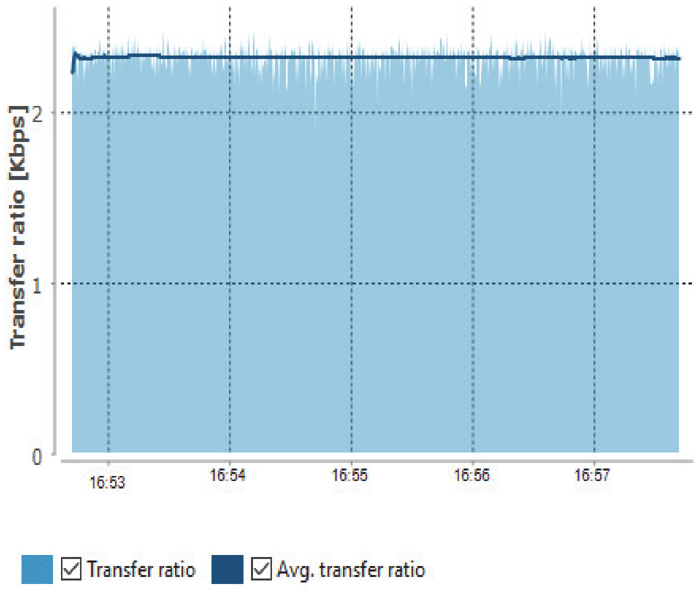

Packets sent 100%
100
Tx errors: 0
Packets received
100
Packets lost: 0

Figure 4. Results from range test tool

Figure 5 shows the results from Throughput tool of XCTU, which measures the transfer rate between two radio modules in the ZigBee sensor network. The throughput type was bidirectional – Cluster ID 0x12. It is important to note that not all protocols and operation modes support Bidirectional – Cluster ID 0x12 Throughput type of test. The duration of the test was 300 s and were sent 1034 packets with 86856 bytes.



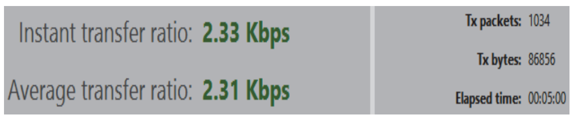

Instant transfer ratio: 2.33 Kbps
Average transfer ratio: 2.31 Kbps
Tx packets: 1034
Tx bytes: 86856
Elapsed time: 00:05:00

Figure 5. Results from the throughput session

Figure 6 shows the collected data from a single sensor node in the ZigBee network.

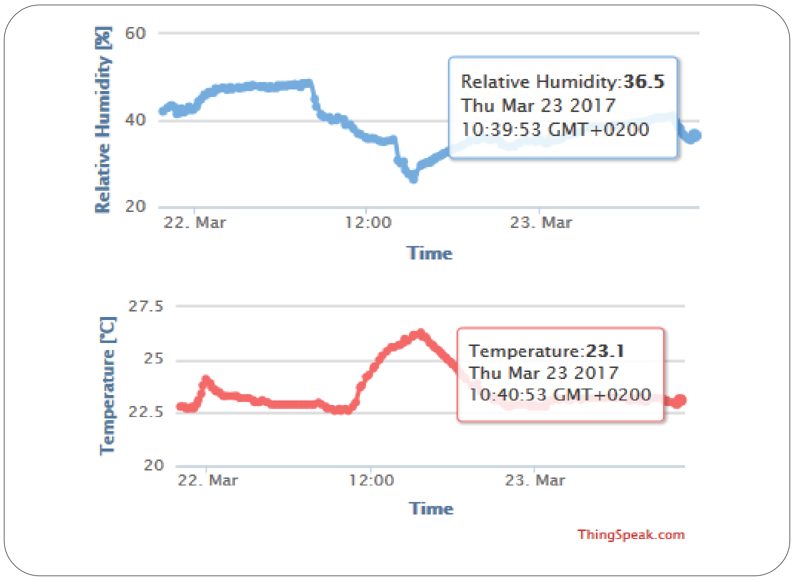


Figure 6. Measured relative humidity and temperature

4. Acquiring Data In Labview Environment

The presented approach can be extended for use in graphical programming environments. Such environment is LabVIEW, that is a high level programming language, which inherits the internal structures of C language, but offers more simplicity and functionality about making signal processing advantage that LabVIEW offers is that it is a graphical language [9, 10]. The programs in this language have, from the user side, the same appearance as a front panel of an electronic device, with buttons, graphic screens, numerical indicators, etc.

LINX is a free software package installed in LabVIEW. This package makes easy to use graphical programming language for interacting with some embedded platforms such as Arduino, chipKIT and myRIO. With LINX installed in LabVIEW it is possible to easily access the device's digital inputs and outputs, analog inputs and outputs, SPI, I2C, UART, PWM and more features. Inside LINX functions, there are several options for the user to choose. The *Open* and *Close* functions are used to start and end the communication with the Arduino microcontroller. In palette *Peripherals* the user finds different options for general digital and analog inputs or outputs. There are also different options for PWM, I2C, SPI and UART communication. The user can also find sensors functions that are already prepared to work with specific sensors. There are different types such as temperature sensors, light sensors, motion sensors, etc.

In the presented work as example for using Arduino as wireless sensor node in LabVIEW, PIR movement sensor HCSR501with

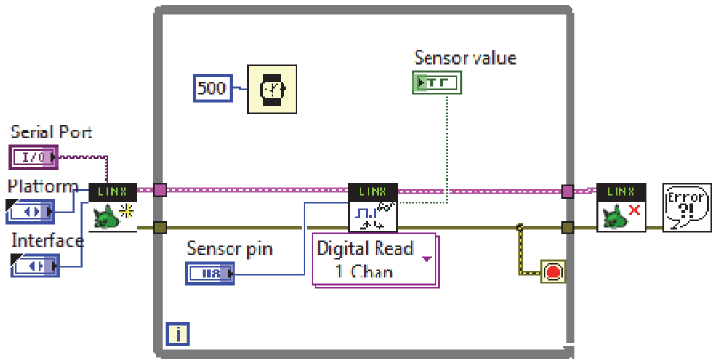
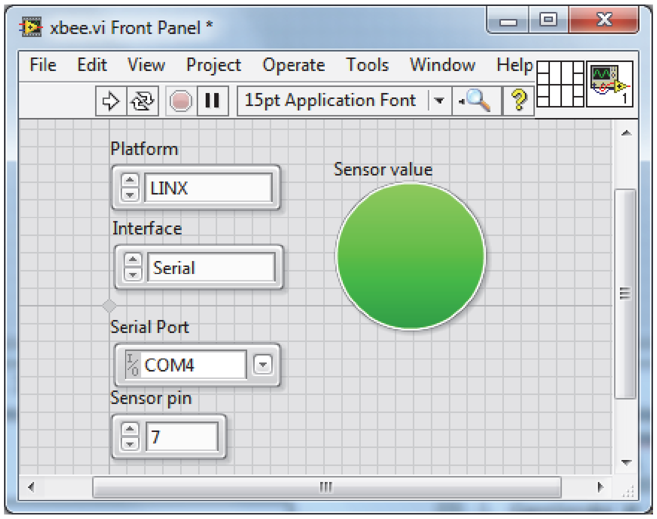


Figure 7. LabVIEW block diagram for communication with Arduino and Xbee explorer

digital output is connected to Arduino. Movement detectors based on this PIR sensor could be used for automatically sensing light for a room, bathroom, basement, porch, warehouse, garage, etc. It can also be used to get ventilators to work. In safety applications, obviously it can work as an alarm to detect when a person is entering some place. Actually, the wireless network topology is the same as in Figure 2. The difference is in the graphical way to send command and receive data from sensor nodes.

Graphical programing code with specific LINX functions is shown in Figure 7. As can be seen in the figure only three functions are needed to receive data from wireless node: Open, Close and Read Digital.

The front panel of wireless communication with sensor is shown in Figure 8. It is used to control only virtual serial port and number of digital pin connected to the motion detection sensor.



Platform: LINX
Interface: Serial
Serial Port: COM4
Sensor pin: 7

Figure 8. Front panel of application for motion detection

5. Conclusion

In present project is suggested an approach for organizing wireless sensor network that use communication protocols according to the standard ZigBee. The proposed and implemented approach extends the performance of sensors, by using open source platform for development of low-cost wireless sensor nodes. In addition as alternative method for wireless communication between the sensor nodes, programing code in LabVIEW graphical programming Environment Is Created.

Acknowledgement

This research is supported by Project No 171П0013-07.

References

[1] Gavrilovska, L., et al. (2011). Eds. *Application and Multidisciplinary* Aspects of Wireless Sensor Networks: Concepts, Integration, *and Case Studies*, Springer, 2011.

[2] Farahani, S. (2008). *ZigBee Wireless Networks and Transceivers*, Elsevier Ltd., 2008.

[3] ZigBee Specification, 053474r17, January 2008.

[4] Bell, C. (2013). *Beginning Sensor Networks with Arduino and Raspberry Pi*, A press, 2013.

[5] MaxDetect Technology Co., Ltd., Digital relative humidity & temperature sensor RHT03, RHT03 datasheet.

[6] Dargie, W., Poellabauer, C. (2010). *Fundamentals of Wireless Sensor Networks: Theory and Practice*. John Wiley &

Sons Ltd., 2010.

[7] Eady, F. (2007). *Hands-On ZigBee: Implementing 802.15.4 with Microcontrollers*, Elsevier Inc, 2007.

[8] Digi International, XBee/XBee-PRO RF Modules 802.15.4 Product Manual, 2015.

[9] Schwartz, M., Ol. Manickum. (2015). *Programming Arduino with LabVIEW*, Packt Publ., 2015.

[10] Larsen, R. W. (2011). *LabVIEW for Engineers*, Prentice Hall Publ., 2011.