DEVELOPING A ZIGBEE WIRELESS NETWORK AND CONTROLLING IT THROUGH THE INTERNET

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KEREM KAYNAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

MAY 2009

Approval of the thesis:

DEVELOPING A ZIGBEE WIRELESS NETWORK AND CONTROLLING IT THROUGH THE INTERNET

submitted by **KEREM KAYNAR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering** _______________

Dr. Attila Özgit
Supervisor, **Computer Engineering Dept., METU** _______________

Examining Committee Members:

Prof. Dr. Payidar Genç
Computer Engineering Dept., METU _______________

Dr. Attila Özgit
Computer Engineering Dept., METU _______________

Assoc. Prof. Dr. Cem Bozşahin
Computer Engineering Dept., METU _______________

Dr. Onur Tolga Şehitoğlu
Computer Engineering Dept., METU _______________

Mert Özarar, M.Sc.
Computer Engineering Dept., METU _______________

Date: _______________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Kerem KAYNAR

Signature :

ABSTRACT

DEVELOPING A ZIGBEE WIRELESS NETWORK AND CONTROLLING IT THROUGH THE INTERNET

Kaynar, Kerem
M.Sc., Department of Computer Engineering
Supervisor: Dr. Attila Özgit

May 2009, 149 pages

The aim of this thesis is to develop a network, whose nodes communicate with the ZigBee wireless network protocol, and control this network with a PC through the Internet. One of the nodes of this network is designed to be master node. The other nodes are slave nodes. The master node can be connected to an Ethernet connected to the Internet. A PC can communicate with the master node via a specific web application over the Internet. The communication between a web server, in which the specific web application is loaded, and the master node is performed using a specific application protocol working over TCP/IP and defined in this thesis. The master node controls the slave nodes of the wireless network formed according to the commands given by the user of a PC over the Internet. The master node contains an implementation of the ZigBee stack along with a suitable application software to communicate with the slave nodes. The master node also contains an implementation of the TCP/IP stack along with a suitable application software to communicate with a web server in which the specific web application is loaded. The slave nodes contain an implementation of the ZigBee stack along with a suitable application software to communicate with the master node. For each type of node, appropriate hardware which is compliant with the software contained by that type of node is used. Each type of node uses microcontroller-based hardware.

Keywords: ZigBee, Wireless Network, Microcontroller, Ethernet, Internet

ÖZ

ZIGBEE KABLOSUZ AĞ PROTOKOLÜNÜ KULLANAN KABLOSUZ BİLGİSAYAR AĞI GELİŞTİRMEK VE BU AĞI İNTERNET ÜZERİNDEN KONTROL ETMEK

Kaynar, Kerem
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Attila Özgit

Mayıs 2009, 149 sayfa

Bu tezin amacı elemanları ZigBee kablosuz ağ protokolüyle haberleşen bir bilgisayar ağı geliştirmek ve bu ağı bir kişisel bilgisayar ile İnternet üzerinden kontrol etmektir. Bu kablosuz bilgisayar ağının bir elemanı ana eleman olarak tasarlanıyor. Diğer elemanlar ise yardımcı elemanlar oluyorlar. Ana ağ elemanı, Ethernet protokolünü kullanan ve İnternete bağlı olan bir bilgisayar ağına bağlanabilir. Bir kişisel bilgisayar bu ana ağ elemanı ile özel bir web uygulaması yoluyla Internet üzerinden haberleşebilir. Bu web uygulamasının yükleneceği web sunucusuyla ana ağ elemanı arasındaki iletişim bu tezde tanimlanan ve TCP/IP üzerinde çalışan özel bir uygulama protokolüyle gerçekleşir. Ana ağ elemanı yardımcı ağ elemanlarını bir kişisel bilgisayarın kullanıcısının İnternet üzerinden verdiği komutlara göre kontrol eder. Ana ağ elemanı yardımcı ağ elemanları ile haberleşebilmek için ZigBee protokol yığını ve uygun bir uygulama yazılımı içerir. Ana ağ elemanı özel web uygulamasının yükleneceği web sunucusuyla haberleşebilmek için TCP/IP yazılım yığını ve uygun bir uygulama yazılımı içerir. Yardımcı ağ elemanları ana ağ elemanı ile haberleşebilmek için ZigBee protokol yığını ve uygun bir uygulama yazılımı içerirler. Her bir ağ elemanı tipi için, o tip ağ elemanının içereceği yazılımla uyumlu olan donanım kullanılır. Her bir tip ağ elemanı mikrodenetleyici tabanlı donanım kullanır.

Anahtar Kelimeler: ZigBee, Kablosuz Ağ, Mikrodenetleyici, Ethernet, İnternet

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his supervisor Dr. Attila Özgit for his guidance, advice, criticism, encouragements and insight throughout the research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACL:	Access Control List
ADC:	Analog-to-Digital Converter
AES:	Advanced Encryption Standard
AIB:	APS Information Base
ALU:	Arithmetic Logic Unit
APDU:	Application Protocol Data Unit
APS:	Application Support Sublayer
APSDE:	APS Data Entity
APSME:	APS Management Entity
ARP:	Address Resolution Protocol
ASDU:	APS Service Data Unit
CAP:	Contention Access Period
CBC-MAC:	Cipher Block Chaining Message Authentication Code
CCA:	Clear Channel Assessment
CCM:	Counter with CBC-MAC Mode of Operation
CCM*:	Enhanced Counter with CBC-MAC Mode of Operation
CFP:	Contention-Free Period
COM:	Communication
COP:	Coprocessor
CPU:	Central Processing Unit

CSMA-CA:	Carrier Sense, Multiple Access with Collision Avoidance
CTR:	Counter Mode
DES:	Data Encryption Standard
3DES:	Triple DES
DHCP:	Dynamic Host Configuration Protocol
DNS:	Domain Name System
DoS:	Denial of Service
ED:	Energy Detection
EEPROM:	Electrically Erasable Programmable Read-Only Memory
EUSART:	Enhanced Universal Synchronous Asynchronous Receiver Transmitter
FFD:	Full-function Device
GHz:	GigaHertz
GPIO:	General Purpose Input Output
GTS:	Guaranteed Time Slot
HAN:	Home Area Network
HTML:	Hyper Text Markup Language
HTTP:	Hyper Text Transfer Protocol
IDE:	Integrated Development Environment
IEEE:	Institute of Electrical and Electronics Engineers, Inc.
IIC (I2C):	Inter-Integrated Circuit
I/O:	Input/Output
IP:	Internet Protocol
IRQ:	Interrupt Request Queue

IV:	Initialization Vector
KB:	Kilo Byte
Kbps:	Kilo Bits Per Second
LDAP:	Lightweight Directory Access Protocol
LLC:	Logical Link Control
LQI:	Link Quality Indication
LR-WPAN:	Low-Rate Wireless Personal Area Network
m:	Meter
MAC:	Medium Access Control
MAC:	Message Authentication Code
MCU:	Microcontroller Unit
METU:	Middle East Technical University
MHz:	MegaHertz
MIC:	Message Integrity Code
MLME:	Medium Access Control Sublayer Management Entity
MPDU:	Medium Access Control Protocol Data Unit
MSDU:	Medium Access Control Service Data Unit
NIB:	Network Information Base
NLDE:	NWK Layer Data Entity
NLME:	NWK Layer Management Entity
NPDU:	Network Level Protocol Data Unit
NWK:	Network
OSI:	Open Systems Interconnection
PAN:	Personal Area Network

PC:	Personal Computer
PDU:	Protocol Data Unit
PHY:	Physical
PIB:	Personal Area Network Information Base
PIC:	Programmable Interface Controller
PICDEM:	PIC Demonstration
PIM:	Plug-In Module
PLME:	Physical Layer Management Entity
POS:	Personal Operating Space
PPDU:	PHY Protocol Data Unit
PSDU:	PHY Service Data Unit
RAM:	Random Access Memory
RC4:	Rivest Cipher 4
RF:	Radio Frequency
RFD:	Reduced-function Device
RFIC:	Radio Frequency Integrated Circuit
RSA:	Rivest Shamir Adleman
RTC:	Real-time Clock
SAP:	Service Access Point
SCSI:	Small Computer System Interface
SPI:	Serial Peripheral Interface
SRAM:	Static Random Access Memory
SSCS:	Service Specific Convergence Sublayer

SSL:	Secure Sockets Layer
TCP/IP:	Transmission Control Protocol/Internet Protocol
TLS:	Transport Layer Security
Tx/Rx:	Transmitter/Receiver
UART:	Universal Asynchronous Receiver Transmitter
UDP:	User Datagram Protocol
WPAN:	Wireless Personal Area Network
XOR:	Exclusive Or
ZDO:	ZigBee Device Object
ZDP:	ZigBee Device Profile

CHAPTER 1

INTRODUCTION

In recent years, a great expansion of remotely controlled devices has appeared in people's daily life. Five years ago, remotely controlled televisions were the only such devices. Now, there are dozens of remotely controlled devices and appliances in the home. To gain the most efficiency from all these remotely controlled devices, they will need to be put under a single standardized control interface so that they can interconnect to a network, specifically a home area network (HAN). One of the most favorable HAN protocols is Zigbee, a software layer based on the IEEE 802.15.4 standard. (The application areas of ZigBee are not limited to HANs.) [3]

Devices such as TVs, garage door openers, light and fan controls generally support one-way, point-to-point control. The remotes that control these devices are not interchangeable and they do not support more than one device. Since most remotely controlled devices are not standardized among manufacturers, even those remotes used for the same function (like turning lights on and off) are not interchangeable with similar remotes from different manufacturers. In other words, you will have as many separate remote control units as you have devices to control. Some modern remotes enable you to control multiple devices by learning transmitting codes. But because the range for remote control is limited by line of sight, they are used generally for home entertainment control. A HAN can solve these problems because it does not require line-of-sight communication and it permits a single remote to command many devices. [3]

Most of the data that circulates within a network of home appliances are small packets that control devices or obtain their status. For many applications such as wireless smoke detectors or wireless home security, a device mostly stays in deep sleep mode and only sends a small amount of information if a trigger event occurs. The main requirements for devices in such types of networks are: low power consumption, the ability to sleep for a long time, high reliability, simplicity and low cost. A home area network should also support different configurations, such as star or mesh networks, to effectively cover a household area of 30 to 70 meters. ZigBee comes into play when taking these requirements into consideration. [2]

An improvement over the subject of HANs is to make a HAN controllable by the owner of the home from outside the home, for example from the working office of the owner of the home. By this improvement, the owner of the home can obtain the status of the home appliances and take the appropriate actions on these appliances (for example, in emergency situations) by means of his/her PC (Personal Computer) without even the need to be at home. For this improvement to be realized, a communication infrastructure must be chosen to connect a HAN to the PC (or PDA or any other thing) of the owner of the home in which the HAN is operating. This communication infrastructure may be TCP/IP over Internet. In this case, the owner of the home may control his/her HAN in the home from anywhere worldwide.

An issue about the improvement mentioned above is security. Only the owner of a HAN can control the HAN from anywhere by means of anything (a PC or PDA etc.) for the HAN to be secure. Any other entity shall not access to the HAN. Appropriate security measures must be taken to secure the communication mechanism between the owner of a HAN and the HAN. An element of the HAN must be chosen as a gateway to the outside world and this element must authenticate the owner of the HAN and secure the messages between the owner of the HAN and itself. By this way, no one can read or modify the messages transferred between the owner of the HAN and the HAN.

Another security consideration is to provide security inside a HAN. The network elements that wish to join to the HAN must be authenticated so that unwanted devices can not join to and destroy the HAN. Also, the data messages between the elements of the HAN must be secured so that they can not be read or modified by third parties. The master controller element of the HAN (chosen by the owner of the HAN) can adjust the security policies used across the HAN for controlling the network joining procedure and securing data messages.

In this thesis, a network whose nodes communicate with the ZigBee wireless data communication protocol is formed. This network represents a small-scale HAN. One of the nodes of this network is supposed to be the master controller. This node is called ZigBee master controller throughout the thesis. Other nodes are slave devices and are called ZigBee slave devices throughout the thesis. The ZigBee master controller controls ZigBee slave devices. ZigBee slave devices can not communicate directly with each other.

ZigBee master controller can be controlled by a PC (Personal Computer), connected to the Internet, through the Internet. The communication between the PC and the ZigBee master controller is performed via a web application implemented for this purpose. The web server in which this web application is loaded can directly communicate with the ZigBee master controller using a specific application protocol, defined and implemented in this thesis work, working over TCP/IP (Transmission Control Protocol/Internet Protocol).

The PC, connected to the Internet, can download the web application mentioned above into its browser by connecting to the web server it is loaded in. The user of the PC can give commands to the ZigBee master controller through this web application. The web application contains the necessary user interface elements for the user of the PC to give commands to the ZigBee master controller. The commands given by the user of the PC are relayed to the ZigBee master controller via the web server in which the web application is loaded.

The commands given by the user of the PC can be related with (targeted to) the ZigBee master controller or any of the ZigBee slave devices. If a command is related with the ZigBee master controller, the ZigBee master controller prepares a response and sends the response to the PC via the web server. If a command is related with any of the ZigBee slave devices, the ZigBee master controller relays the command to the related ZigBee slave device after receiving the command from the web server and then waits for a response from the ZigBee slave device. After receiving the response from the ZigBee slave device, the ZigBee master controller relays the response to the PC via the web server. The web application contains the necessary user interface elements used for displaying the response coming from the ZigBee master controller in a user-friendly way.

By this system, a person can control the ZigBee wireless network formed via a PC through the Internet. The PC must be connected to the Internet. The connection of the PC to the Internet may be wired or wireless. The location of the PC is irrelevant. There is no need to load a client application program to the PC for it to communicate with the ZigBee master controller. The user of the PC must only download the web application to the browser of his PC to start to communicate with the ZigBee master controller.

The web application and the ZigBee master controller are equipped with appropriate security software to allow only the owner of the ZigBee network (and possibly some other authorized users) to communicate with the ZigBee wireless network formed in this thesis. This security software is responsible for authenticating and authorizing entities wishing to communicate with the ZigBee wireless network. The security mechanisms employed by the security software relies mainly on the SSL (Secure Sockets Layer) protocol.

The ZigBee wireless network to be formed contains exactly one ZigBee master controller for controlling other ZigBee devices and coordinating the ZigBee wireless network. The ZigBee master controller acts as a bridge (gateway) between the web server and the ZigBee wireless network by relaying messages coming from the web server to the ZigBee devices in the ZigBee wireless network and relaying responses coming from the ZigBee devices in the ZigBee wireless network to the web server. When relaying a message or a response, the ZigBee master controller processes it and converts it into an appropriate format that is understandable by the target device.

The ZigBee master controller also handles the management of the ZigBee wireless network formed. It stores information about the ZigBee slave devices joined to the network, manages the joining of the ZigBee slave devices to the network and the leaving of the ZigBee slave devices from the network.

The ZigBee wireless network to be formed can have any number of ZigBee slave devices such that the number of all devices in the network does not go beyond the number (65536) that can be supported by the ZigBee wireless communication protocol. Two ZigBee slave devices are implemented in this thesis work for demonstration purposes. Each of these ZigBee slave devices must implement the logic of a specific type of device that can be controlled by an end user on a PC connected to the Internet. The commands that can be given by the end user to a specific ZigBee slave device are dependent on the device type logic implemented by that ZigBee slave device. Each of these ZigBee slave devices has the capability of finding the wireless network formed by the ZigBee master controller, joining to that network and leaving from that network.

The information given in the rest of the thesis can be described as follows. In Chapter 2 (Literature of Survey), there is a general outline of the ZigBee protocol and, information about the past and ongoing research and work performed in the application areas of ZigBee. In Chapter 3 (Proposed System Architecture), the overall operation of the system, the hardware and software architectures of the ZigBee master controller and ZigBee slave devices, the details of the specific application protocol used between the web server (in which the web application, implemented in the thesis, is loaded) and the Zigbee master controller, the architecture of the web application mentioned above, and the general security architecture of the entire system formed in the thesis are described.

Chapter 4 (Implementation Details) starts by describing the specific hardware components (boards, ICs etc.) used in the implementation of the ZigBee master controller and the ZigBee slave devices. It continues by revealing the details of the algorithms and data structures used in the ZigBee master controller, ZigBee slave device and web application software. It ends by discussing the implementation issues about the security mechanisms used throughout the entire system. In Chapter 5 (Performance), the performance metrics used for the evaluation of the system are explained, and the corresponding performance values are shown and interpreted. Lastly, Chapter 6 (Conclusion) explains the conclusions of the thesis work and gives clues about the future work (improvements) that can be performed about the thesis work. The weak points in the design and implementation of the system in the thesis work are also mentioned in this chapter.

CHAPTER 2

LITERATURE OF SURVEY

2.1 ZigBee Wireless Communication Protocol

In the system in the thesis, the communication between the ZigBee master controller and the ZigBee slave devices is performed through the ZigBee wireless communication protocol. Both the ZigBee master controller and the ZigBee slave devices have an implementation of a ZigBee stack in their structure. In this section, the details of the ZigBee protocol are described to the extent that is necessary and enough for this thesis.

The ZigBee stack architecture is made up of a set of blocks called layers. Each layer performs a specific set of services for the layer above by means of two entities: a data entity provides a data transmission service and a management entity provides all other services. According to [1], each service entity presents an interface to the upper layer through a service access point (SAP) and each SAP provides a number of service primitives to achieve the required functionality. The ZigBee stack architecture is depicted in the below figure. [1]

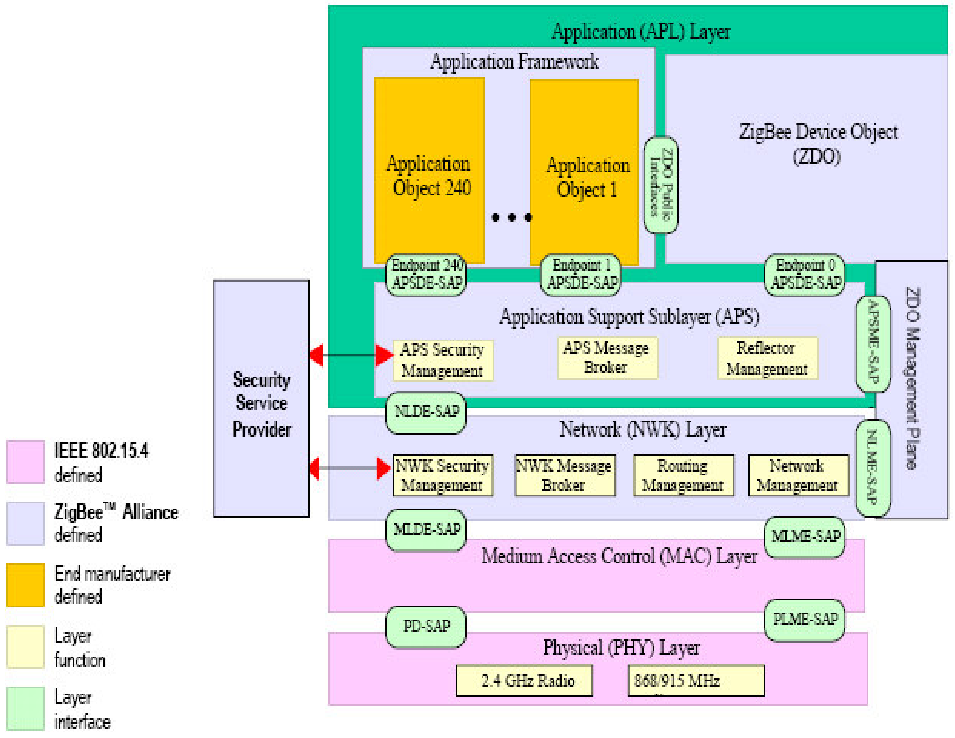


Figure 2.1: The ZigBee Stack Architecture [1]

The ZigBee stack architecture is based on the standard Open Systems Interconnection (OSI) seven layer model. The IEEE 802.15.4-2003 standard defines the two lower layers: the physical (PHY) layer and the medium access control (MAC) sublayer. The ZigBee Alliance builds on this foundation by providing the network (NWK) layer and the application layer. The application layer is comprised of the application support sublayer (APS), the ZigBee device object (ZDO) and the manufacturer-defined application objects. [1]

2.1.1 IEEE 802.15.4 Wireless Standard

IEEE 802.15.4 wireless standard contains medium access control (MAC) and physical (PHY) layer specifications for low-rate wireless personal area networks (LR-WPANs) [4]. Wherever the term “LR-WPAN” is used in this section, it is actually intended to mean LR-WPAN compliant with IEEE 802.15.4 wireless standard.

An LR-WPAN is a simple, low-cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. According to [4], the main objectives of an LR-WPAN are ease of installation, reliable data transfer, short-range operation, extremely low cost and a reasonable battery life while maintaining a simple and flexible protocol. According to [4], some of the characteristics of an LR-WPAN are:

- Over-the-air data rates of 250 kb/s, 40 kb/s and 20 kb/s
- Star or peer-to-peer operation
- Allocated 16 bit short or 64 bit extended addresses
- Allocation of guaranteed time slots (GTSs)
- Carrier sense multiple access with collision avoidance (CSMA-CA) channel access
- Fully acknowledged protocol for transfer reliability
- Low power consumption
- Energy detection (ED)
- Link quality indication (LQI)
- 16 channels in the 2450 MHz band, 10 channels in the 915 MHz band and 1 channel in the 868 MHz band [4]

Two different device types can be found in an LR-WPAN network: a full-function device (FFD) and a reduced-function device (RFD). An FFD can operate in three modes serving as a personal area network (PAN) coordinator, a coordinator or a device. An FFD can talk to RFDs or other FFDs. An RFD can talk only to an FFD.

An RFD is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor. A candidate RFD device in these type of applications does not have the need to send large amounts of data and can only associate with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity. [4]

Depending on the application requirements, the LR-WPAN can operate in either of two topologies: “the star topology or the peer-to-peer topology”. Both are shown in the below figure. [4]

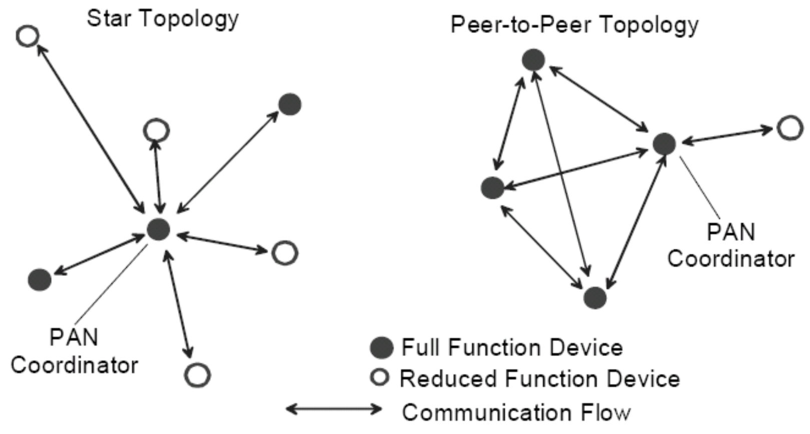


Figure 2.2: Star and peer-to-peer topology examples [4]

In the star topology, the communication is performed between devices and a single central controller, called the PAN coordinator. A device typically has some associated application and is either the initiation point or the termination point for network communications. A PAN coordinator can also have a specific application and it can be used to initiate, terminate or route communication around the network. The PAN coordinator is the primary controller of the PAN. All devices operating on a network of either topology shall have unique 64 bit extended addresses. This address can be used for direct communication within the PAN or it can be replaced with a short address allocated by the PAN coordinator when the device associates. The PAN coordinator may be mains powered. The other devices are most likely battery powered. Applications that can use a star topology include home automation, personal computer (PC) peripherals, toys and personal health care. [4]

The peer-to-peer topology also has a PAN coordinator; but, it differs from the star topology in that any device can communicate with any other device as long as they are in range of one another. Peer-to-peer topology allows more complex network formations to be established, such as mesh networking topology. Applications such as industrial control and monitoring, wireless sensor networks, asset and inventory tracking, intelligent agriculture and security could use such a network topology. A peer-to-peer network can be ad hoc, self-organizing and self-healing. It can also allow multiple hops to route messages from any device to any other device on the network. Such functions can be added at the network layer and are not part of IEEE 802.15.4 standard. [4]

Each independent PAN will select a unique identifier. This PAN identifier allows communication between devices within a network using short addresses and enables transmissions between devices across independent networks. [4]

The LR-WPAN architecture is defined in terms of a number of blocks in order to simplify the standard. These blocks are called “layers”. Each layer is responsible for one part of the standard and offers services to the higher layers. The layout of the blocks is based on the open systems interconnection (OSI) seven-layer model. The interfaces between the layers serve to define the “logical links” that are described in the IEEE 802.15.4 standard. [4]

An LR-WPAN device comprises a PHY, which includes the radio frequency (RF) transceiver along with its low-level control mechanism, and a MAC sublayer that provides access to the physical channel for all types of transfer. The figure below shows these blocks in a graphical representation. The upper layers shown in the figure below consist of a network layer, which provides network configuration, manipulation and message routing, and an application layer, which provides the intended function of the device. The definition of these upper layers is outside the scope of IEEE 802.15.4 standard. An IEEE 802.2 Type 1 logical link control (LLC) can access the MAC sublayer through the service specific convergence sublayer (SSCS). The LR-WPAN architecture can be implemented either as embedded devices or as devices requiring the support of an external device such as a PC. [4]

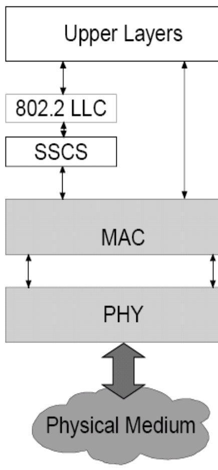


Figure 2.3: LR-WPAN Device Architecture [4]

The PHY provides two services: the PHY data service and the PHY management service interfacing to the physical layer management entity (PLME). The PHY data service enables the transmission and reception of PHY protocol data units (PPDUs) across the physical radio channel. The features of the PHY are activation and deactivation of the radio transceiver, ED, LQI, channel selection, clear channel assessment (CCA) and transmitting as well as receiving packets across the physical medium. [4]

According to [4], the radio shall operate at one of the following license-free frequency bands:

- 868–868.6 MHz (e.g. Europe)
- 902–928 MHz (e.g. North America)
- 2400–2483.5 MHz (worldwide)

The MAC sublayer provides two services: the MAC data service and the MAC management service interfacing to the MAC sublayer management entity (MLME) service access point (SAP) (known as MLME-SAP). The MAC data service enables the transmission and reception of MAC protocol data units (MPDUs) across the PHY data service. The features of the MAC sublayer are beacon management, channel access, GTS management, frame validation, acknowledged frame delivery, association and disassociation. In addition, the MAC sublayer provides methods for implementing application specific security mechanisms. [4]

The LR-WPAN uses two types of channel access mechanism depending on the network configuration. Nonbeacon-enabled networks use an "unslotted CSMA-CA" channel access mechanism. Beacon-enabled networks use a slotted CSMA-CA channel access mechanism. [4]

A brief overview of the concept of service primitives is provided next. The services of a layer are the capabilities it offers to the user in the next higher layer or sublayer by building its functions on the services of the next lower layer. This concept is illustrated in the figure below, showing the service hierarchy and the relationship of the two correspondent N-users and their associated N-layer (or sublayer) peer protocol entities. [4]

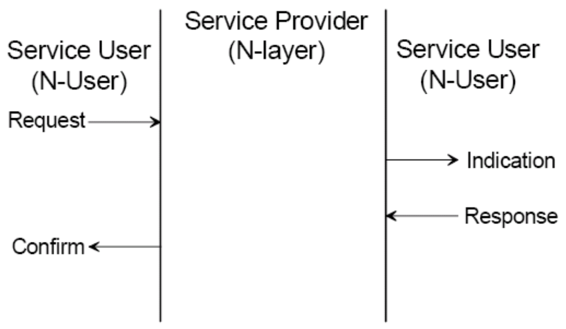


Figure 2.4: Service Primitives [4]

Services are specified by describing the service primitives and parameters that characterize it. A service may have one or more related primitives that constitute the activity that is related to that particular service. Each service primitive may have zero or more parameters that convey the information required to provide the service. According to [4], a primitive can be one of four generic types:

- Request: The request primitive is passed from the N-user to the N-layer to request that a service is initiated.
- Indication: The indication primitive is passed from the N-layer to the N-user to indicate an internal N-layer event that is significant for the N-user. This event may be logically related to a remote service request or it may be caused by an N-layer internal processing.
- Response: The response primitive is passed from the N-user to the N-layer to complete a procedure previously invoked by an indication primitive.
- Confirm: The confirm primitive is passed from the N-layer to the N-user to convey the results of one or more associated previous service requests. [4]

2.1.2 Network Layer Provided By ZigBee

The network (NWK) layer is required to provide functionality to ensure correct operation of the IEEE 802.15.4 MAC sublayer and to provide a suitable service interface to the application layer. In order to interface with the application layer, the network layer conceptually contains two service entities that provide the necessary functionality. These service entities are the "data service entity and the management service entity". The NWK layer data entity (NLDE) provides the data transmission service via its associated SAP, the NLDE-SAP. The NWK layer management entity (NLME) provides the management service via its associated SAP, the NLME-SAP. The NLME utilizes the NLDE to achieve some of its management tasks and it also maintains a database of managed objects known as the network information base (NIB). The NLDE shall provide a data service to allow an application to transport application protocol data units (APDU) between two or more devices. The devices themselves must be located on the same network. [1]

According to [1], the NLDE shall provide the following services:

- Generation of the Network level PDU (NPDU): The NLDE shall be able to generate an NPDU from an application support sub-layer PDU via the addition of an appropriate protocol header.
- Topology specific routing: The NLDE shall be able to transmit an NPDU to an appropriate device that is either the final destination of the communication or the next step toward the final destination in the communication chain.
- Security: The ability to ensure both the authenticity and confidentiality of a transmission.

The NLME shall provide a management service to allow an application to interact with the stack. According to [1], the NLME shall provide the following services:

- Configuring a new device: The ability to sufficiently configure the stack for operation as required. Configuration options include beginning an operation as a ZigBee coordinator or joining an existing network.
- Starting a network: The ability to establish a new network.
- Joining and leaving a network: The ability to join or leave a network as well as the ability for a ZigBee coordinator or ZigBee router to request that a device leave the network.
- Addressing: The ability of ZigBee coordinators and routers to assign addresses to devices joining the network.
- Neighbor discovery: The ability to discover, record and report information related to the one-hop neighbors of a device.
- Route discovery: The ability to discover and record paths through the network, whereby messages may be efficiently routed.
- Reception control: The ability for a device to control when the receiver is activated and for how long, enabling MAC sublayer synchronization or direct reception. [1]

The network layer supports multiple network topologies including star, cluster tree and mesh as shown in the below figure. [3]

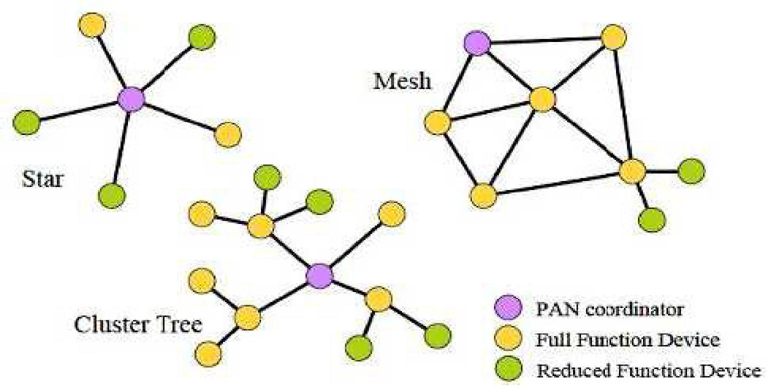


Figure 2.5: Network Topologies Supported by the NWK Layer [3]

In the star network topology the network consists of a ZigBee coordinator and ZigBee end devices. In mesh and tree network topologies the network consists of a ZigBee coordinator, possible ZigBee routers and ZigBee end devices. The difference between a tree and a mesh topology is that in a mesh topology, ZigBee devices are allowed to use peer-to-peer communication and mesh networks are self-repairing, whereas in a tree network, one-way routing is used and the tree networks are not self-repairing. In all these network topologies, the ZigBee routers and the ZigBee coordinator are so called full function devices (FFD) and the ZigBee end devices can be full or reduced function devices (RFD). In an RFD all the routing, coordinating functions are discarded. This way the stack of very simple devices (like light switches) can be greatly reduced and the energy usage and microcontroller requirements are reduced greatly. [3]

The formation of a new ZigBee network is started through a network layer primitive that is restricted to the ZigBee coordinator and also restricted to a ZigBee coordinator that is not currently joined to a network. At the beginning of network formation, the ZigBee coordinator performs an energy detection scan over a specified set of RF channels. When this channel scan is completed, the channels are ordered according to increasing RF energy and channels whose energy levels are seen too high are discarded. Next, the ZigBee coordinator performs an active scan on each of the selected RF channels to search for other ZigBee devices and networks. The ZigBee coordinator selects the best RF channel for a new network based on the results of the active scan. The ZigBee coordinator gives preference to any channel on which no existing networks were found when selecting the best RF channel. Then the coordinator selects the logical network identifier (a number) that will be applied to all devices that later join the network. Finally the coordinator starts to allow other devices to join the network. [2]

As part of the process of joining a network, each device is assigned a logical network address. In ZigBee, network addresses are assigned either by a network coordinator or by a ZigBee router using a tree-structured algorithm. [2]

The ZigBee network layer (NWK) is a crucial component that supports ZigBee systems like a skeleton. All ZigBee devices contain a network layer, but the network layer functions that a device performs depends upon its role as a ZigBee coordinator, router or end device. The ZigBee coordinator can perform all network functions while reduced functionality is available to the ZigBee routers and end devices. The network layer functions that can be performed by three types of ZigBee devices are shown in the below table. [3]

Table 2.1: Network Layer Functions Performed vs. ZigBee Device Types [3]

ZigBee Coordinator	**ZigBee Router**	**ZigBee End Device**	**ZigBee Network Layer Function**
X			Establish a new ZigBee network
X	X		Assign logical network addresses
X	X		Permit other devices to join/leave
X	X		Maintain lists of neighbors and routes
X	X		Route network-layer packets
X	X	X	Transfer network-layer packets
X	X	X	Join/leave a ZigBee network

When the coordinator permits new devices to join the network, these devices join the network through a process called “association”. In ZigBee, association is a very rapid process so that formation of a ZigBee network is designed to be considerably faster than formation of a Wi-Fi or Bluetooth network. Devices can also join a network "directly" through a previously designated parent device that is a ZigBee coordinator or router. Devices that lose contact with their parent device can rejoin a network through a process known as "orphaning". [2]

2.1.3 Application Layer Provided By ZigBee

The ZigBee application layer comprises the APS sublayer, the ZigBee Device Object (ZDO) containing the ZDO management plane, and the manufacturer-defined application objects [1].

According to [1], the responsibilities of the APS sub-layer include:

- Maintaining tables for binding, that is, the ability to match two devices together based on their services and their needs
- Forwarding messages between bound devices
- Group address definition, removal and filtering of group addressed messages
- Address mapping from 64 bit IEEE addresses to 16 bit NWK addresses and vice versa
- Fragmentation, reassembly and reliable data transport

According to [1], the responsibilities of the ZDO include:

- Defining the role of the device within the network (e.g. ZigBee coordinator or end device)
- Discovering devices on the network and determining which application services they provide
- Initiating and/or responding to binding requests
- Establishing a secure relationship between network devices

The application support sub-layer (APS) provides an interface between the network layer (NWK) and the application layer (APL) through a general set of services. These services are used by both the ZDO and the manufacturer-defined application objects. According to [1], the services of the APS sub-layer are provided by two entities:

- The APS data entity (APSDE) through the APSDE service access point (APSDE-SAP)

- The APS management entity (APSME) through the APSME service access point (APSME-SAP) [1]

The APSDE provides the data transmission service for the transport of application PDUs between two or more devices located on the same network including filtering of group addressed messages. The APSDE also provides fragmentation and reassembly of packets larger than the supported data payload of the APSDU and reliable data transport. The APSME provides security services, binding of devices, establishment and removal of group addresses. The APSME also maintains a database of managed objects, known as the APS information base (AIB). The AIB supports address mapping between 64 bit IEEE addresses and 16 bit NWK addresses. [1]

According to [2], the APSDE shall provide the following services:

- Generation of the Application level PDU (APDU): The APSDE shall take an application PDU and generate an APS PDU by adding the appropriate protocol overhead.
- Binding: This is the ability to match two devices together based on their services and their needs. Once two devices are bound, the APSDE shall be able to transfer a message received from one bound device to the second device.
- Group Address Filtering: This provides the ability to filter group addressed messages based on whether the device is a member of the group or not.
- Reliable transport: This increases the reliability of transactions above that available from the NWK layer by employing end-to-end retries.
- Duplicate rejection: Messages offered for transmission will not be received more than once.

The APSME shall provide a management service to allow an application to interact with the stack. The APSME shall provide the capability to match two devices together based on their services and their needs. This service is called the binding service and the APSME shall be able to construct and maintain a table to store binding information. According to [2], the APSME shall provide the following services:

- AIB Management: The ability to get and set attributes in the device's AIB.
- Security: The ability to set up authentic relationships with other devices through the use of secure keys.
- Group Management: This provides the ability to declare a single address shared by multiple devices, add devices to the group and remove devices from the group. [2]

The application framework in ZigBee is the environment in which application objects are hosted on ZigBee nodes. Inside the application framework, the application objects send and receive data through the APSDE-SAP. According to [1], the application objects perform the following functions through the ZDO public interfaces:

- Control and management of the protocol layers in the ZigBee device
- Initiation of standard network functions [1]

Up to 240 distinct application objects can be defined. Each application object interfaces on an endpoint indexed from 1 to 240. Two additional endpoints are defined for APSDE-SAP usage: endpoint 0 is reserved for the data interface to the ZDO and endpoint 255 is reserved for the data interface to broadcast data to all application objects. Endpoints 241-254 are reserved for future use. [1]

The key to communicating between devices (application objects) on a ZigBee network is agreement on a profile. An example of a profile would be home automation. This ZigBee profile allows a series of device types to exchange control messages to form a wireless home automation application. These devices are designed to exchange well known messages to take actions such as turning a lamp on or off, sending a light sensor measurement to a lighting controller or sending an alert message if an occupancy sensor detects movement. An example of another type of profile is the device profile that defines common actions between ZigBee devices. For example, wireless networks rely on the ability for autonomous devices to join a network and discover other devices and services on devices within the network. Device and service discovery are features supported within the device profile. [2]

A profile contains clusters which are identified by cluster identifiers. Clusters contain information about the data flowing out of or into a device. Cluster identifiers are unique within the scope of a particular profile. Binding decisions are taken by matching an output cluster identifier to an input cluster identifier assuming both are within the same profile. Clusters contain attributes. ZigBee messages sent between devices set or get the values of these attributes. When a ZigBee message is received by a device, the device takes the appropriate actions according to the attributes and set/get operations on these attributes contained by the ZigBee message. [1]

The ZDO represents a fundamental class of functionality that provides an interface between the application objects, the device profile and the APS. The ZDO is located between the application framework and the application support sublayer. It satisfies common requirements of all applications operating in a ZigBee protocol stack. [1]

According to [1], the ZDO is responsible for the following:

- Initializing the application support sub-layer (APS), the network layer (NWK), the Security Service Provider
- Assembling configuration information from the application objects to determine and implement discovery, security management, network management and binding management

The ZDO presents public interfaces to the application objects in the application framework layer for control of device and network functions by the application objects. The ZDO interfaces to the lower portions of the ZigBee protocol stack on endpoint 0 through the APSDE-SAP for data and through the APSME-SAP for control messages. [1]

2.1.4 Security Services Specification

Security services provided by ZigBee protocol include methods for key establishment, key transport, frame protection and device management. These services form the building blocks for implementing security policies within a ZigBee device. Where applicable, the ZigBee security architecture complements and makes use of the security services that are already present in the IEEE 802.15.4 security specification. The level of security provided by the ZigBee security architecture depends on the protection of the symmetric keys, and on the proper implementation of the cryptographic mechanisms and associated security policies involved. [1]

Trust in the security architecture depends on trust in the secure initialization and installation of keying material and on trust in the secure processing and storage of keying material. Some assumptions are made by the ZigBee protocol for the proper operation of ZigBee security services. In the case of indirect addressing, it is assumed that the binding manager is trusted. Implementations of security protocols, such as key establishment, are assumed to properly execute the complete protocol. Random number generators are assumed to operate as expected. [1]

Additionally, it is assumed that secret keys do not become available outside the device in an unsecured way. This means that a device will not intentionally or inadvertently transmit its keying material to other devices, unless the keying material is protected such as during key transport. An exception to this assumption occurs when a device that has not been preconfigured with a key joins the network.

In this case, a single key may be sent unprotected, thus resulting in a brief moment of vulnerability. [1]

The caution that must be considered in the above mentioned assumptions is that due to the low-cost nature of ad hoc network devices, one cannot generally assume the availability of tamper-resistant hardware. Hence, physical access to a device may result in access to secret keying material and other privileged information and access to the security software and hardware. [1]

ZigBee has to assume that different applications using the same radio are not logically separated due to cost constraints. In addition, from the perspective of a given device, it is not possible to verify whether cryptographic separation between different applications on another device or even between different layers of the communication stack is properly implemented. Therefore, one has to assume that separate applications using the same radio trust each other; that is, there is no cryptographic task separation between them. In addition, lower layers (for example, APS, NWK or MAC) are fully accessible by any of the applications. These assumptions lead to an "open trust model" for a device; different layers of the communication stack and all applications running on a single device trust each other. The "open trust model" on a device has some consequences. It allows reuse of the same keying material among different layers on the same device. It also allows end-to-end security to be performed on a device-to-device basis rather than between pairs of particular layers (or even pairs of applications) on two communicating devices. [1]

According to [1], these observations lead to the following architectural design choices:

- First, the principle that "the layer that originates a frame is responsible for initially securing it" is established. For example, if a MAC layer disassociate frame needs protection, MAC layer security shall be used.

Likewise, if a NWK command frame needs protection, NWK layer security shall be used.

- Second, if protection from theft of service is required (that is, malicious network devices), NWK layer security shall be used for all frames, except those passed between a router and a newly joined device (until the newly joined device receives the Network key). Thus, only a device that has joined the network and successfully received the Network key will be able to have its messages communicated more than one hop across the network.
- Third, due to the "open trust model", security can be based on the reuse of keys by each layer. For example, the active Network key shall be used to secure APS layer broadcast frames, NWK layer frames or MAC layer commands. Reuse of keys helps reduce storage costs.
- Fourth, end-to-end security is enabled to make it possible that only source and destination devices have access to their shared key. This ensures that routing of messages between devices can be performed independent of trust considerations.
- Fifth, to simplify interoperability between devices, the security level used by all devices in a given network and by all layers of a device shall be the same if security is used. If an application needs more security for its payload than is provided by a given network, it shall form its own separate network with a higher security level.

Security among a network of ZigBee devices is based on "link" keys and a "network" key. Unicast communication between APL peer entities is secured by means of a 128-bit link key shared by two devices, whereas broadcast communications are secured by means of a 128-bit network key shared among all devices in the network. The intended recipient is always aware of the exact security arrangement; that is, the recipient knows whether a frame is protected with a link or a network key. [1]

A device shall acquire link keys either via key transport, key establishment or pre-installation (for example, during factory installation). A device shall acquire a network key via key transport or pre-installation. The key establishment technique for acquiring a link key is based on a “master” key. A device shall acquire a master key (for purposes of establishing corresponding link keys) via key transport or pre-installation. Ultimately, security between devices depends on the secure initialization and installation of these keys. [1]

The ZigBee security architecture includes security mechanisms at three layers of the protocol stack. The MAC, NWK and APS layers are responsible for the secure transport of their respective frames. Additionally, the APS sublayer provides services for the establishment and maintenance of security relationships. The ZigBee Device Object (ZDO) manages the security policies and the security configuration of a device. [1]

In the below discussion, CTR stands for counter mode, CBC-MAC stands for cipher block chaining message authentication code, CCM stands for counter with CBC-MAC mode of operation and CCM* stands for enhanced counter with CBC-MAC mode of operation. Specifically, at least one of ZigBee’s security needs is the ability to protect incoming and outgoing frames using the security levels based on CCM*. CCM* is a generic combined encryption and authentication block cipher mode. CCM* is only defined for use with block ciphers with a 128-bit block size, such as AES-128. The CCM* ideas can easily be extended to other block sizes, but this will require further definitions. CCM* uses the block cipher in cipher block chaining mode to calculate the MAC and in counter mode to encrypt the frames. CCM* is a minor modification of CCM specified in the IEEE 802.15.4 specification. CCM* includes all of the features of CCM and additionally offers encryption-only and integrity-only capabilities. [1]

For security purposes, ZigBee defines the role of trust center. The trust center is the device trusted by devices within a network to distribute keys for the purpose of network and end-to-end application configuration management. All members of the network shall recognize exactly one trust center and there shall be exactly one trust center in each secure network. In high-security, commercial applications a device can be pre-loaded with the trust center address and initial master key (for example, via an unspecified mechanism). Alternatively, if the application can tolerate a moment of vulnerability, the master key can be sent via an in-band unsecured key transport. If not pre-loaded, a device’s trust center defaults to the ZigBee coordinator or a device designated by the ZigBee coordinator. In low-security, residential applications a device securely communicates with its trust center using the network key, which can be preconfigured or sent via an in-band unsecured key transport. [1]

2.2 ZigBee Applications

This thesis work includes forming a ZigBee wireless network and controlling this network through the Internet via a PC. This section of the thesis mentions about example applications in the literature using ZigBee protocol.

ZigBee enables reliable, cost-effective and low-power wireless monitoring and control products based on open standards. The primary goals of ZigBee and the ZigBee Alliance are to address the market need for standards-based, interoperable wireless products in areas such as industrial process monitoring, home and building automation and control, consumer electronics, medical monitoring and similar areas. ZigBee is designed to support these goals through simplicity, long battery life, mesh networking capabilities, security, reliability and low implementation cost. The ZigBee Alliance provides standards definition, interoperability testing, certification testing and branding for ZigBee devices to ensure they meet all applicable standards and interoperate as intended. [11]

The figure below shows a next-generation IEEE 802.15.4 silicon platform, this one from Freescale Semiconductor.

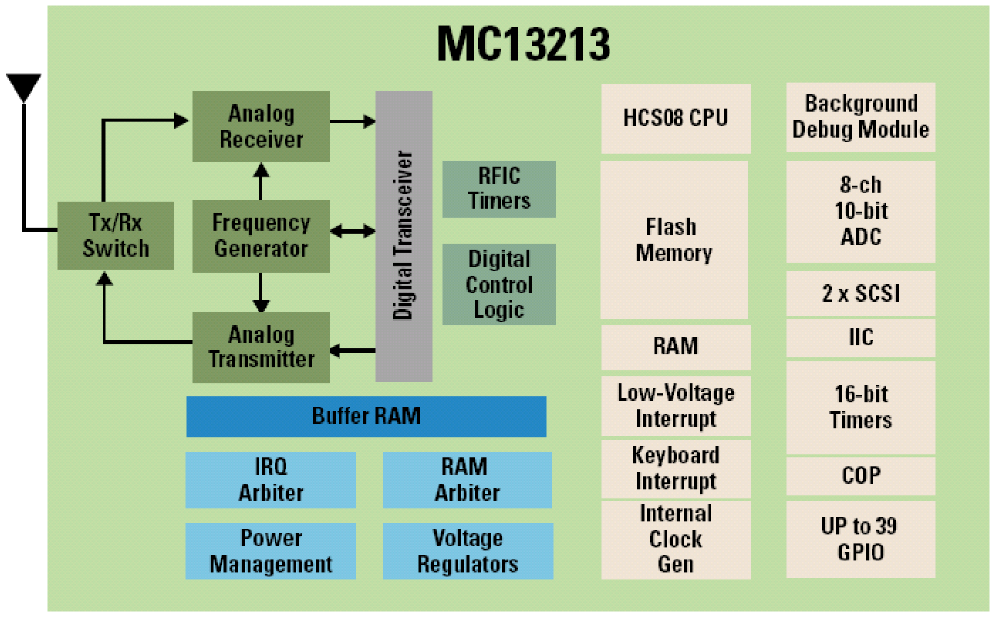


Figure 2.6: MC13213 – An Example IEEE 802.15.4 Silicon Platform [12]

A single-package system, the MC13213 contains all the required functionality to take full advantage of ZigBee and IEEE 802.15.4 technology. On the left side of the block diagram in the above figure are the 2.4 GHz radio circuits, while on the right side is the 8-bit MCU as well as various peripheral devices that are available for application development. Within the transceiver are all the necessary mechanisms to manage the RF channel as well as the protocol's management of packets. This means that the MCU does not have to deal with RF channel timing issues at the symbol or bit level and is free to sleep or do other work while messages are being received or transmitted. In addition, the transceiver contains timers and interrupt inputs that can further relieve the MCU from the burden of dealing with other protocol specifics. [12]

The MCU contains functions that a sensor or control device needs, including ADCs, timers and I/O bits, and performs the high-level management of the RF data modem as well as hosting the sensor/control application. Because of sophisticated power management features built into both the MCU and the transceiver, there are a number of power-down states available that allow the application developer to trade functionality, latency and average power consumption. [12]

ZigBee deployment traditionally includes control and automation of homes, buildings, factories, and entertainment systems. Most of these types of systems require only a simple microcontroller. Moving to a higher-performance embedded processor opens doors to new kinds of ZigBee applications, which fall into the three classes explained in the following three paragraphs. [11]

Class 1 of new kinds of ZigBee applications is decision-maker applications. In the role of decision-maker, an embedded processor distills great amounts of data to find the essential information that is relevant to a given application domain. The resulting small amount of information can then be sent to other nodes in a ZigBee network. Security systems serve as good examples for this type of configuration. The use of biometric information has become an effective way to guard against access to sensitive resources. However, data from a fingerprint sensor or a retina scanner needs to be processed against a database of verified users. To ensure an acceptable user experience, the time to process the raw biometric information must not exceed a predetermined delay. Many manufacturers of biometric products aim for 500 msec as the upper time limit between the biometric scan and the match/no-match decision. In a real fingerprint-access system, a high-performance processor, such as one from the Blackfin family, can make a decision in less than 200 msec when running at 400 MHz. This provides time for the system to perform multiple scan and decision-making passes, thus increasing the overall accuracy. Because the decision output from the processor contains a small amount of information (such as "passed fingerprint authentication" or "failed fingerprint authentication"), the bandwidth requirement on the network is low. The main aspect of the network in security systems like fingerprint access is the latency. A rule of thumb is that every hop that a message goes through adds between 10 and 100 msec of latency, depending on the load of the network. This latency, along with the time needed to verify a fingerprint, must not exceed the acceptable user delay defined by the end manufacturer. An embedded processor, coupled with the flexibility of a ZigBee wireless network, allows for a cost-effective security system solution. [11]

Class 2 of new kinds of ZigBee applications is media compressor/decompressor applications. While ZigBee is not a high-data-rate protocol, it is possible to judiciously transfer audio and video media across a ZigBee infrastructure. Using one processor to compress media at the source ZigBee node and another processor to decompress the media at the destination ZigBee node can achieve this. One example application of this type is a video doorbell. The setup of such a doorbell system is actually quite simple with a ZigBee network. A video camera at the front door, connected to an efficient media processor and a ZigBee transceiver, serves as the broadcast point. Multiple video displays utilizing media processors connected to ZigBee transceivers, can be placed throughout the building to pick up images or low-bit-rate video of guests. Because of the real-time aspect of the data, the media processor must perform the encode and decode algorithms within acceptable user-defined time constraints. A prototype video-doorbell system has been implemented using an Analog Devices ADSP-BF533 embedded media processor and an Ember EM260 ZigBee coprocessor. A JPEG encoder/decoder implemented on the 32-bit Blackfin ADSP-BF533 media processor consumes ~50 cycles/pixel for a 12:1 compression ratio. For one frame with 640×480 pixels of resolution, this is 15.4 million cycles. While standard low-bit-rate video encoding is an easy task for today's media processors, it poses challenges to low-bit-rate ZigBee networks. The most bandwidth-efficient media codecs are the most computationally intensive. The nominal bit rate of the underlying IEEE 802.15.4 standard is 250 kbps. Because of the CSMA/CA architecture, the realistic maximum bit rate is on the order of half the nominal bit rate. In practice, one can expect to achieve upwards of

60 kbps with large packet sizes in a configuration without security. With network bandwidth of this order, a good-quality VGA-sized JPEG image can be transmitted every second. One way to increase the frame rate when transmitting images over a low-bit-rate link like ZigBee is to implement a more advanced codec that yields a higher compression ratio than JPEG. Advanced media codec algorithms running on modern processors enable the possibility of media transmission over ZigBee networks. [11]

Class 3 of new kinds of ZigBee applications is Gateway/administrator applications. The Gateway/administrator class of ZigBee nodes is used to bridge a large ZigBee network with a more prolific technology like Ethernet, Wi-Fi or USB. An embedded processor with higher levels of peripheral integration can decrease the bill of materials. An example application is a central administrator ZigBee node that collects data from hundreds of nodes in an industrial environment. The data may need to be processed or filtered before being presented to a PC user through any standard interface like Wi-Fi. The presentation of data can most easily be done with a web server running on the administrator node. Depending on the number of slave ZigBee nodes the administrator serves, the embedded processor can be taxed in terms of data bandwidth. There is also a possibility that the administrator needs to store a lot of data locally. For this reason, it is advisable to choose a processor with an on-chip flash, a NAND-flash port or a hard-drive controller. [11]

In [13], the deployment of ZigBee in existing industrial automation networks is discussed. ZigBee technology has industrial networks as one of its primary targets. Implementation of ZigBee-enabled devices is attractive, but in many cases their coexistence with existing networks is required. In [13], the ways to incorporate ZigBee-enabled devices into existing network topologies effectively, gaining the benefits of ZigBee technology while preserving existing investments, are examined. [13]

CHAPTER 3

PROPOSED SYSTEM ARCHITECTURE

3.1 General System Architecture

The aim of this thesis is to construct a ZigBee wireless network and control this network with a PC through the Internet. At least one of the nodes of this wireless network has the responsibility of controlling other nodes of the network and this node is called the ZigBee master controller. The other nodes of the wireless network are called ZigBee slave devices. ZigBee slave devices cannot communicate with each other directly.

The general architecture of the system that is formed in this thesis is shown in the below figure. Two ZigBee slave devices are implemented in this thesis work. These have the role of light and occupancy sensor whose services and attributes are defined in the Home Control, Lighting Profile supplied by the ZigBee Alliance. The ZigBee slave devices implemented in this thesis work are reduced function devices (RFDs). They can sleep when their receiver is idle. The ZigBee master controller is a full function device (FFD) and is the coordinator of the constructed wireless network. It is mains-powered and does not sleep when its receiver is idle. It controls the ZigBee slave devices by sending ZigBee commands to them.

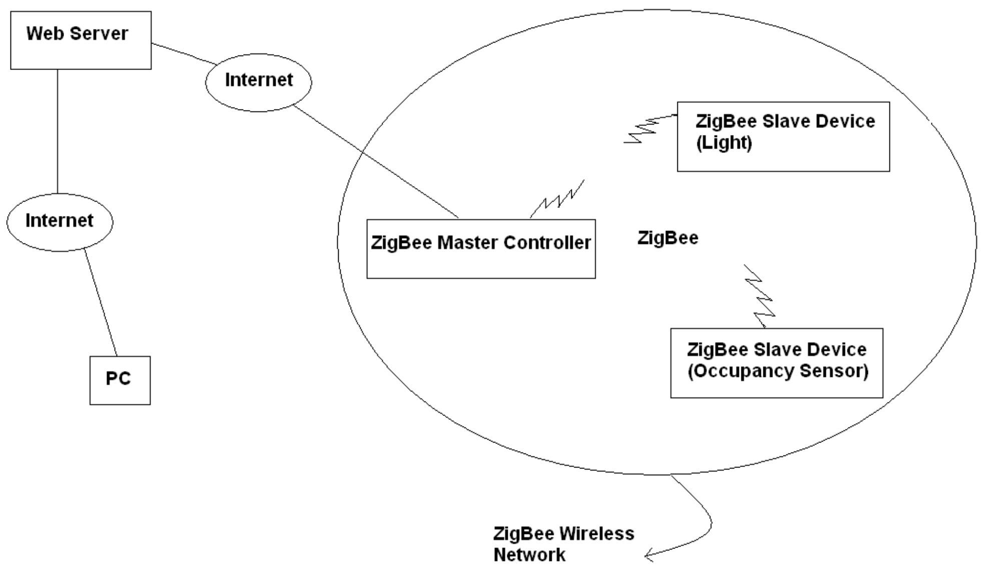


Figure 3.1: The general system architecture

A PC connected to the Internet can control the constructed wireless network by sending commands to the ZigBee master controller via a web application implemented in this thesis work and loaded into a suitable web server. The PC can send commands to any of the ZigBee slave devices indirectly through the ZigBee master controller. The PC can also send commands to the ZigBee master controller. The PC can not directly talk to the ZigBee master controller. The communication between the PC and the ZigBee master controller is performed via the web server in which the web application mentioned above is loaded. The PC can download the web application from the web server into its browser. The web server directly talks to the ZigBee master controller. The communication between the web server and the ZigBee master controller is performed using a specific application protocol working over TCP/IP through the Internet.

For the communication between the PC and the ZigBee master controller to be realized, both the PC and the ZigBee master controller must connect to the web server in which the web application mentioned above is loaded. Since the web server can be anywhere on the Internet, both the PC and the ZigBee master controller must be connected to the Internet to connect to the web server. The connection of the PC to the Internet may be wired or wireless. The ZigBee master controller connects to the Internet by connecting to an Ethernet connected to the Internet.

For the parts of the whole system where Internet is used as the communication infrastructure, security is mainly based on SSL protocol. The communication between the end user's PC and the web server is protected by using SSL as the security infrastructure. Some other higher level security measures working over SSL are also taken to further authenticate the end user and authorize access to the web application stored in the web server. The communication between the web server and the ZigBee master controller is also secured by SSL and higher level authorization mechanism working over SSL.

For inside the ZigBee wireless network part of the system, security is provided by using security services supplied by ZigBee. The ZigBee master controller and slave device software is properly configured to use ZigBee security services with the ZigBee master controller acting as the trust center of the ZigBee wireless network.

The general operation of the entire system is depicted in the below figure. A PC connected to the Internet sends a command related with any ZigBee device (the ZigBee master controller or any ZigBee slave device) to the ZigBee master controller via a web application implemented in the thesis. The communication between the web server in which the web application is loaded, and the ZigBee master controller is performed using the specific application protocol (implemented in the thesis) working over TCP/IP.

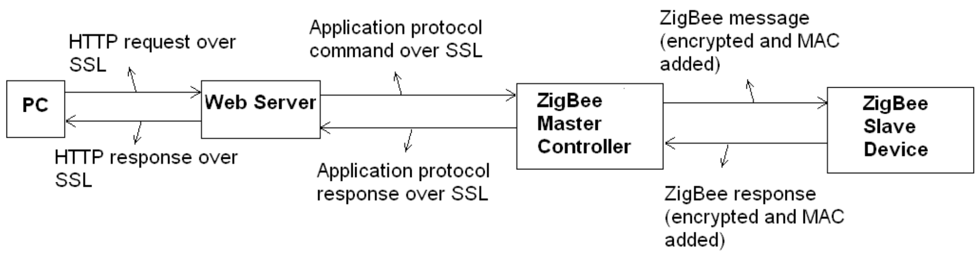

HTTP request over SSL
Application protocol command over SSL
ZigBee message (encrypted and MAC added)
PC
Web Server
ZigBee Master Controller
ZigBee Slave Device
HTTP response over SSL
Application protocol response over SSL
ZigBee response (encrypted and MAC added)

Figure 3.2: The general operation of the system

If the command (received from the PC) is related with the ZigBee master controller, the ZigBee master controller prepares a response and sends the response to the PC via the web server in which the web application is loaded. If the command is related with any ZigBee slave device, the ZigBee master controller puts the command into a form acceptable by the ZigBee slave device and sends the command to the ZigBee slave device. The ZigBee slave device takes the appropriate actions given by the command, prepares a response and sends the response to the ZigBee master controller. The ZigBee master controller puts the response into a form acceptable by the web server and sends the response to the web server. After that, the web server sends the response to the PC in a user-friendly form via the web application.

3.2 ZigBee Master Controller Hardware Architecture

The hardware architecture of the ZigBee master controller is mainly comprised of three elements which are the Ethernet Card, ZigBee RF (Radio Frequency) Card and MCU (Microcontroller Unit) block. Actually, there are two MCUs used in the ZigBee master controller. One of these MCUs is a 16-bit microcontroller and the other is a 8-bit microcontroller. The hardware architecture of the ZigBee master controller is shown in the below figure.

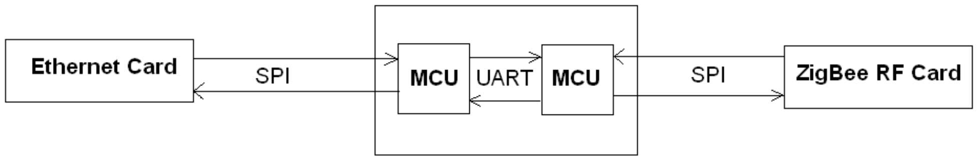


Figure 3.3: The hardware architecture of the ZigBee master controller

The ZigBee master controller uses one of the MCUs (the first MCU) and the Ethernet Card to connect to an Ethernet connected to the Internet and to receive and process commands received from a PC connected to the Internet. The Ethernet Card receives the commands coming from the PC and relays these commands to the first MCU. The first MCU processes the received commands and relays these commands to the second MCU. The communication between the Ethernet Card and the first MCU is performed using SPI (Serial Peripheral Interface) mechanism. Detailed information about SPI can be found at [5, 6, 7]. The communication between the two MCUs is performed using UART (Universal Asynchronous Receiver Transmitter) mechanism.

The ZigBee master controller uses the second MCU and the ZigBee RF Card to send/receive commands to/from the ZigBee slave devices and process these commands. The second MCU receives a command from the first MCU, converts the command to a form acceptable by the ZigBee slave device the command is related with. The second MCU then sends the command to the ZigBee slave device using the ZigBee RF Card. After a while, the ZigBee RF Card receives a response to the command from the ZigBee slave device and relays the response to the second MCU. The second MCU converts the response to a form acceptable by the first MCU (converts it to a form conforming to the specific application protocol) and sends the response to the first MCU via UART mechanism. After receiving the response, the first MCU processes it and sends it to the web server.

The communication between the second MCU and the ZigBee RF Card is performed using SPI mechanism.

3.3 ZigBee Master Controller Software Architecture

The software architecture of the ZigBee master controller can be divided into two parts. The first part contains software needed by the ZigBee master controller to connect to and communicate across an Ethernet connected to the Internet. This software receives commands from the web server and processes them. It interacts with the Ethernet card and the MCU, controlling the Ethernet Card, in the ZigBee master controller. The second part contains software needed by the ZigBee master controller to form and communicate across a ZigBee network containing ZigBee slave devices. This software interacts with the ZigBee RF Card and the MCU, controlling the ZigBee RF Card, in the ZigBee master controller. The below figure depicts the software components found in the ZigBee master controller.

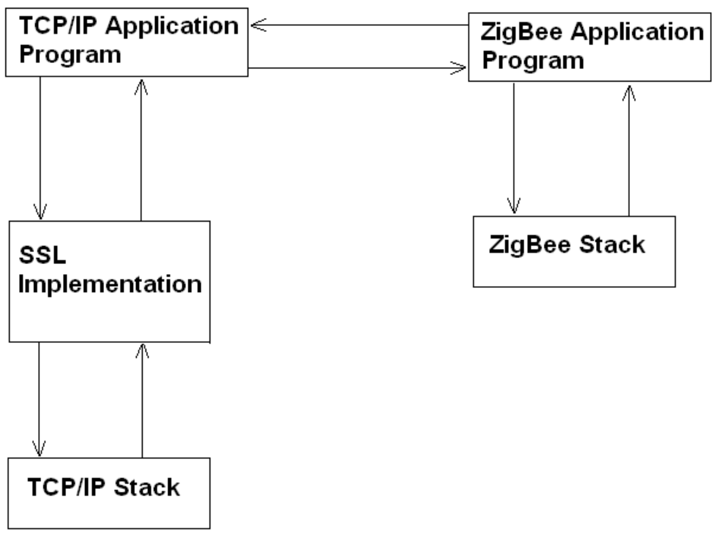


Figure 3.4: The software architecture of the ZigBee master controller

The ZigBee master controller needs to have a TCP/IP stack to communicate with a PC, connected to the Internet, through the Internet (via the web server); because the communication between the web server and the ZigBee master controller is performed using the specific application protocol (defined in this thesis work) working over TCP/IP. This stack is needed to provide the network software constructs defined in the TCP/IP specification to the ZigBee master controller. This stack must handle the execution and management of the TCP/IP state machine relieving the application programmer from the burden of managing the TCP/IP state machine. This stack must also be compliant with the Ethernet Card found on the ZigBee master controller. There is such a TCP/IP stack loaded on the program memory of the MCU, controlling the Ethernet Card, on the ZigBee master controller.

The ZigBee master controller needs to have some security mechanism used to protect the communication between itself and the web server. This security mechanism is chosen to be SSL so there must be an SSL implementation working over TCP/IP stack in the ZigBee master controller. It is essential that this SSL implementation is compliant with the TCP/IP stack loaded on the ZigBee master controller. There is such an SSL implementation loaded on the program memory of the MCU, controlling the Ethernet Card, on the ZigBee master controller. This SSL implementation protects the TCP/IP stack services with the SSLv3 protocol.

There is an application program working over the TCP/IP stack services secured by the SSL implementation. It is loaded on the program memory of the MCU, controlling the Ethernet Card, on the ZigBee master controller. This application program handles the message exchange between the web server and the MCU, controlling the Ethernet Card, on the ZigBee master controller. This application program also handles the message exchange between the two MCUs on the ZigBee master controller. This application program is called as TCP/IP application program throughout the thesis.

The TCP/IP application program first opens a TCP server socket to a specific port. Then, it adds an SSL listener over this TCP server socket to listen for incoming SSL client connections. A PC connected to the Internet can connect to the web server and execute the code of the web application residing in the web server. The web server can then connect to the TCP server socket opened by the ZigBee master controller, secured by SSL and listening for client connections. The web server also uses SSL (implemented by the web application) to connect to the TCP server socket of the ZigBee master controller. The SSL implementation on the ZigBee master controller is configured to authenticate the client and the SSL implementation on the web application is configured to authenticate the server.

After opening and securing the TCP server socket, the TCP/IP application program continually checks the TCP server socket for incoming SSL client connections. When the web server connects to the TCP server socket opened by the ZigBee master controller, the TCP/IP application program on the ZigBee master controller performs a user-based authorization based on the authenticated client certificate received from the web server. This client certificate is specific to the web application executing on the web server. If the authorization does not succeed, the TCP/IP application program cancels the connection to the web server. If the authorization succeeds, the TCP/IP application program starts to check for incoming data into the TCP server socket.

The TCP/IP application program reads a specified amount of data bytes for each command. It does not read the data for a command until it sends a response to the previous command to the web server. After reading a command, the TCP/IP application program processes the command and relays the command to the MCU, controlling the ZigBee RF Card, on the ZigBee master controller via UART mechanism byte by byte. Then, the TCP/IP application program waits for a response to the command. After receiving the response to the command, the TCP/IP application program processes the response and then relays the response to the web server by writing the response to the TCP server socket byte by byte.

After sending the response to the web server, the TCP/IP application program starts to check for incoming data into the TCP server socket for the next command.

The TCP/IP application program uses the TCP/IP services secured by the SSL implementation to create and open a TCP server socket, listen for client connections on the TCP server socket, read data from the TCP server socket and write data to the TCP server socket. It also uses the UART module embedded in the MCU, controlling the Ethernet Card, to exchange UART messages with the other MCU, controlling the ZigBee RF Card, on the ZigBee master controller. It implements an interrupt handler mechanism to handle UART receive interrupts occurring when a byte is received via UART by the MCU, controlling the Ethernet Card (the mechanism is described in detail in Chapter 4 Implementation Details).

The ZigBee master controller needs to have a ZigBee stack implementing the ZigBee state machine to communicate with the ZigBee slave devices. This stack needs to be capable of receiving, sending and securing ZigBee specific messages, and storing network parameters for the device it is loaded in. This stack must also be compliant with the ZigBee RF Card used on the ZigBee master controller. There is such a ZigBee stack loaded on the program memory of the MCU, controlling the ZigBee RF Card, on the ZigBee master controller.

There is an application program, implemented on the ZigBee master controller, used to handle the communication with the ZigBee slave devices using the ZigBee protocol. This application program is called as ZigBee application program throughout the thesis. The ZigBee application program is loaded on the program memory of the MCU, controlling the ZigBee RF Card, on the ZigBee master controller and works on the ZigBee stack loaded on the same place. This application program serves as a bridge between the MCU, controlling the Ethernet Card, of the ZigBee master controller and the ZigBee slave devices. It handles the message exchange between the MCU, controlling the Ethernet Card, on the ZigBee master controller and the ZigBee slave devices.

The first thing the ZigBee application program does is to scan the allocated channels determined by the application developer, choose the best channel and then form a ZigBee wireless network on that channel. It then becomes the coordinator of the network and allows other devices to join the network. It stores the type and identifier of each ZigBee slave device joined to the network.

The ZigBee application program is interrupted when it receives a command from the MCU, controlling the Ethernet Card, on the ZigBee master controller via UART mechanism. The interrupt occurred in this case is the UART receive interrupt. When the ZigBee application program receives the command, it stores the command in a buffer. Then, it converts the command to a ZigBee message to be sent to the ZigBee slave device related with the command if there is such a ZigBee slave device. If there is not such a ZigBee slave device, it sends back an error message to the MCU, controlling the Ethernet Card, via UART mechanism.

If the ZigBee slave device related with the command exists, the ZigBee application program sends the ZigBee message (obtained from the command) to the ZigBee slave device and waits for a ZigBee response message. The ZigBee stack handles the transmission, reception and securing of ZigBee messages for the Zigbee application program. When the ZigBee application program receives the ZigBee response message from the ZigBee slave device; it processes the ZigBee response message, converts the ZigBee response message to a form compliant with the specific application protocol and sends the converted response message to the MCU, controlling the Ethernet Card, on the ZigBee master controller via UART mechanism.

Not all the commands sent to the ZigBee application program are targeted to the ZigBee slave devices. Some commands are intended to be processed by the ZigBee application program. In those cases, the ZigBee application program does not convert the received command to a ZigBee message since the command is not needed to be sent to any ZigBee slave device. In such cases, the ZigBee application program processes the received command, prepares a response formatted according to the specific application protocol and sends the response to the MCU, controlling the Ethernet Card, via UART mechanism.

The ZigBee application program provides a mapping between the specific application protocol (defined in this thesis) and the ZigBee protocol. If the command coming from the MCU, controlling the Ethernet Card, is targeted to a ZigBee slave device, it must be converted to a ZigBee message before it is sent to the ZigBee slave device. This conversion process is performed by the ZigBee application program loaded on the MCU, controlling the ZigBee RF Card, on the ZigBee master controller. To perform this conversion process, the ZigBee application program uses its information about the attributes of the specific devices defined in the Home Control, Lighting Profile, and its information about the format of the fields of the specific protocol messages defined by the specific application protocol. The ZigBee application program first discovers the ZigBee slave device, the received command is related with, by using the device type and identifier information given in the command compliant with the specific application protocol. Then, it finds the attributes, of the ZigBee slave device, whose values are intended to be set or got by the command. It also finds the values to which some of these attributes are intended to be set by the command. The ZigBee application program then forms a ZigBee message from the command by using these extracted information.

The ZigBee application program performs the opposite of the conversion process mentioned above when it receives a ZigBee response message from a ZigBee slave device in response to a ZigBee message sent to the ZigBee slave device. In this case, the ZigBee application program converts the received ZigBee response message to a response compliant with the specific application protocol. A ZigBee response message contains the results of attribute set operations or the values of specific attributes of a ZigBee slave device. The ZigBee application program encodes the values of attributes or the results of attribute set operations related with the ZigBee slave device to the fields of the response message compliant with the specific application protocol.

The ZigBee application program determines a timeout value for a ZigBee message and starts a timer when it sends the ZigBee message. If the timer reaches that timeout value before the ZigBee application program receives a ZigBee response message, the ZigBee application program sends a response, indicating a timeout condition and conforming to the specific application protocol, to the MCU, controlling the Ethernet Card, via UART mechanism.

As mentioned above, the ZigBee application program stores a list of records containing the device types and identifiers of ZigBee slave devices joined to the network coordinated by the ZigBee master controller. The ZigBee application program fetches device type and identifier information about each ZigBee slave device by sending active endpoint and simple descriptor requests to each ZigBee slave node. A ZigBee slave node can contain up to 240 endpoints each of which can accommodate a different ZigBee slave device. The ZigBee application program receives a list of all active endpoints in a ZigBee slave node by sending an active endpoint request to the ZigBee slave node. After receiving the list of all active endpoints, the ZigBee application program sends a simple descriptor request to the ZigBee slave node for each active endpoint of the ZigBee slave node to get a simple descriptor of the ZigBee slave device residing on each active endpoint. A simple descriptor contains information about the device type and attributes of the ZigBee slave device residing on a specific endpoint on a ZigBee slave node.

After getting the simple descriptor for each ZigBee slave device residing on each ZigBee slave node, the ZigBee application program produces a device identifier for each ZigBee slave device. Then, the ZigBee application program stores a record, containing the device type and identifier of a ZigBee slave device, for each ZigBee slave device.

The ZigBee application program also handles the leaving of a ZigBee slave device from the network coordinated by the ZigBee master controller. When a ZigBee slave device leaves the network, the ZigBee application program deletes its record containing its device type and device identifier.

The ZigBee application program uses the services provided by the ZigBee stack to be notified of the received ZigBee primitives, send ZigBee messages, form a network, send active endpoint and simple descriptor requests etc..

3.4 ZigBee Slave Device Hardware Architecture

There are two ZigBee slave devices implemented in this thesis work. These two ZigBee slave devices reside on two different ZigBee slave nodes. One of the ZigBee slave devices implements the logic of light defined by the Home Control, Lighting Profile. The other ZigBee slave device implements the logic of occupancy sensor also defined by the Home Control, Lighting Profile. The hardware and software architectures of these two ZigBee slave devices are similar. The hardware architecture of a ZigBee slave device is shown in the below figure.

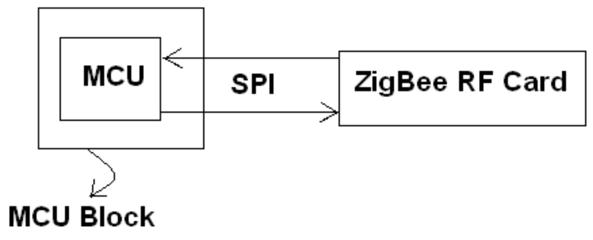


Figure 3.5: The hardware architecture of a ZigBee slave device

The hardware architecture of a ZigBee slave device is comprised mainly of two elements which are the ZigBee RF Card and MCU block. The MCU block for a ZigBee slave device contains one MCU which is a 8-bit microcontroller. The MCU uses the ZigBee RF Card to communicate with the ZigBee master controller. The ZigBee RF Card receives a ZigBee message from the ZigBee master controller and relays the ZigBee message to the MCU. The MCU processes the received ZigBee message, takes the appropriate actions according to the commands encapsulated in the received ZigBee message, prepares a ZigBee response message and sends the ZigBee response message to the ZigBee master controller via the ZigBee RF Card. The communication between the MCU and the ZigBee RF Card is performed using SPI mechanism.

3.5 ZigBee Slave Device Software Architecture

A ZigBee slave device needs to have a ZigBee stack to communicate with the ZigBee master controller using the ZigBee wireless protocol. This ZigBee stack must be capable of performing ZigBee network discovery, making join attempts to any of the discovered networks, sending, receiving and securing ZigBee messages etc.. This stack must execute the ZigBee state machine and be compliant with the ZigBee RF Card on the ZigBee slave device. There is such a ZigBee stack loaded on the program memory of the MCU of a ZigBee slave device.

There is also an application program operating on the ZigBee stack in a ZigBee slave device. This application program is called as slave application program throughout the thesis. This application program is loaded on the program memory of the MCU of a ZigBee slave device. This application program receives ZigBee messages from the ZigBee master controller, processes these messages, takes the appropriate actions according to these messages, prepares and sends ZigBee response messages to the ZigBee master controller. The application program uses the services provided by the ZigBee stack to perform all network-related operations. The overall software architecture of a ZigBee slave device is shown in the below figure.

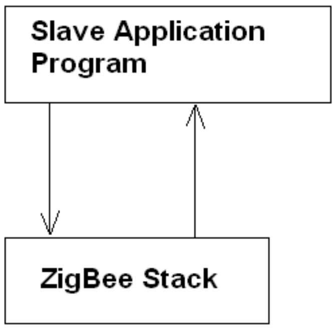


Figure 3.6: The software architecture of a ZigBee slave device

The slave application program initiates a network discovery process on the allowed channels determined by the application developer to get information about existing networks on the selected channels. It stores the received information about the existing networks in a list. Then, the slave application program tries to join to the first network it finds. If it is not allowed to join to the first network it finds, the slave application program tries to join to the next network it finds and it goes on like this.

The slave application program receives ZigBee messages from the ZigBee master controller. A ZigBee message can contain one or more transactions. Each transaction is related with one attribute of a ZigBee slave device. Each transaction intends to set or get the value of the attribute it is related with. When receiving a ZigBee message, the slave application program examines the transactions of the received ZigBee message one by one. If any of the transactions requests a response, the slave application program creates a ZigBee response message and fills this response message with response transactions. A response transaction contains a response to a transaction for the received ZigBee message. For each transaction requesting acknowledgement or the value of an attribute, the slave application program prepares a response transaction containing the requested acknowledgement or value of an attribute and embeds the prepared response transaction in a ZigBee response message.

The slave application program performs the operation on an attribute requested by each transaction in the received ZigBee message. Then, if a ZigBee response message is prepared by the slave application program, the slave application program sends the ZigBee response message to the ZigBee master controller. When preparing a ZigBee response message (if needed), the slave application program stores the contents of the prepared ZigBee response message in a buffer. When an error occurs in the transmission of a ZigBee response message to the ZigBee master controller, the slave application program prepares the same ZigBee response message using the stored information in the buffer and transmits the same ZigBee response message to the ZigBee master controller.

The slave application program causes the ZigBee slave device it is loaded in to sleep when the following conditions are true:

- There is no data to be received from the parent ZigBee device (ZigBee master controller) of the ZigBee slave device
- There is no ZigBee primitive to be executed by the ZigBee stack loaded on the ZigBee slave device
- There is no background processes executed by the ZigBee stack loaded on the ZigBee slave device
- There is no application-specific processes to be executed by the slave application program

The slave application program deactivates the transceiver and activates a specific interrupt before going to sleep. An event causing this specific interrupt to occur causes the slave application program to wake up. When the slave application program wakes up, it activates its transceiver and requests data, incoming to it, from the parent device of the ZigBee slave device it is loaded in. When a ZigBee slave device is sleeping, its parent stores the messages incoming to the ZigBee slave device and the slave application program requests these messages just after the ZigBee slave device wakes up.

3.6 Web Application Architecture

As mentioned above, a PC connected to the Internet can connect to the ZigBee master controller via a specific web application. The real connection is constructed between the web server in which the specific web application is loaded and the ZigBee master controller. This web application contains two web pages. One of these web pages is the login page. The main application logic resides in the other web page. This other web page is called as the default web page throughout the thesis.

The architecture of the login page mainly consists of two parts. The first part is the user interface part of the login page. The second part is the event-handling part of the login page. The architecture of the login page is depicted in the below figure.

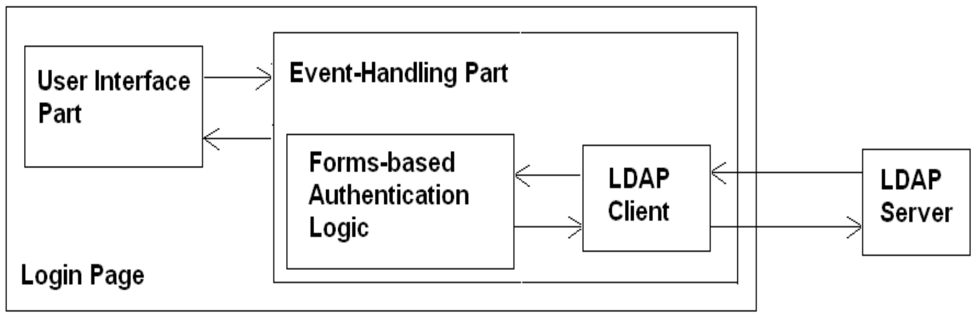


Figure 3.7: The architecture of the login page of the web application

The user interface part of the login page contains the necessary user interface web controls used to collect login name, password and other login-related information from the end user. This part also displays warning to the end user if the login to the web application is unsuccessful. The collected user credentials are sent to the event-handling part of the login page.

The event-handling part contains the event handling routines handling the events caused by activating the user interface elements in the user interface part, and the web page’s lifetime events. The event-handling part contains forms-based authentication logic used to verify the user credentials against a user store and form a specifically structured data to be used to identify the end user in the future attempts of the end user to access to the web application. The user store in question is implemented by an LDAP server. The event-handling part of the login page contains LDAP client software used by the forms-based authentication logic to connect to the LDAP server.

After collecting the end user’s credentials, the forms-based authentication logic tries to connect to the LDAP server (by means of the LDAP client software) using TLS. Then, it performs bind operation on the LDAP server by sending end user’s credentials to the LDAP server for authentication of the end user to the LDAP server. The LDAP server tries to verify the end user’s credentials against the LDAP directory, it contains, to authenticate the end user. If the authentication of the end user to the LDAP server is not completed successfully, the end user is not allowed to log in to the web application. Otherwise, the forms-based authentication logic forms an encrypted ticket containing the end user’s identity and sends this ticket to the end user. By using this ticket, the end user can automatically connect to the web application in his/her future attempts.

Each leaf LDAP entry in the LDAP directory maintained by the LDAP server represents a specific end user authorized to connect to the web application. Each such entry should contain a field (attribute) storing the login name of the end user and a field storing the password (in an encrypted form) of the end user represented by the entry. The entry may contain other fields storing information such as common name, email address, telephone number of the end user. According to the fields contained by a leaf LDAP entry, new object classes defining specific fields can be formed and assigned to the entries. The requirement to define new object classes is dependent on the specific LDAP server selected in the implementation. If the utilities provided by the LDAP server selected by the implementer define all the object classes defining the fields contained by all the LDAP entries, there is no need to define a new object class. Information about LDAP can be found at [14].

The security across the communication between the end user and the web application is provided by using SSL, forms-based authentication over SSL, user-based authorization and subject-based authorization. The secure communication infrastructure between the end user and the web application must be provided by the web server (in which the web application is loaded). Namely, the administrator of the web server must configure the web server to use SSL for the web application. Forms-based authentication logic is loaded on the login page of the web application. User-based authorization and subject-based authorization are used to authorize access to the default web page of the web application, so they are handled by the default web page.

The architecture of the default web page of the web application is comprised mainly of two parts. The first part contains the user interface elements with which the end user can interact to send commands to the ZigBee master controller. In this part, there are also user interface elements used to show the results of the commands sent by the end user to the ZigBee master controller. The second part contains the event handling routines handling the events caused by activating the user interface elements in the first part, and the web page's lifetime events. The architecture of the default web page of the web application is depicted in the below figure.

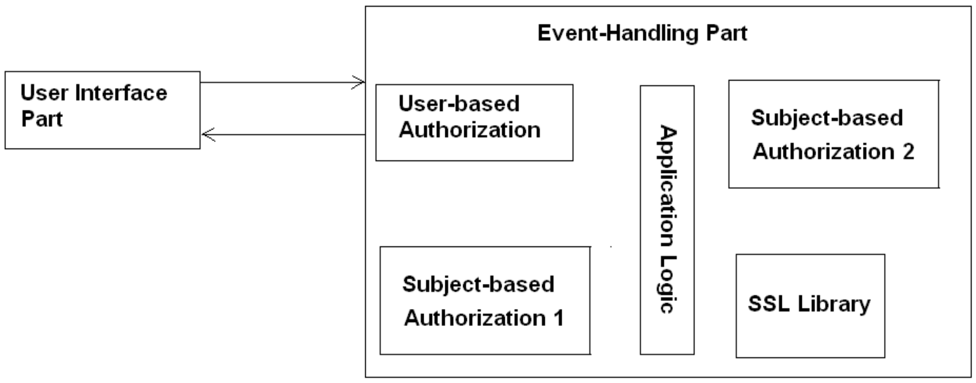


Figure 3.8: The architecture of the default web page of the web application

User-based authorization and subject-based authorization 1 logic in the event-handling part of the default web page are used to authorize access to the default web page based on the identity of the end user. User-based authorization uses the login name of the end user authenticated against a user store whereas, subject-based authorization 1 uses the subject name field of an X509 certificate of the end user to authorize access to the default web page. If one of these authorization mechanisms does not succeed, the end user is not allowed to download the default web page into his browser.

Event-handling part of the default web page uses SSL library to communicate with the ZigBee master controller in a secure way. It must use the appropriate SSL library services to authenticate the server, namely the ZigBee master controller. Event-handling part also authorizes access to itself by the server by using the subject name field of the X509 certificate of the server. This authorization is handled by subject-based authorization 2 part shown in the above figure.

The default web page opens a TCP connection (TCP client socket) to the ZigBee master controller. It secures this TCP connection via SSL. The IP address and active TCP port information of the ZigBee master controller is hard-coded into the web page creating this TCP connection. By using this connection, the default web page can send the commands given by the end user to the ZigBee master controller. A command given by the end user is first converted by the default web page to a form compliant with the specific application protocol defined in this thesis. Then, the default web page sends the command to the ZigBee master controller byte by byte and waits for a response from the ZigBee master controller. After receiving and processing the response, the default web page displays the response to the end user in a user-friendly way.

The end user can get the current list of ZigBee slave devices, joined to the network coordinated by the ZigBee master controller, by sending a specific command to the ZigBee master controller. The list of these ZigBee slave devices is shown in a list (device list) in the user interface part of the default web page. The end user can select a ZigBee slave device from this list and send a command to the selected device indirectly through the ZigBee master controller. When the end user selects a ZigBee slave device from the device list, the clusters associated with the selected ZigBee slave device appear in another list (cluster list) in the user interface part of the default web page. The end user can select a cluster from the cluster list. When the end user selects a cluster from the cluster list, the attributes associated with the selected cluster and the user interface controls used to give commands related with these attributes appear in a section of the user interface part of the default web page. The end user can give commands to the selected ZigBee slave device (in the device list) by filling and activating these user interface controls.

When the end user activates a control in the user interface part of the default web page to give a command to the ZigBee master controller, the event-handling part of the default web page fetches the data in the user interface elements filled by the end user and prepares a command compliant with the specific application protocol. Then, the event-handling part sends this command to the ZigBee master controller over the TCP socket created and controlled by itself, and waits for a response. After receiving the response from the ZigBee master controller, the event-handling part decodes the response which is compliant with the specific application protocol. The event-handling part fetches the information in the data fields of the response and uses this information to fill the user interface controls (in the user interface part of the default web page) used to show the response to the end user.

3.7 General Security Architecture of the System

The communication over Internet between any two entities of the system formed in this thesis is secured by using SSL on both entities. There are sometimes additional security mechanisms working over SSL on one or both of these entities. The security across the communications on the ZigBee wireless network is provided by using security services provided by the ZigBee wireless protocol. In the previous sections, some issues about the security mechanisms used throughout the system are discussed. The aim of this section is to provide a general view of the system in terms of security mechanisms used. The figure below depicts general security architecture of the system.

The communication between the PC and the web server is secured with SSL. The web server shall be configured to use SSL for the web application. Over SSL, there are three security mechanisms used to secure the communication between the PC and the web server. These are all handled by the web application. These mechanisms are forms-based authentication, user-based authorization and subject-based authorization. Forms-based authentication verifies the login name and password entered by the end user of the PC against a user credentials store. It also automatically authenticates the user in the user’s future access to the web application by using specific mechanisms, if the user wishes. User-based authorization restricts access to the web application based on the login name supplied by the end user. Subject-based authorization restricts access to the web application based on the subject name field of the X509 certificate of the user collected by the web application during SSL handshake.

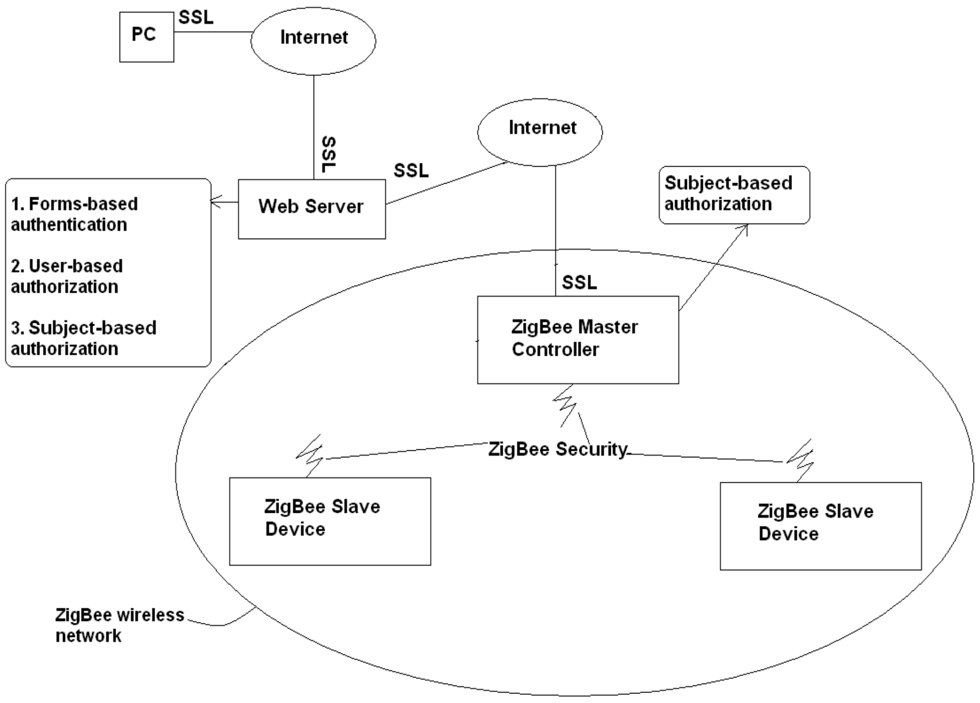


Figure 3.9: General security architecture of the system

The communication between the web server and the ZigBee master controller is also secured with SSL used by both parties. The web application uses its own SSL implementation independent of the web server. The ZigBee master controller also has an SSL implementation. The web application configures its SSL implementation to request a server certificate and authenticate the server. The ZigBee master controller configures its SSL implementation to request a client certificate and authenticate the client. Each party performs subject-based authorization which restricts access to the party depending on the subject name field of the X509 certificate of the accessor collected during SSL handshake.

The communication inside the ZigBee wireless network formed in the thesis is secured by using the security services provided by the ZigBee protocol. Every ZigBee device shall configure its ZigBee stack to use the same security mode provided by ZigBee. By this way, every ZigBee device uses the same encryption algorithm and same MIC (Message Integrity Code) length.

3.8 Specific Application Protocol

This section explains the specific application protocol used in the communication between the ZigBee master controller and the web server in which the web application is loaded.

In ZigBee protocol, devices and their interactions are defined in profiles including device attributes [1]. Device attributes are grouped in clusters contained in a profile [1]. The thesis work is demonstrated using the Home Control, Lighting Profile defined by ZigBee Alliance. This profile contains descriptions about six devices [9]. These devices are Dimming Load Controller, Switch Load Controller (light), Occupancy Sensor, Dimming Remote Control, Switch Remote Control and Light Sensor Monochromatic [9]. The communication between ZigBee devices (ZigBee master controller and ZigBee slave devices) is performed via messages setting and getting the values of device attributes supported by the ZigBee slave devices. These device attributes are defined in the Home Control, Lighting Profile. ZigBee slave devices can communicate only with the ZigBee master controller by transferring above mentioned ZigBee messages.

There is a question of how an end user communicates with the ZigBee devices. This is done via a web application that can be uploaded into an appropriate web server. This web application receives commands from the end user and sends these commands to the ZigBee master controller which is responsible from controlling ZigBee slave devices. Then, the ZigBee master controller processes these commands relaying them to the ZigBee slave devices if necessary and sends the responses which could be received from the ZigBee slave devices to the web application. After that, the web application responds the end user with the information received from the ZigBee master controller. The web application can only communicate with the ZigBee master controller and cannot directly talk to the ZigBee slave devices.

The application protocol between the web server (in which the web application is loaded) and the ZigBee master controller has two sides, one from the web server to the ZigBee master controller and the other the opposite way. The structure of the messages sent from the web server (web application) to the ZigBee master controller upon end user request is as shown in the below figure.

What each field, shown in the below figure, is used for could be defined as follows. The first field, total number of fields, is used to keep track of total field count in the message. It accounts for its own field. This field is used in the ZigBee master controller to split the message received from the web application into its tokens (fields) and display those tokens. The fields of the message received from the web application are separated by the space character. It is possible to split the message into its tokens without knowing the total number of fields but this field is used for debug purposes also.

total number of fields	**total number of commands**	**Cluster identifier**	**command #1**	**.....**	**command #n**	**device type**

device identifier	**parameter #1**	**.....**	**parameter #m**

Figure 3.10: The fields of a message sent from the web server to the ZigBee master controller

The second field, total number of commands, gives the count of the command fields coming after the third field.

The third field, cluster identifier, gives the identifier of the cluster containing the attributes targeted by the commands in the message. A message can contain more than one command, set or get more than one attribute value. But there is a restriction. A message can only set or get attributes pertaining to one specific cluster. Referencing more than one cluster in a message is strictly prohibited. This is because a ZigBee message sent from the ZigBee master controller to a ZigBee slave device can contain transactions referencing a single cluster, and the ZigBee master controller sends one ZigBee message upon receipt of a message from the web application [1]. (The latter reason is a preference.) There can be more than one transaction in a ZigBee message each of which setting or getting different attributes [1]. But these attributes should pertain to a single cluster [1].

In this thesis work, two ZigBee devices defined in the Home Control, Lighting Profile are implemented as ZigBee slave devices for demonstration. These are switch load controller, namely light, and occupancy sensor. There are three clusters defined for light and two clusters defined for occupancy sensor in the profile [9]. The clusters defined for light are OnOffSRC_CLUSTER, ProgramSLC_CLUSTER and StatusSLC_CLUSTER [9]. A programmatic identifier is assigned for each of these three clusters. These are 1 for OnOffSRC_CLUSTER, 2 for StatusSLC_CLUSTER and 3 for ProgramSLC_CLUSTER. These programmatic identifiers are used as the cluster identifier field in the messages for the application protocol. The clusters defined for occupancy sensor are OccupancyOS_CLUSTER and ProgramOS_CLUSTER [9]. A programmatic identifier is also assigned for each occupancy sensor cluster, 1 for OccupancyOS_CLUSTER and 2 for ProgramOS_CLUSTER.

A cluster is an input or output cluster for the device it is related with [1]. The attributes in an input cluster of a device are only settable by other devices [1]. The attributes in an output cluster of a device are only gettable by other devices [1]. So an attribute of a device is either gettable or settable (but not both) depending on the cluster it is found in [1]. OnOffSRC_CLUSTER and ProgramSLC_CLUSTER are input clusters and StatusSLC_CLUSTER is an output cluster for the light [9]. OccupancyOS_CLUSTER is an output cluster and ProgramOS_CLUSTER is an input cluster for the occupancy sensor [9].

The next fields, command #x fields, contain command names for a specific device. According to the values of these fields, specific attributes supported by the device can be set or got. Each command is related with at most one attribute. If the attribute related with a command is settable, the value of the attribute is set by this command to a specific value. If the attribute related with a command is gettable, the value of the attribute is fetched by this command. One or more commands can correspond to a single attribute.

OnOffSRC_CLUSTER for the light contains one attribute named OnOffSRC_OnOff [9]. Three commands are related with this settable attribute. These commands are turnon, turnoff and toggle commands. These commands turn on, turn off and toggle the on/off status of the light implemented by the target device, respectively. The attribute mentioned above can take one of three values corresponding to the turn on, turn off or toggle action determined by the corresponding command #x field.

According to [9], ProgramSLC_CLUSTER for the light contains the following attributes: ProgramSLC_Override, ProgramSLC_Auto, ProgramSLC_FactoryDefault, ProgramSLC_PresetO_LSSLC, ProgramSLC_ResetO_LSSLC, ProgramSLC_BrownOutMinVolt, ProgramSLC_ShutDownPkCurrent and ProgramSLC_MeteringPeriod. Last four of these attributes are implemented in this thesis work and the corresponding command names are resetloadstatus, brownout, shutdown and metering. These commands are respectively used to reset the load status of the light, set the brownout minimum voltage of the light to a specific value, set the shutdown peak current of the light to a specific value and set the metering period of the light to a specific value. In fact, what these commands do is not important for us. The important thing is which attributes are affected by these commands and how these attributes are affected. Last four of the attributes mentioned above are settable by these commands and ProgramSLC_ResetO_LSSLC attribute is set to 0 when resetloadstatus command is received. Each of the other three attributes is set to the value of the corresponding parameter #x field when corresponding command is received.

The last cluster, StatusSLC_CLUSTER for the light contains the following attributes according to [9]: StatusSLC_OnOff, StatusSLC_RunTime, StatusSLC_Watts, StatusSLC_TotalPower, StatusSLC_Energy, StatusSLC_Vars, StatusSLC_Voltage, StatusSLC_Current, StatusSLC_Ripple, StatusSLC_Frequency, StatusSLC_Phase, StatusSLC_PhaseAngle, StatusSLC_PF, StatusSLC_PeakWatts, StatusSLC_PeakTotalPower, StatusSLC_PeakVoltage, StatusSLC_PeakCurrent and StatusSLC_LoadType. All of these attributes are implemented in this thesis work. The corresponding command names respectively are onoff, runtime, watts, power, energy, vars, voltage, current, ripple, frequency, phase, angle, pf, peakwatts, peakpower, peakvoltage, peakcurrent and loadtype. When one of these commands is received, the value of the corresponding attribute is returned.

OccupancyOS_CLUSTER for the occupancy sensor contains one attribute named OccupancyOS_CurrentState [9]. The command name for this attribute is chosen to be currentstate. When this command is received, the value of the attribute mentioned above is returned.

According to [9], ProgramOS_CLUSTER for the occupancy sensor contains the following attributes: ProgramOS_ReportTime, ProgramOS_TimeOut, ProgramOS_CurrentStateOn, ProgramOS_CurrentStateOff, ProgramOS_Override, ProgramOS_Auto and ProgramOS_FactoryDefault. First four of these attributes are implemented in this thesis work. The command names given to the implemented attributes are respectively reporttime, timeout, currentstateon, currentstateoff. When one of these commands is received, the value of the corresponding attribute is set to the value of the corresponding parameter #x field.

There can be other command names that are not related with an attribute in any cluster. These type of commands are for general actions. An example of this is the command getdevicelist. It is used by the web application to get the current list of ZigBee slave devices from the ZigBee master controller.

The key point of commands is that if a command is related with an attribute, it either causes a get operation or a set operation on the attribute but not both depending on the attribute. Recall that an attribute of a device is either gettable or settable (but not both) depending on the cluster it is found in [1]. If the command is related with a set operation, there is a corresponding parameter field at the end of the message. The value of this parameter field is used for the set operation on the attribute the command is related with. If the command is related with a get operation, there is no such parameter field. For each command, related with an attribute, received from the web application, the ZigBee master controller sets up a set or get transaction according to whether the command causes a set or get operation. Then, it combines the prepared transactions in a ZigBee message to be sent to a specific ZigBee slave device.

The next field is the device type field. Two types of devices are implemented in this thesis work, namely the light and the occupancy sensor. The device type name used for the light in the program is “light” and for the occupancy sensor is “OS”. This field and the device identifier field coming next together determine a specific ZigBee slave device. There is a specific device type name used for general actions, it is “all”.

The next field is the device identifier field. This field is used to identify the device to which the ZigBee message prepared by the ZigBee master controller is sent. Device identifiers are assigned by the ZigBee master controller for each slave device (slave endpoint). Device identifiers start counting from 1 for each device type. For example, if there are two lights in the system, the first light is named “light 1” and the second light is named “light 2” by the ZigBee master controller. In this naming scheme, light is the device type and 1 and 2 are device identifiers.

Next and finally comes parameter #x fields. If any command in the message causes a set attribute operation, there will be a corresponding parameter #x field in the message, the value of which indicates the value to which the corresponding attribute is set. For each set operation (command), there will be at most one parameter field. The value in the parameter field corresponding to a set command is added to a set transaction prepared by the ZigBee master controller for that command.

The structure of the message sent by the ZigBee master controller to the web server in response to the web server’s message containing attribute set or get commands is shown in the below figure:

Total number of fields	**Set operation result**	**get operation result**	**.....**	**get operation result**	**set operation result**

Figure 3.11: The structure of the message sent from the ZigBee master controller to the web server in response to the web server’s message containing attribute set or get commands

The number of fields in the message is variable again. The fields of the message are separated by the semicolon character. The first field contains the total number of fields in the message. This field is used as an error checking mechanism when splitting the message into its fields (tokens) by the web application. The web application stores the number of commands sent to the ZigBee master controller in a single message. The first field of the message, coming from the ZigBee master controller in response to the web application’s message, should be equal to the stored number of commands plus 1 if an error does not occur in transit. If they are not equal, there is an error in the message received from the ZigBee master controller and the message is ignored and an error message is displayed.

The remaining fields in the message shown in the above figure are used to convey the results of set and get attribute operations caused by the commands in the message sent by the web application to the ZigBee master controller. The order of operation results is the same as the order of corresponding commands in the message sent to the ZigBee master controller. The set operation result can be the string “Yes” or “No” depending on whether the set attribute operation is completed successfully by the ZigBee slave device or not. The get operation result fields are filled with the actual results of the get attribute operations, namely the values for the corresponding attributes. The get and set operation result fields are used by the web application to return responses to the end user’s requests.

There is a specific message returned by the ZigBee master controller to the web application when the ZigBee slave device, whose attribute is intended to be set or got, is not found by the ZigBee master controller in its database of networked slave devices. This message is “2;Device not found”. When the web application receives this message, it understands that the device which the end user is commanded left from the network controlled by the ZigBee master controller. The web application displays a suitable error message to the end user.

There is another specific message returned by the ZigBee master controller to the web application when the ZigBee slave device, whose attribute is intended to be set or got, exists in the ZigBee master controller’s database of networked slave devices; but does not respond in a specified time interval to the ZigBee master controller. This message is “2;No connection”. When the web application receives this message, it understands that the ZigBee slave device, which the end user is commanded, exists but is not reached by the ZigBee master controller. The web application displays a suitable error message to the end user and the end user can then try to resend the command to that ZigBee slave device.

The structure of the message shown in the figure above is the general format of a message sent by the ZigBee master controller to the web server in response to attribute set or get operations. There is an alternative message sent by the ZigBee master controller to the web server in response to the web server’s message containing the command getdevicelist. The structure of this message is shown in the below figure.

Total number of fields	**device type**	**Device identifier**	**.....**	**device type**	**device identifier**

Figure 3.12: The structure of the message sent from the ZigBee master controller to the web server in response to the web server’s message containing the command getdevicelist

The number of fields in the above message is again varying. The fields of the above message are again separated by the semicolon character. The first field contains the total number of fields in the message. After this field, any number of device type – device identifier field pairs come. A device type – device identifier field pair corresponds to one specific ZigBee slave device found in the network controlled by the ZigBee master controller. These slave devices are identified by the ZigBee master controller when they join the network controlled by the ZigBee master controller.

Device type field is an integer determining whether the device is a light or an occupancy sensor. If the device type field is 1, the device is a light and if it is 2, the device is an occupancy sensor.

Device identifier field determines the identifier of the device whose type is given by the previous device type field. As mentioned before, device identifiers start counting from 1 for each device type and are given by the ZigBee master controller.

The message, whose structure is given in the above figure, indicates to the web application the ZigBee slave devices currently found in the network controlled by the ZigBee master controller.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 Hardware

In this section, hardware details of ZigBee slave devices and ZigBee master controller are described. There is one master controller and there are two slave devices implemented for demonstration purposes in the ZigBee wireless network formed in this thesis work. One of the slave devices takes the role of light and the other slave device takes the role of occupancy sensor. The role of a slave device is determined by the software loaded into the slave device.

4.1.1 ZigBee Slave Device Hardware

The hardware configurations of two ZigBee slave devices are similar. The board used in the implementation of ZigBee slave devices is PICDEM Z motherboard developed by the Microchip company. The hardware details of this board can be found in [8]. A 40-pin PIC microcontroller is used as a central processing unit (CPU) in the design of ZigBee slave devices. The specific 40-pin PIC microcontroller used is PIC18LF4620. This 40-pin microcontroller is placed into the 40-pin microcontroller socket found on PICDEM Z motherboard. In fact, PICDEM Z motherboard is shipped with a PIC18LF4620 microcontroller placed into the 40-pin microcontroller socket on PICDEM Z motherboard [8]. The hardware implementation of a ZigBee slave device is shown in the below figure.

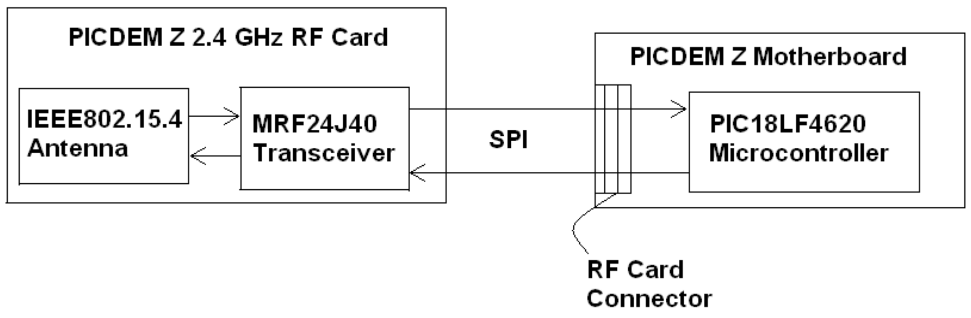


Figure 4.1: The hardware implementation of a ZigBee slave device

There should also be a IEEE 802.15.4 transceiver and a radio frequency (RF) antenna in ZigBee slave devices for wireless ZigBee communication with the ZigBee master controller. A ZigBee RF card containing a ZigBee transceiver and an RF antenna is used for this purpose. The specific ZigBee RF card used in ZigBee slave devices is PICDEM Z 2.4 GHz RF Card produced by the Microchip company.

The IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF card should be connected to the 40-pin microcontroller on PICDEM Z motherboard in ZigBee slave devices for it to be commanded by the microcontroller. PICDEM Z motherboard provides an RF card connector which is used to connect a supported RF card to PICDEM Z motherboard [8]. This RF card connector provides +3.3V DC, an SPI bus and a few discrete digital I/O control signals [8]. By using this RF card connector, the PICDEM Z 2.4 GHz RF card is connected to PICDEM Z motherboard. With this connection, the MRF24J40 IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF card becomes connected to the PIC18LF4620 microcontroller on PICDEM Z motherboard in a ZigBee slave device.

The interface between the MRF24J40 IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF card and the PIC18LF4620 microcontroller on PICDEM Z motherboard is Serial Peripheral Interface (SPI). Detailed information about SPI can be found at [5, 6, 7]. The resulting pin mappings between PIC18LF4620 on PICDEM Z motherboard and the MRF24J40 IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF card in ZigBee slave devices are shown in the below table.

Table 4.1: Pin Mappings between PIC18LF4620 and the MRF24J40 IEEE802.15.4 transceiver in ZigBee Slave Devices

PIC18LF4620 pin	**MRF24J40 IEEE802.15.4 transceiver pin**
RC2	RESET
RC1	WAKE
RB0	INT
RC4	SDO
RC5	SDI
RC3	SCK
RC0	CS

In the above table, the pin in the first column in each row is connected to the pin in the second column in that row. The pin mappings between PIC18LF4620 and the MRF24J40 IEEE802.15.4 transceiver shown in the above table are not strict. The designer may choose to connect PIC18LF4620 on PICDEM Z motherboard to the MRF24J40 IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF card directly without using the RF card connector on PICDEM Z motherboard and may adjust the pin mappings as he/she wants. However, in the software, the designer should change the corresponding parts for the pin mappings appropriately for the changes he/she makes physically in the pin mappings to take effect.

In the SPI communication between the MRF24J40 IEEE802.15.4 transceiver and PIC18LF4620 microcontroller, the MRF24J40 IEEE802.15.4 transceiver is the slave device and PIC18LF4620 microcontroller is the master device so the PIC18LF4620 microcontroller supplies the clock source and initiates the communication. In addition, PIC18LF4620 microcontroller sends a low voltage signal (logic 0) to the CS pin of the MRF24J40 IEEE802.15.4 transceiver, so the MRF24J40 IEEE802.15.4 transceiver is chosen to be the slave device for SPI communication with the PIC18LF4620 microcontroller.

The main limitation imposed by the ZigBee slave device hardware on the system implemented in the thesis is caused by the small amount of memory of PIC18LF4620 microcontroller on PICDEM Z motherboard. The size of program memory (ROM) of PIC18LF4620 is 64KB and the size of RAM on PIC18LF4620 is 3986 bytes [23]. These two quantities are enough for a ZigBee slave node containing at most two active endpoints according to my observations. If there are more than two endpoints on which specific device implementations reside, then RAM and ROM on PIC18LF4620 may not be adequate. In the thesis, one specific device logic is implemented for each ZigBee slave node, namely each slave node contains one active endpoint. Although each ZigBee slave node contains one active endpoint, approximately %77 of ROM on PIC18LF4620 and %56 of RAM on PIC18LF4620 is used. As a solution to this limitation, an EEPROM can be connected to PIC18LF4620 microcontroller by locating it on the prototyping area on PICDEM Z motherboard. This EEPROM can be used to store both program instructions and data.

4.1.2 ZigBee Master Controller Hardware

ZigBee master controller has mainly two functionalities. First, it should connect to an Ethernet, connected to the Internet, via an Ethernet controller so that it can communicate with a PC connected to the Internet. Second, it should communicate with ZigBee slave devices using wireless ZigBee communication protocol.

In this thesis work, it was decided that the ZigBee master controller was implemented with two microcontrollers communicating with each other. Each main functionality of the ZigBee master controller is assigned to one microcontroller. The microcontrollers selected for use in the ZigBee master controller are the 100-pin PIC24FJ128GA010 microcontroller and the 40-pin PIC18LF4620 microcontroller. One important reason behind these selections is that they are plug-in compatible with the motherboards used in the implementation of the ZigBee master controller.

The ZigBee master controller device is implemented by using two motherboards. One of the boards used is Explorer 16 development board produced by the Microchip company. The hardware details of this board can be found in [10]. The other board is PICDEM Z motherboard used also in the implementation of ZigBee slave devices. The two boards used in the implementation of the ZigBee master controller should communicate and therefore they are connected via RS-232 serial interface. Both of the boards support RS-232 serial interface. One important reason behind the selection of these motherboards is that the Ethernet Card and ZigBee RF Card used in the implementation of ZigBee master controller are plug-in compatible with Explorer 16 development board and PICDEM Z motherboard respectively.

A Plug-In Module (PIM) containing a 100-pin PIC24FJ128GA010 microcontroller is plugged into the 100-pin PIM socket of the Explorer 16 development board. The 40-pin PIC18LF4620 microcontroller is plugged into the 40-pin microcontroller socket on PICDEM Z motherboard. The hardware implementation of the ZigBee master controller is shown in the below figure.

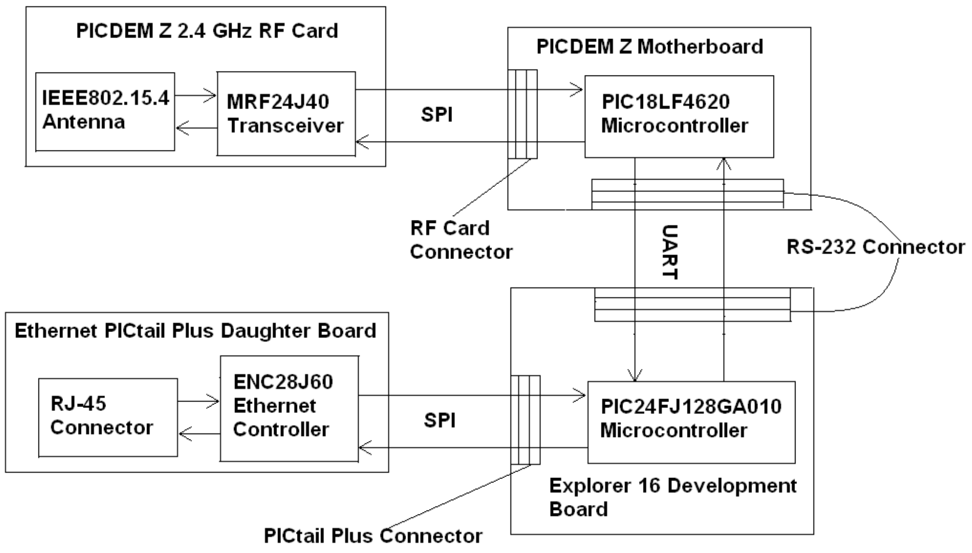


Figure 4.2: The hardware implementation of the ZigBee master controller

The PIC24FJ128GA010 on the ZigBee master controller is responsible from the first functionality mentioned above. Namely, the PIC24FJ128GA010 is used to command the Ethernet controller to connect to an Ethernet connected to the Internet. The second microcontroller, PIC18LF4620, is used to send commands to the ZigBee RF Card to communicate with the ZigBee slave devices.

The two-microcontroller system of the ZigBee master controller works as follows. A PC connected to the Internet sends a command to the ZigBee master controller through the Internet. The first microcontroller, PIC24FJ128GA010, on the ZigBee master controller takes the command, sent by the PC, via the Ethernet controller connected to it. The PIC24FJ128GA010 then relays the command to the second microcontroller, PIC18LF4620. The PIC18LF4620 takes the command, parses the command and sends appropriate commands to the appropriate ZigBee slave device through the ZigBee RF Card connected to it. Then, the PIC18LF4620 takes the reply from the ZigBee slave device through the ZigBee RF Card again. It processes the reply and sends it to the PIC24FJ128GA010. The

PIC24FJ128GA010 relays the reply to the PC, which sent the initial command, via the Ethernet controller through the Internet.

A board containing an Ethernet controller is connected to the Explorer 16 development board in the ZigBee master controller. This board is Ethernet PICtail Plus Daughter Board. Ethernet PICtail Plus Daughter board is connected to the Explorer 16 development board by means of one of the PICtail Plus Card edge modular expansion connectors found on the Explorer 16 development board. These connectors on the Explorer 16 development board makes it easy to connect a PICtail Plus Card to the Explorer 16 development board by eliminating the need to manually connect the controller on the PICtail Plus Card to the microcontroller on the Explorer 16 development board. By connecting the Ethernet PICtail Plus Daughter board to the Explorer 16 development board by using the PICtail Plus Card edge modular expansion connector found on the Explorer 16 development board, ENC28J60 Ethernet controller on the Ethernet PICtail Plus Daughter board becomes connected to the PIC24FJ128GA010 microcontroller on the Explorer 16 development board. The communication interface between the ENC28J60 Ethernet controller and the PIC24FJ128GA010 microcontroller is SPI. The pin mappings between the ENC28J60 Ethernet controller on the Ethernet PICtail Plus Daughter Board, and the PIC24FJ128GA010 microcontroller (on the Explorer 16 development board) are shown in the below table.

In the SPI communication between the PIC24FJ128GA010 and the ENC28J60 Ethernet controller in the ZigBee master controller, the PIC24FJ128GA010 is the master device and the ENC28J60 Ethernet controller is the slave device. The clock source is supplied by the master device PIC24FJ128GA010. The PIC24FJ128GA010 sends a low voltage signal to the CS pin of the ENC28J60 Ethernet controller so that the ENC28J60 Ethernet controller becomes a slave device in SPI communication.

Table 4.2: Pin Mappings between PIC24FJ128GA010 microcontroller and ENC28J60 Ethernet controller in the ZigBee Master Controller

PIC24FJ128GA010 pin	**ENC28J60 Ethernet controller pin**
RF6	SCK
RF7	SDO
RF8	SDI
RB0	INT
RD14	CS
RD15	RESET

The pin mappings shown in the above table can be changed physically, or the communication mechanism (SPI) can be changed, or both changes can be applied simultaneously. In such a case, the software loaded to the PIC24FJ128GA010 should be changed accordingly to reflect the physical changes and/or implement the new communication mechanism.

The Ethernet PICtail Plus Daughter Board is plugged into the top 30-pin segment of the populated PICtail Plus Card edge modular expansion connector on the Explorer 16 development board. If desired, it can be plugged into the middle 30-pin segment of the same populated connector. However, in this case the software loaded into the PIC24FJ128GA010 microcontroller, connected to the ENC28J60 Ethernet controller on the Ethernet PICtail Plus Daughter Board, should be changed to reflect the use of SPI2 pins for communication between the ENC28J60 Ethernet controller and the PIC24FJ128GA010 microcontroller.

By connecting the ENC28J60 Ethernet controller on the Ethernet PICtail Plus Daughter Board to the PIC24FJ128GA010 and then connecting the Ethernet PICtail Plus Daughter Board to an Ethernet (which has access to the Internet) via an Ethernet cable, the connection of the ZigBee master controller to the Internet is achieved.

The ZigBee master controller should participate in another communication to fulfill its second functionality. It sends and receives commands to/from the ZigBee slave devices using ZigBee wireless communication protocol. The second microcontroller on the ZigBee master controller, PIC18LF4620, performs this functionality via ZigBee RF Card. The PICDEM Z 2.4 GHz RF Card is used as the ZigBee RF Card. The PICDEM Z 2.4 GHz RF Card is connected to PICDEM Z motherboard, containing the PIC18LF4620 microcontroller, by means of the RF card connector on PICDEM Z motherboard. With this connection, the MRF24J40 IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF Card becomes connected to the PIC18LF4620 microcontroller on PICDEM Z motherboard. The communication between the MRF24J40 IEEE802.15.4 transceiver and the PIC18LF4620 microcontroller is performed via SPI mechanism. The pin mappings between the MRF24J40 IEEE802.15.4 transceiver on the PICDEM Z 2.4 GHz RF Card and the PIC18LF4620 microcontroller on PICDEM Z motherboard on the ZigBee master controller are shown in the below table.

In the SPI communication between the MRF24J40 IEEE802.15.4 transceiver and the PIC18LF4620 microcontroller, the PIC18LF4620 microcontroller is the master device. It supplies the clock source and initiates the communication. It also sends a low voltage signal (logic 0) to the CS pin of the MRF24J40 IEEE802.15.4 transceiver so that the MRF24J40 IEEE802.15.4 transceiver is selected as the slave device in the SPI communication.

Table 4.3: Pin Mappings between PIC18LF4620 microcontroller and MRF24J40 IEEE802.15.4 transceiver in the ZigBee Master Controller

PIC18LF4620 pin	**MRF24J40 IEEE802.15.4 transceiver pin**
RC2	RESET
RC1	WAKE
RB0	INT
RC4	SDO
RC5	SDI
RC3	SCK
RC0	CS

In the Explorer 16 development board used by the ZigBee master controller, there is a 2*16 LCD which has a connection to the 100-pin PIM socket [10]. The LCD is connected to the PIC24FJ128GA010 microcontroller on the Explorer 16 development board and is used to display debug messages produced by the ZigBee master controller. The debug messages that are produced by the PIC18LF4620 microcontroller (on PICDEM Z motherboard) in the ZigBee master controller are sent to the PIC24FJ128GA010 microcontroller (on the Explorer 16 development board) in the ZigBee master controller and the PIC24FJ128GA010 microcontroller sends the received debug messages to the LCD. The 40-pin PIC18LF4620 microcontroller does not have direct access to the LCD.

The communication between the PIC24FJ128GA010 and the PIC18LF4620 microcontrollers in the ZigBee master controller is performed via the Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) module found in these microcontrollers. The connection between the two microcontrollers is achieved by means of the RS-232 serial ports provided by the boards on which these microcontrollers reside (namely, the Explorer 16 development board and the

PICDEM Z motherboard). The RS-232 serial port of the Explorer 16 development board of the ZigBee master controller is connected to the RS-232 serial port of the PICDEM Z motherboard of the ZigBee master controller via a male-to-male crossover RS-232 serial cable. By this way, the EUSART connection between the two microcontrollers of the ZigBee master controller is achieved. The pin mappings between the PIC24FJ128GA010 and PIC18LF4620 microcontrollers in the ZigBee master controller are shown in the below table.

Table 4.4: Pin Mappings between PIC24FJ128GA010 and PIC18LF4620 microcontrollers in the ZigBee Master Controller

PIC24FJ128GA010 pin	**PIC18LF4620 pin**
RF4	RC7
RF5	RC6

The PIC24FJ128GA010 and PIC18LF4620 microcontrollers are both placed in EUSART asynchronous mode. The communication between these two microcontrollers is full-duplex. Since there are limited number of byte-sized buffers for holding the bytes received via EUSART in both of the microcontrollers, there must be some mechanism avoiding the fast sender from causing buffer overflow in the slow receiver which takes and processes the data slower than the sender's preparation and transmission of data. An acknowledgement mechanism is used in software to avoid such a buffer overflow in the receiver (in both of the microcontrollers since each of them receives data via EUSART from the other).

The main limitations imposed by the ZigBee master controller hardware on the system implemented in the thesis are also caused by memory inadequacy of the microcontrollers used by the ZigBee master controller. The software (ZigBee application program) loaded on PIC18LF4620 microcontroller on PICDEM Z motherboard on the ZigBee master controller uses approximately %97 of ROM and %91 of RAM of PIC18LF4620. Adding extra functionality such as ZigBee indirect messaging (binding) to ZigBee application program of the ZigBee master controller seems to be impossible.

The size of the program memory (ROM) of PIC24FJ128GA010 microcontroller is 128KB and the size of RAM of PIC24FJ128GA010 is 8KB [24]. The software (TCP/IP application program) loaded on PIC24FJ128GA010 microcontroller on Explorer 16 development board on the ZigBee master controller uses approximately %32 of ROM and %30 of RAM of PIC24FJ128GA010. Although these two percentages are not high values, adding specific future improvements such as embedding a web server within PIC24FJ128GA010 microcontroller on the ZigBee master controller, adding client authentication to Microchip SSL implementation loaded on PIC24FJ128GA010 microcontroller, may easily result in overrun of memory of PIC24FJ128GA010 microcontroller. These specific future improvements are very important for the performance increase across the whole system implemented in the thesis.

4.2 Software

In this section of the thesis; mainly, the semantics of the application programs of the devices of the system are explained. Data structures, algorithms and general code structure related with these application programs are detailed by using diagrams and code snippets where necessary. In addition to the application programs, the specific third-party stacks and implementations used in the devices are also mentioned. Additionally, the details of the web application program are explained.

The ZigBee master controller has two application programs as mentioned in Section 3.3 (“ZigBee Master Controller Software Architecture”) of this thesis.

These application programs are called TCP/IP application program and ZigBee application program (as indicated in Section 3.3 of this thesis). Each of the ZigBee slave devices has one application program called slave application program, as mentioned in Section 3.5 ("ZigBee Slave Device Software Architecture") of this thesis. Slave application program of the ZigBee slave device implementing the logic of light is called as Light slave application program and slave application program of the ZigBee slave device implementing the logic of occupancy sensor is called as Occupancy Sensor slave application program throughout the thesis.

The source files for the ZigBee master controller and the ZigBee slave devices are written in the C programming language for PIC microcontrollers. TCP/IP application program is supposed to execute on the 16-bit PIC24FJ128GA010 microcontroller. It is compiled with the MPLAB C30 C Compiler which can produce object code for the PIC24F family of microcontrollers and is developed by the Microchip company. The resulting object files produced by the MPLAB C30 C Compiler are linked with the MPLAB LINK30 Object Linker which can link object code produced for the PIC24F family, and is developed by the Microchip company.

ZigBee application program and slave application programs are compiled with the MPLAB C18 C Compiler which can produce object code for the PIC18 family of microcontrollers and is developed by the Microchip company. The resulting object files produced by the MPLAB C18 C Compiler are linked with the MPLINK Object Linker which can link object code produced for the PIC18 family, and is developed by the Microchip company.

The writing, compilation and linking of all of the code is performed in the integrated development environment MPLAB IDE also developed by the Microchip company. The subsections of this section describes ZigBee master controller software implementation issues, TCP/IP application program, ZigBee application program, ZigBee slave device software implementation issues, slave application programs and web application code.

4.2.1 ZigBee Master Controller Software

The ZigBee master controller has two microcontrollers, namely the PIC24FJ128GA010 (found on the Eplorer 16 development board) and the PIC18LF4620 (found on the PICDEMZ motherboard). The ZigBee master controller connects to an Ethernet (connected to the Internet) via the PIC24FJ128GA010 microcontroller and the Ethernet controller connected to the PIC24FJ128GA010. By this way, it can communicate with any PC connected to the Internet through the Internet via a web server in which a specific web application is loaded. The communication between the ZigBee master controller and such a web server is performed through a specific application protocol which works over TCP/IP. Therefore, the ZigBee master controller should have a TCP/IP software stack implemented on the PIC24FJ128GA010 microcontroller. The ZigBee master controller should also protect the TCP/IP communication between itself and the web server with SSL. The Microchip TCP/IP Stack version 4.55 containing an SSL implementation is used as the TCP/IP software stack in the ZigBee master controller. It is loaded on the program memory of the PIC24FJ128GA010 microcontroller on the ZigBee master controller.

The ZigBee master controller also communicates with the ZigBee slave devices through the ZigBee wireless communication protocol. Therefore, there must be a ZigBee protocol software stack loaded on the ZigBee master controller. The Microchip Stack for the ZigBee protocol is used as the ZigBee protocol software stack in the ZigBee master controller. The Microchip Stack for the ZigBee protocol is loaded into the program memory of the PIC18LF4620 microcontroller in the ZigBee master controller.

In the following two subsections of this section, TCP/IP application program and ZigBee application program of the ZigBee master controller are described.

4.2.1.1 TCP/IP Application Program

The communication between a PC, connected to the Internet, and the ZigBee master controller is performed via a web server in which a specific web application is loaded. The communication between the web server and the ZigBee master controller is performed via a specific application protocol working over TCP/IP and this communication is implemented by TCP/IP application program on the ZigBee master controller side. TCP/IP application program implements TCP/IP connection to an Ethernet connected to the Internet. By this way, the ZigBee master controller can connect to the Internet and communicate with the web server also connected to the Internet.

TCP/IP application program is loaded onto the program memory of the PIC24FJ128GA010 microcontroller (on the Explorer 16 development board) of the ZigBee master controller. Besides providing TCP/IP connection to an Ethernet, TCP/IP application program also relays the commands, received from a PC via the web server, to ZigBee application program loaded on the PIC18LF4620 microcontroller (on the PICDEM Z motherboard) of the ZigBee master controller. It also receives the responses to these commands from ZigBee application program. So there must be a communication mechanism between the two application programs loaded on two seperate microcontrollers of the ZigBee master controller. This communication mechanism is selected to be EUSART supported by both microcontrollers. An acknowledgement mechanism is implemented in software over EUSART communication between the two application programs to avoid receive buffer overflows.

There are five source files implemented in the context of this thesis work for TCP/IP application program. Three of these files are C header files and two of them are C source files. These five files shall be compiled into a MPLAB IDE project with the Microchip TCP/IP stack source files. Then this project is built and an executable hex file is generated. Some of the code fragments in these five source files are taken from TCP/IP Demo Application part of the Microchip TCP/IP Solution which also contains the Microchip TCP/IP stack source files. These five source files are named as TCPServer.c, MainApp.c, TCPServer.h, TCPIPConfig.h and HardwareProfile.h.

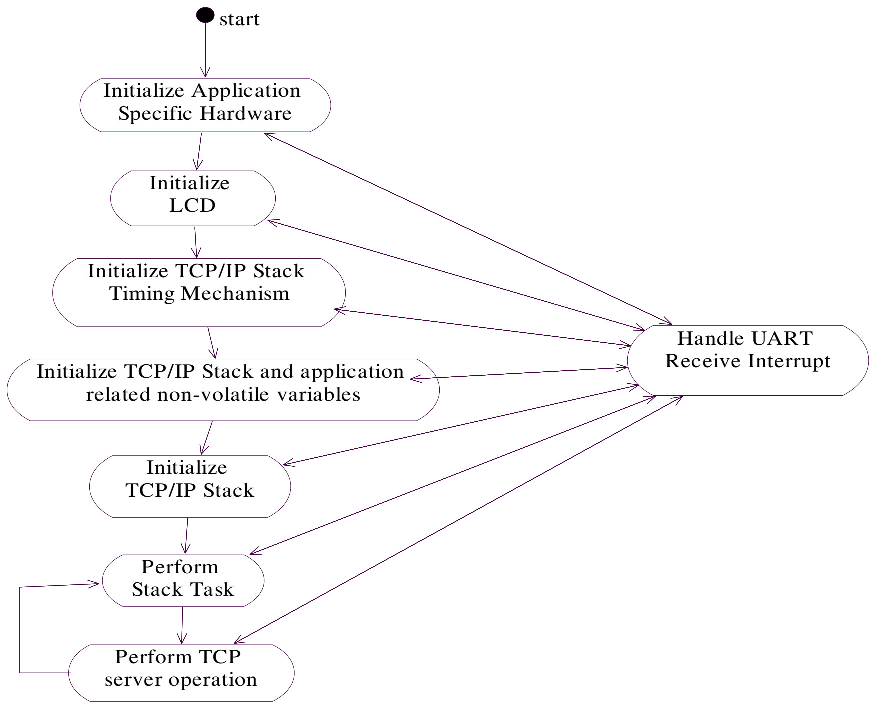

start
Initialize Application Specific Hardware
Initialize LCD
Initialize TCP/IP Stack Timing Mechanism
Initialize TCP/IP Stack and application related non-volatile variables
Initialize TCP/IP Stack
Perform Stack Task
Perform TCP server operation
Handle UART Receive Interrupt

Figure 4.3: The main workflow of TCP/IP application program

The main workflow of the TCP/IP application program is depicted by the activity diagram in the above figure. The operation of TCP/IP application program starts by initializing application specific hardware the most important part of which is initializing the UART subsystem of the PIC24FJ128GA010 microcontroller of the ZigBee master controller. The registers of the PIC24FJ128GA010 microcontroller controlling the operation of the UART subsystem are set to appropriate values. Some of these registers are set according to the baud rate value (19200) chosen for the UART communication between the two microcontrollers of the ZigBee master controller. Also, UART receive interrupts are enabled.

The operation of TCP/IP application program continues by initializing the LCD on the Explorer 16 development board. The LCD is used to display the debug messages produced by the two microcontrollers of the ZigBee master controller. After initializing the LCD, the timing mechanism provided by the Microchip TCP/IP stack is initialized. The timing mechanism works independently from the core stack layers, but is essential for the proper operation of the stack layers.

After initializing the timing mechanism provided by the TCP/IP stack, TCP/IP application program initializes stack and application related non-volatile variables. The most important part of this activity is to initialize the MAC address, IP address, subnet mask, gateway address, primary and secondary DNS server addresses used by the TCP/IP stack in the ZigBee master controller. If there is a DHCP server on the Ethernet to which the ZigBee master controller connects, TCP/IP application program can be changed to handle DHCP messages coming from the DHCP server. In such a case, the IP address used by the TCP/IP stack can be changed according to the messages coming from the DHCP server.

The operation of TCP/IP application program continues by initializing the core TCP/IP stack layers (MAC, ARP, IP, TCP, UDP). After that, TCP/IP application program enters into an infinite loop. In the infinite loop, TCP/IP application program performs normal stack task including checking for incoming packet to the

TCP/IP stack, checking type of incoming packet and calling appropriate stack entity to process the incoming packet. After that in the infinite loop, TCP/IP application program performs TCP server operation which is explained later in this section.

If there is a UART receive interrupt on the PIC24FJ128GA010 microcontroller after enabling the UART receive interrupts, the interrupt is handled by a specific interrupt handler which receives (from ZigBee application program) a specified amount of bytes via the UART subsystem by polling the UART subsystem. The amount of bytes received depends on the first byte received via the UART subsystem. After receiving each byte, the interrupt handler stores the byte and sends an acknowledgement to ZigBee application program. During UART receive interrupt handling, the UART receive interrupts are disabled by the interrupt handler to avoid any further incoming byte coming to the UART subsystem from causing a UART receive interrupt. The workflow for the UART receive interrupt handler routine is shown in the below figure.

The bytes received via the UART subsystem are stored on the appropriate location in RAM of the PIC24FJ128GA010 microcontroller. According to the first byte received via the UART subsystem, the received bytes can be stored in a specific variable or in a buffer to be used when performing TCP server operation in the main workflow of TCP/IP application program. The setting of this specific variable indicates that a response message came from ZigBee application program and is waiting to be processed. The buffer is called ZigBee receive buffer and holds the response message coming from ZigBee application program.

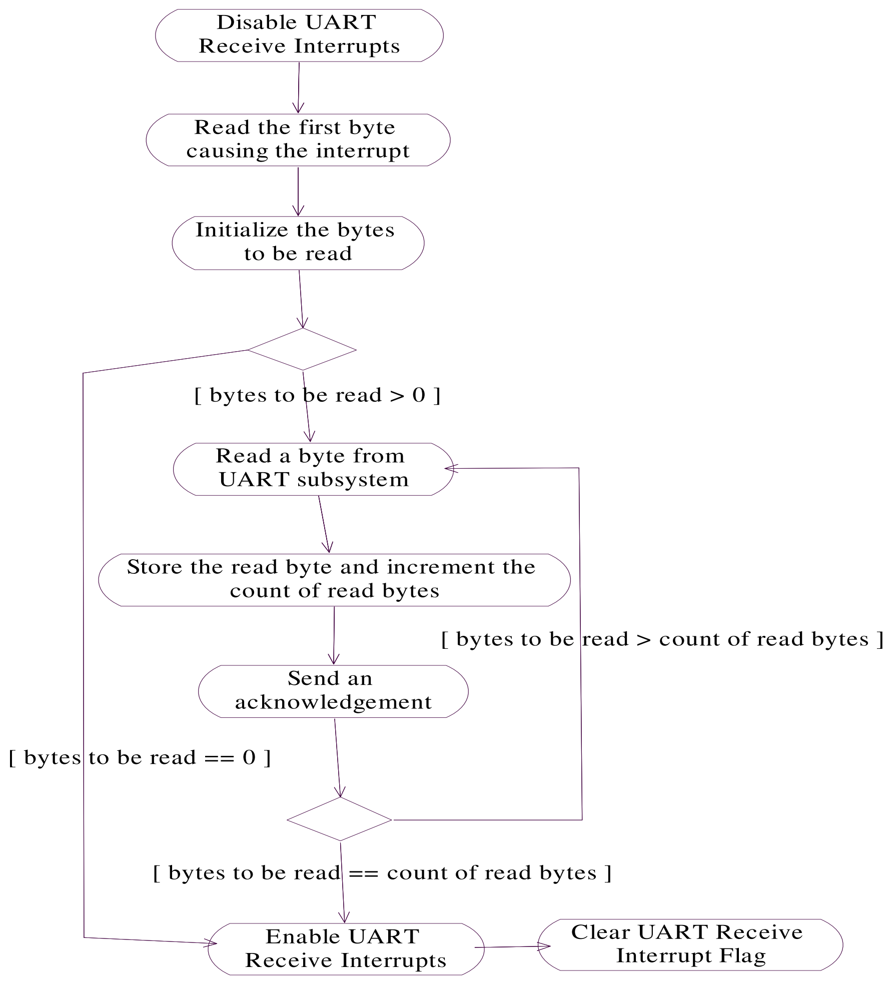

Disable UART Receive Interrupts
Read the first byte causing the interrupt
Initialize the bytes to be read
[bytes to be read > 0]
Read a byte from UART subsystem
Store the read byte and increment the count of read bytes
[bytes to be read > count of read bytes]
Send an acknowledgement
[bytes to be read == 0]
[bytes to be read == count of read bytes]
Enable UART Receive Interrupts
Clear UART Receive Interrupt Flag

Figure 4.4: The workflow for the UART receive interrupt handler for TCP/IP application program

If the first byte received via the UART subsystem is equal to a specific value, then the received bytes are composed of the length and contents of a debug message produced by ZigBee application program to be displayed on the LCD by TCP/IP application program. In such a case, the UART receive interrupt handler displays the incoming debug message on the LCD, if the LCD is not being used by the interrupted section of the main workflow of TCP/IP application program. If the LCD is being used by the interrupted section, the UART receive interrupt handler sends a negative acknowledgement to ZigBee application program after reading the first received byte. Upon receiving the negative acknowledgement, ZigBee application program stops message transfer, waits for a specified amount of time and then tries to resend the debug message to TCP/IP application program.

At the end of the UART receive interrupt handler, the UART receive interrupts are reenabled and the UART receive interrupt flag is cleared.

It is mentioned above that TCP/IP application program performs TCP server operation in its infinite loop during its main workflow. The outline of the workflow of TCP server operation is shown in the below figure.

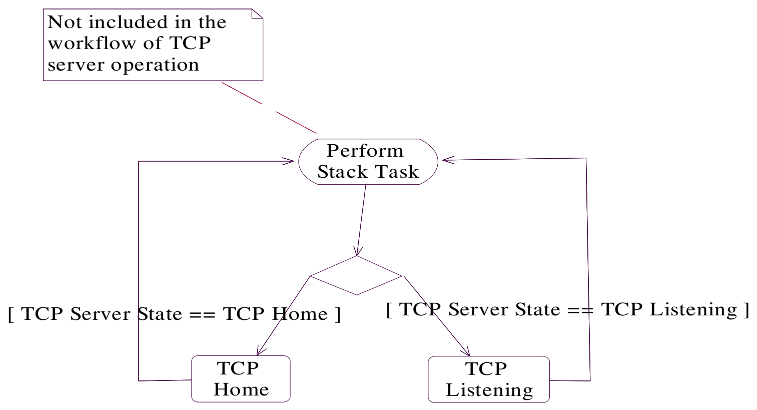
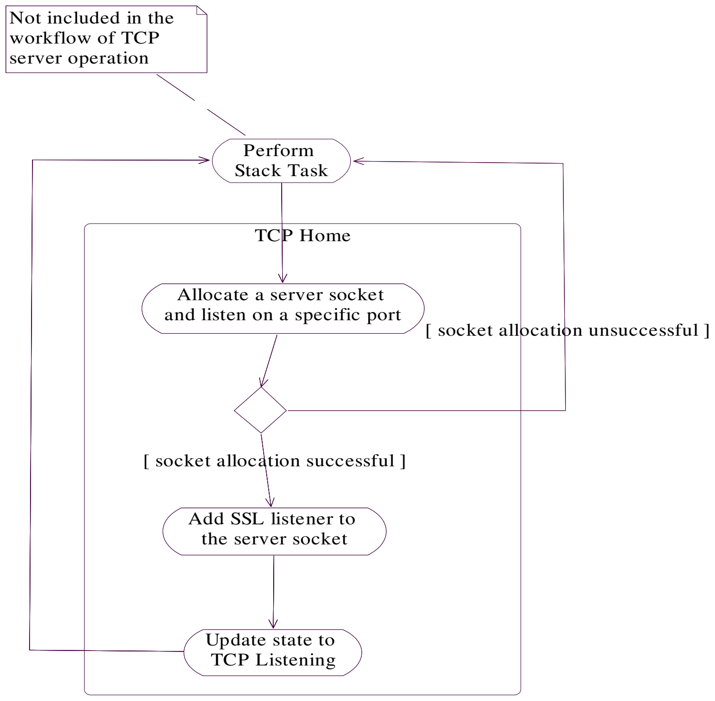

Not included in the workflow of TCP server operation
Perform Stack Task
[TCP Server State == TCP Home]
[TCP Server State == TCP Listening]
TCP Home
TCP Listening

Figure 4.5: The outline of the workflow of TCP server operation performed by TCP/IP application program

During TCP server operation, TCP/IP application program implements a TCP server and that TCP server can be in one of two states. One of the states is called TCP Home state in which a server socket is allocated to listen and accept connections on a specific port. The other state is called TCP Listening state in which TCP/IP application program waits for incoming connections, authenticates and authorizes an incoming connection, and handles data exchange through the authenticated and authorized connection.



Not included in the workflow of TCP server operation
Perform Stack Task
TCP Home
Allocate a server socket and listen on a specific port
[socket allocation unsuccessful]
[socket allocation successful]
Add SSL listener to the server socket
Update state to TCP Listening

Figure 4.6: The operations performed in TCP Home state

During its main workflow, after performing normal stack task, TCP/IP application program goes to TCP Home state or TCP Listening state of the TCP server implemented by itself. The operations performed in TCP Home state are depicted in the above figure. In TCP Home state of the TCP server, TCP/IP application program allocates a TCP server socket listening on a specific port. If the allocation is not successful, the TCP server remains in TCP Home state and TCP/IP application program continues by performing normal stack task. Otherwise, an SSL listener is added to the allocated socket and the socket becomes waiting for incoming SSL client connections. In this case, the state of the TCP server is also updated so that its new state is TCP Listening.

The operations performed in TCP Listening state are depicted in the below figure. On entry into TCP Listening state, TCP/IP application program checks whether a client connects to the TCP server socket (allocated in TCP Home state) via SSL. If no client is connected to the TCP server socket, TCP/IP application program finishes TCP server operation and continues by performing normal stack task. If a client is connected to the TCP server socket, that client is already authenticated via SSL and the actions stated in the following paragraphs are performed.

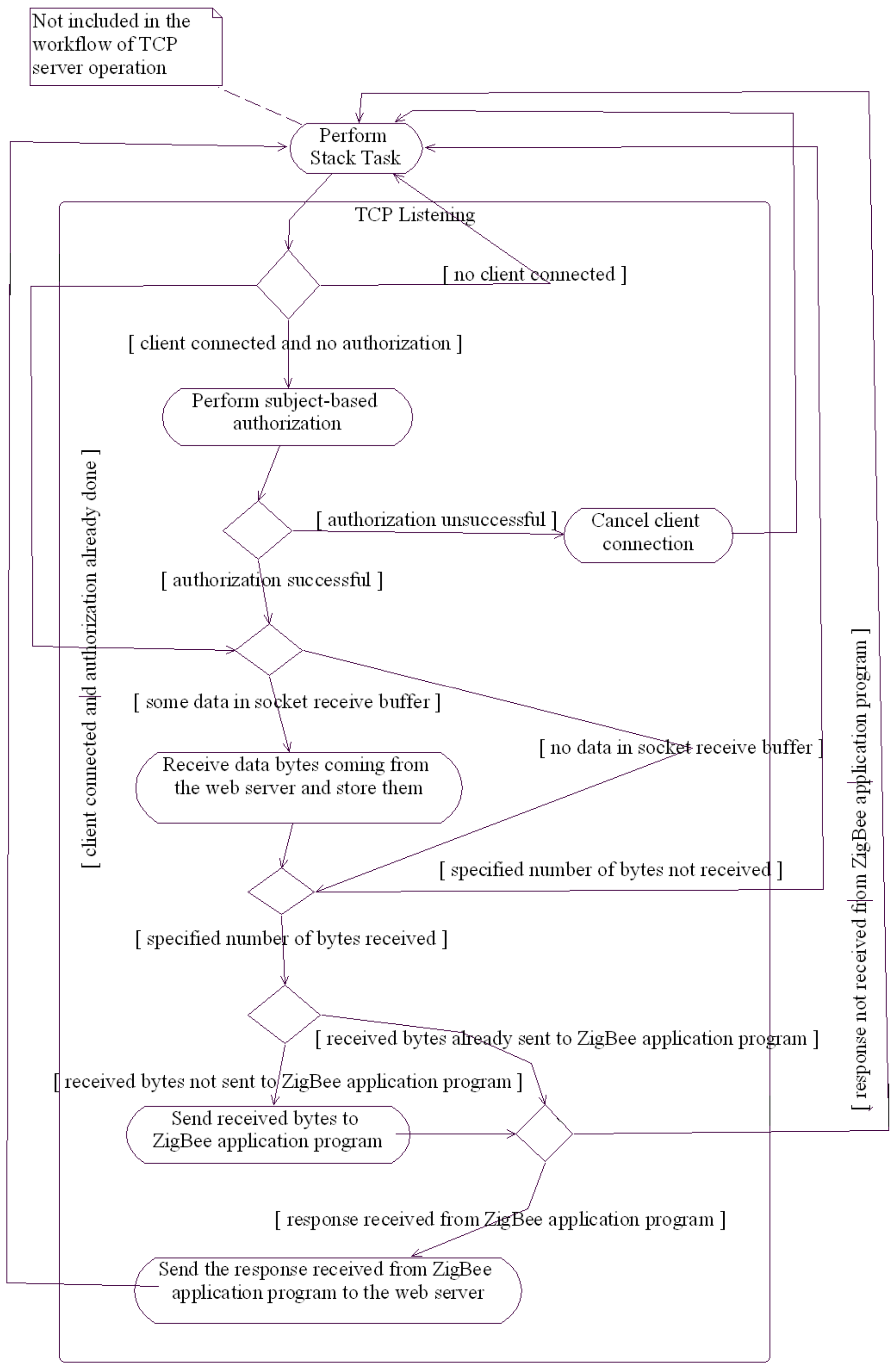

Not included in the workflow of TCP server operation
Perform Stack Task
TCP Listening
[no client connected]
[client connected and no authorization]
Perform subject-based authorization
[authorization unsuccessful]
Cancel client connection
[authorization successful]
[client connected and authorization already done]
[some data in socket receive buffer]
[no data in socket receive buffer]
Receive data bytes coming from the web server and store them
[specified number of bytes not received]
[specified number of bytes received]
[received bytes already sent to ZigBee application program]
[received bytes not sent to ZigBee application program]
Send received bytes to ZigBee application program
[response not received from ZigBee application program]
[response received from ZigBee application program]
Send the response received from ZigBee application program to the web server

Figure 4.7: The operations performed in TCP Listening state

Firstly, it must be determined whether the connected client is authorized to exchange data with TCP/IP application program. If it has been already determined in a previous execution of TCP server operation that the client is authorized to communicate with TCP/IP application program, TCP/IP application program continues its execution as stated in the next paragraph. Otherwise, TCP/IP application program performs a subject-based authorization in which it compares the subject name field of the X509 certificate of the connected client with the identity of the web application implemented in this thesis. Only this web application is authorized to communicate with TCP/IP application program. If the client is not authorized to communicate with TCP/IP application program according to the results of the authorization, TCP/IP application program closes the client connection and continues by performing normal stack task. If the client is authorized to communicate with TCP/IP application program (authorization successful), TCP/IP application program continues its execution as stated in the next paragraph.

TCP/IP application program receives data bytes in the TCP server socket receive buffer if there exists any, stores the received data in a buffer called Ethernet receive buffer and discards the socket receive buffer. Then, TCP/IP application program checks the number of data bytes in Ethernet receive buffer. If all of the bytes of a command are not received from the web server yet (a command from the web server is composed of a specified number of bytes), TCP/IP application program exits TCP server operation and continues by performing normal stack task. Otherwise, TCP/IP application program sends the received command to ZigBee application program if it is not already sent and checks the existence of a response from ZigBee application program. If a response has come, TCP/IP application program sends the response to the web server. After that, TCP/IP application program continues by performing normal stack task.

As can be seen in the above two figures, TCP/IP application program does not wait for anything in any phase of TCP server operation. It enters into the appropriate state of TCP server, performs operations related with that state and then, exits TCP server operation and continues by performing normal stack task. When performing normal stack task, it checks for incoming client connections and incoming data from a connected client. After that, it again enters into the appropriate state of TCP server, processes the incoming data while performing operations related with that state and then, exits TCP server operation and continues by performing normal stack task. It goes on like this.

As an end note, Microchip TCP/IP stack version 4.55, which contains an SSL implementation, does not support client authentication. Namely, it does not support client certificate reception and validation. So, the subject-based client authorization to be performed by TCP/IP application program in TCP Listening state (mentioned above) is not implemented now and remains as an implementation plan to be realized in the future.

4.2.1.2 ZigBee Application Program

ZigBee application program is loaded on the program memory of the PIC18LF4620 microcontroller placed on the 40-pin microcontroller socket on the PICDEM Z motherboard used for the ZigBee master controller. ZigBee application program is responsible for taking commands from TCP/IP application program loaded on the PIC24FJ128GA010 microcontroller of the ZigBee master controller, sending these commands to the ZigBee slave devices via the ZigBee wireless communication protocol, taking responses to these commands from the ZigBee slave devices via the ZigBee protocol and sending these responses to TCP/IP application program. The communication between TCP/IP application program and ZigBee application program is performed via UART communication mechanism supported by the EUSART module found in both the PIC18LF4620 and PIC24FJ128GA010 microcontrollers of the ZigBee master controller. ZigBee application program uses the Microchip Stack for the ZigBee Protocol to perform

ZigBee communication between the ZigBee master controller and the ZigBee slave devices.

ZigBee application program handles the conversion of messages between the formats of the two protocols: specific application protocol working over TCP/IP and the ZigBee protocol. A command coming from TCP/IP application program to ZigBee application program is compliant with the specific application protocol working over TCP/IP. Upon receiving the command, ZigBee application program processes the command and converts the command into a ZigBee message to be sent to the ZigBee slave device related with the command.

After sending the resulting ZigBee message to the ZigBee slave device, ZigBee application program waits for a response from the ZigBee slave device. The response to be received from the ZigBee slave device is compliant with the ZigBee protocol. After receiving the response, ZigBee application program processes the response, converts the response into a form compliant with the specific application protocol and sends it to TCP/IP application program.

In the context of this thesis work, one source file is produced for the implementation of ZigBee application program. The name of the file is ZigBeeMasterController2.c. The important data structures and algorithms defined in this file are explained in the following paragraphs of this section. In addition to this file, UART.c and UART.h files containing UART C function definitions for PIC18 microcontrollers, and Delay.c and Delay.h files containing delay function definitions for PIC18 microcontrollers are added to the application project from the Microchip TCP/IP stack. Also, three files are generated by the ZENA Network Analyzer developed by the Microchip company. These three files are myZigBee.c, zigbee.def (export definition file) and zLink.lkr (linker script).

The most important data structure defined in ZigBee application program is the one which defines a ZigBee node connected to the network coordinated by the

ZigBee master controller. This data structure is called as ZigBee node structure. An array of this data structure is defined to hold information about all the ZigBee nodes connected to the ZigBee master controller. The size of this array is defined to be 10 by ZigBee application program to avoid memory overrun. This array can be called the ZigBee master controller's database of networked ZigBee nodes.

The details of ZigBee node structure is shown in the below figure. ZigBee node structure contains the short and extended address of the ZigBee node it describes. ZigBee application program collects this address information when the ZigBee node joins to the network coordinated by the ZigBee master controller.

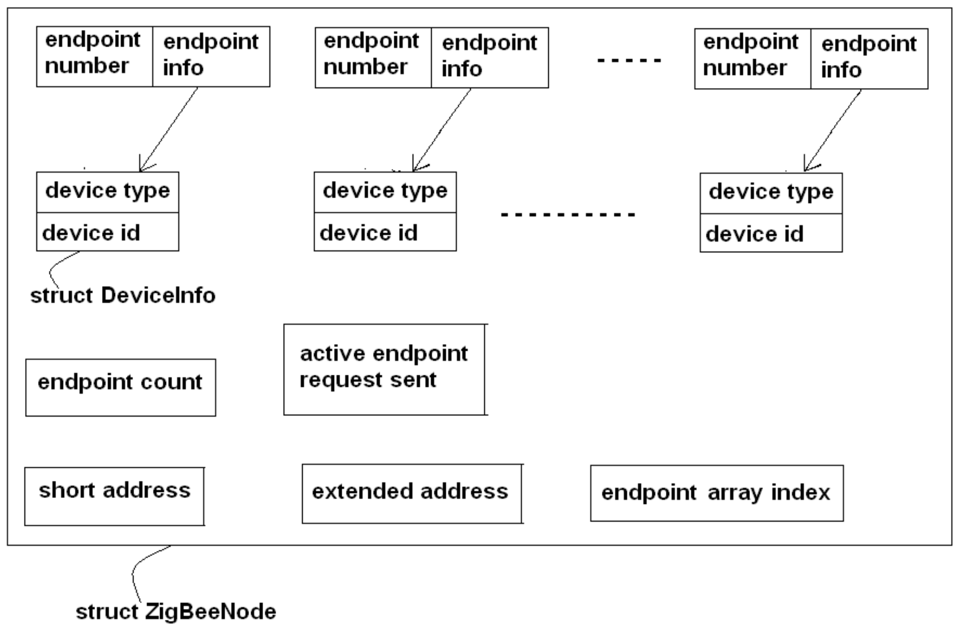


Figure 4.8: The details of ZigBee node structure defined by ZigBee application program

ZigBee node structure contains the active endpoint count and active endpoint numbers of the ZigBee node it describes. ZigBee application program collects the active endpoint count of the ZigBee node by sending active endpoint request to it after it is joined to the network. After receiving active endpoint response from the ZigBee node, ZigBee application program sends a simple descriptor request to the ZigBee node for each active endpoint of the ZigBee node. With these simple descriptor request messages, ZigBee application program collects the device type of each ZigBee slave device residing on each active endpoint of the ZigBee node. ZigBee application program also assigns a device identifier to each ZigBee slave device. The device type and device identifier of each ZigBee slave device of the ZigBee node are also stored in ZigBee node structure.

ZigBee node structure also contains a boolean data field (active endpoint request sent) which determines whether or not ZigBee application program sends an active endpoint request to the ZigBee node described by the structure. This structure also contains a data field (endpoint array index) which determines the number of the active endpoint, of the ZigBee node, for which ZigBee application program sends a simple descriptor request last.

The other important data structures and variables defined by ZigBee application program are shown in the below figure. The ZigBee message information structure shown in the below figure contains information about the ZigBee message prepared by ZigBee application program. This structure contains the following information:

- profile identifier of the ZigBee message
- cluster number of the attributes contained in the ZigBee message
- short and extended address of the ZigBee node containing the ZigBee slave device targeted by the ZigBee message
- endpoint number of the ZigBee slave device targeted by the ZigBee message
- the time period in milliseconds during which ZigBee application program waits for a ZigBee response from the ZigBee slave device after sending the ZigBee message (timeout period)

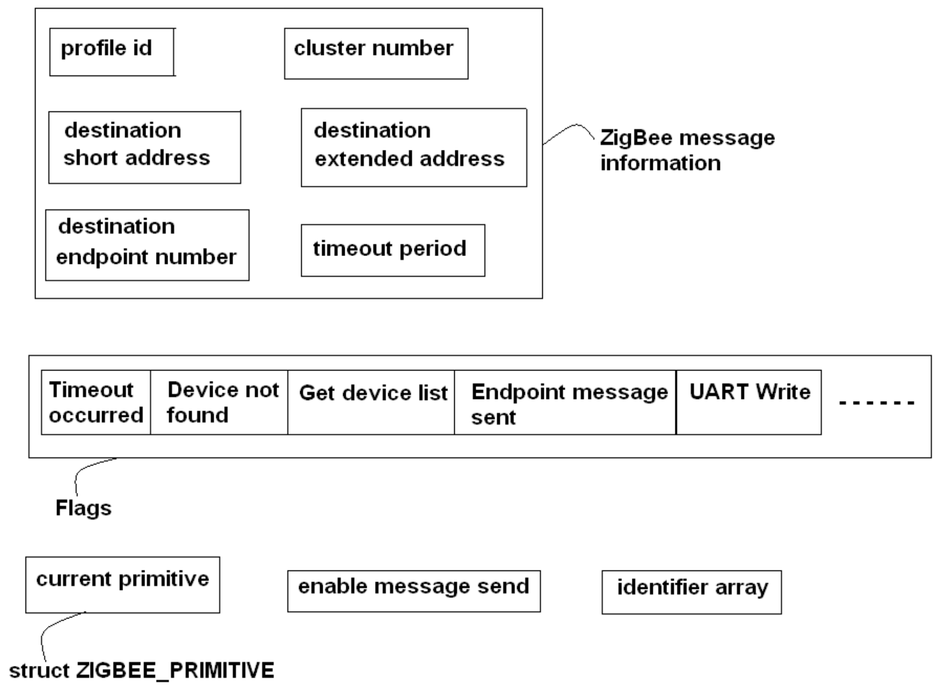


Figure 4.9: Important data structures and variables defined by ZigBee application program

ZigBee application program defines a Flags structure containing flags indicating the occurrence of specific conditions during the execution of ZigBee application program. Some of the flags defined are:

- Timeout occurred indicating that the ZigBee response message did not come from the ZigBee slave device in a specified time interval
- Device not found indicating that the ZigBee slave device targeted by the message coming from TCP/IP application program is not found in the ZigBee master controller's database of networked ZigBee slave devices
- Get device list indicating that the message coming from TCP/IP application program contains a get device list command which aims to get the list of current ZigBee slave devices connected to the ZigBee master controller

- Endpoint message sent indicating that an active endpoint request or simple descriptor request is sent to a ZigBee node and a response message has not come yet from the ZigBee node
- UART Write indicating that ZigBee application program is in the process of writing to the UART module of the PIC18LF4620 microcontroller on which it resides

ZigBee application program stores the current ZigBee primitive to be executed in a specific variable. This variable is used in the operation of performing normal ZigBee stack task. After performing ZigBee stack task, the value of the current primitive to be executed may change.

ZigBee application program only sends a new ZigBee message to a ZigBee slave device after it receives a message from TCP/IP application program via UART mechanism. After receiving a message from TCP/IP application program, ZigBee application program enable ZigBee message sending by setting a specific variable (enable message send).

When a new ZigBee slave device joins to the network coordinated by the ZigBee master controller, ZigBee application program tries to extract its device type information by sending simple descriptor request to the ZigBee node containing the ZigBee slave device. After getting its device type, ZigBee application program assigns a device identifier to the ZigBee slave device. ZigBee application program assigns the next positive integer available for the device type of the ZigBee slave device, to the ZigBee slave device as its device identifier. The next available positive integer for each device type is stored in the identifier array.

The general operation of ZigBee application program is shown in the below figure. There is an initialization section at the beginning of ZigBee application program.

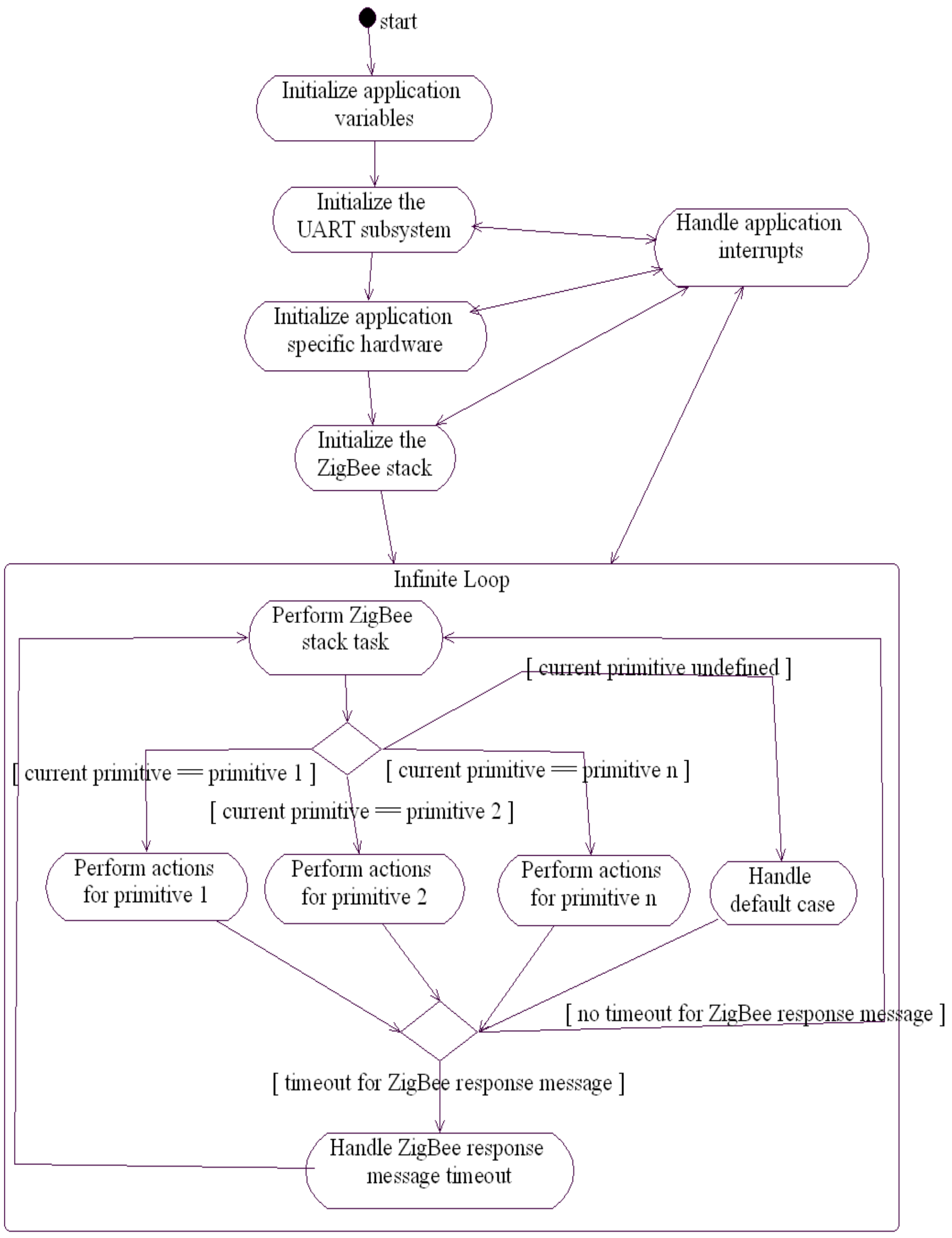

start
Initialize application variables
Initialize the UART subsystem
Handle application interrupts
Initialize application specific hardware
Initialize the ZigBee stack
Infinite Loop
Perform ZigBee stack task
[current primitive undefined]
[current primitive == primitive 1]
[current primitive == primitive n]
[current primitive == primitive 2]
Perform actions for primitive 1
Perform actions for primitive 2
Perform actions for primitive n
Handle default case
[no timeout for ZigBee response message]
[timeout for ZigBee response message]
Handle ZigBee response message timeout

Figure 4.10: General operation of ZigBee application program

ZigBee application program initializes current primitive to be executed to no primitive, flags to all zeros and identifier array to all ones at the start of the initialization section. In initializing the UART subsystem, the registers of the PIC18LF4620 microcontroller controlling UART operation are initialized to appropriate values. Some of these registers are assigned values depending on the baud rate used (19200). In initializing the application-specific hardware, ZigBee application program initializes the pins, of the PIC18LF4620 microcontroller, used to communicate with and control the operation of the ZigBee RF Card. It also initializes the registers, of the PIC18LF4620 microcontroller, used to control the external interrupt mechanism of the PIC18LF4620. This mechanism is used to receive interrupts from the ZigBee RF Card. At the end of the initialization section, the core layers of ZigBee stack are initialized.

After the initialization section, there is an infinite loop which starts by performing normal ZigBee stack task which includes executing the current primitive, checking for incoming packets, determining the primitive to be handled by the application and updating the current primitive with the primitive to be handled by the application. After that in the infinite loop, ZigBee application program performs specific actions for the current primitive generated by the ZigBee stack of the ZigBee master controller. There is a primitive for the result of a network formation request made by ZigBee application program, the indication of a new ZigBee slave device's joining to the network, the indication of a ZigBee slave device's leaving from the network, the reception of data by the ZigBee stack from a ZigBee slave device, the result of transmission of data to a ZigBee slave device etc..

In the infinite loop, ZigBee application program may send a ZigBee message to a ZigBee slave device. Just after sending the ZigBee message, ZigBee application program starts a timer for the ZigBee response message that will come from the ZigBee slave device. At the end of the infinite loop, it is controlled whether there is such a timer started. If there is such a timer started, it is also controlled whether the timer has expired or not. If the timer has expired, ZigBee application program sends a message, indicating timeout condition for a ZigBee response message, to TCP/IP application program.

Lastly, there is an interrupt handler routine provided by the ZigBee stack and implemented to handle application-specific interrupts (such as UART receive interrupts) by ZigBee application program. In this routine, ZigBee application program mainly handles UART receive interrupts occurred when TCP/IP application program sends data via UART mechanism to ZigBee application program. ZigBee application program receives a message compliant with the specific application protocol from TCP/IP application program and after receiving such a message, it enables ZigBee message sending.

When a UART receive interrupt occurs, the interrupt handler routine disables further UART receive interrupts. After reading each byte by polling UART receive register, it sends an acknowledgement byte to TCP/IP application program. Before the routine returns, it reenables UART receive interrupts. During the UART receive interrupt handling, UART receive interrupts are disabled so that further bytes received via UART do not cause a UART receive interrupt in ZigBee application program. ZigBee application program reads the received bytes by polling.

In the infinite loop within ZigBee application program, ZigBee application program performs specific actions according to the current ZigBee primitive generated internally by the ZigBee stack. The actions performed by ZigBee application program for specific primitives are described in the remainder of this section.

In the initialization section, ZigBee application program sets the current primitive to no primitive. After that, in the infinite loop a normal ZigBee stack task is performed. In the infinite loop, ZigBee application program performs the actions shown in the below figure, if the current primitive equals no primitive after performing stack task. ZigBee application program starts no primitive handling by checking whether the ZigBee master controller has already formed a ZigBee network or not. If it has not formed a network yet and it is not in the middle of forming a network, ZigBee application program requests for network formation from the ZigBee stack by issuing a specific ZigBee primitive to the stack after filling the parameters of this primitive appropriately.

If the ZigBee master controller has already formed network, ZigBee application program continues as follows. It checks whether the ZigBee stack is ready to transmit or not. If the stack is not ready to transmit, ZigBee application program exits from the current iteration of the infinite loop. Otherwise, it tries to perform application specific tasks that can transmit ZigBee messages as follows.

ZigBee application program requests the ZigBee stack to send active endpoint or simple descriptor request message to a ZigBee node connected to the network coordinated by the ZigBee master controller, if it has received a response to the previous such request message sent to any ZigBee node. Which request message is sent to which ZigBee node is determined as follows by ZigBee application program. ZigBee application program stores a list of networked ZigBee nodes with their endpoint information. The nodes are stored in network joining order. ZigBee application program traverses the list from the start. If ZigBee application program has not sent an active endpoint request to a node being traversed, it requests the ZigBee stack to send an active endpoint request to that node. If it has already sent active endpoint request to that node but it has not sent a simple descriptor request for each endpoint of that node, it requests the ZigBee stack to send a simple descriptor request for an endpoint of that node which has the lowest number and has not received a simple descriptor request yet. If it has sent all the required active endpoint and simple descriptor request to that node, it passes to the next node in the list and performs the same checks.

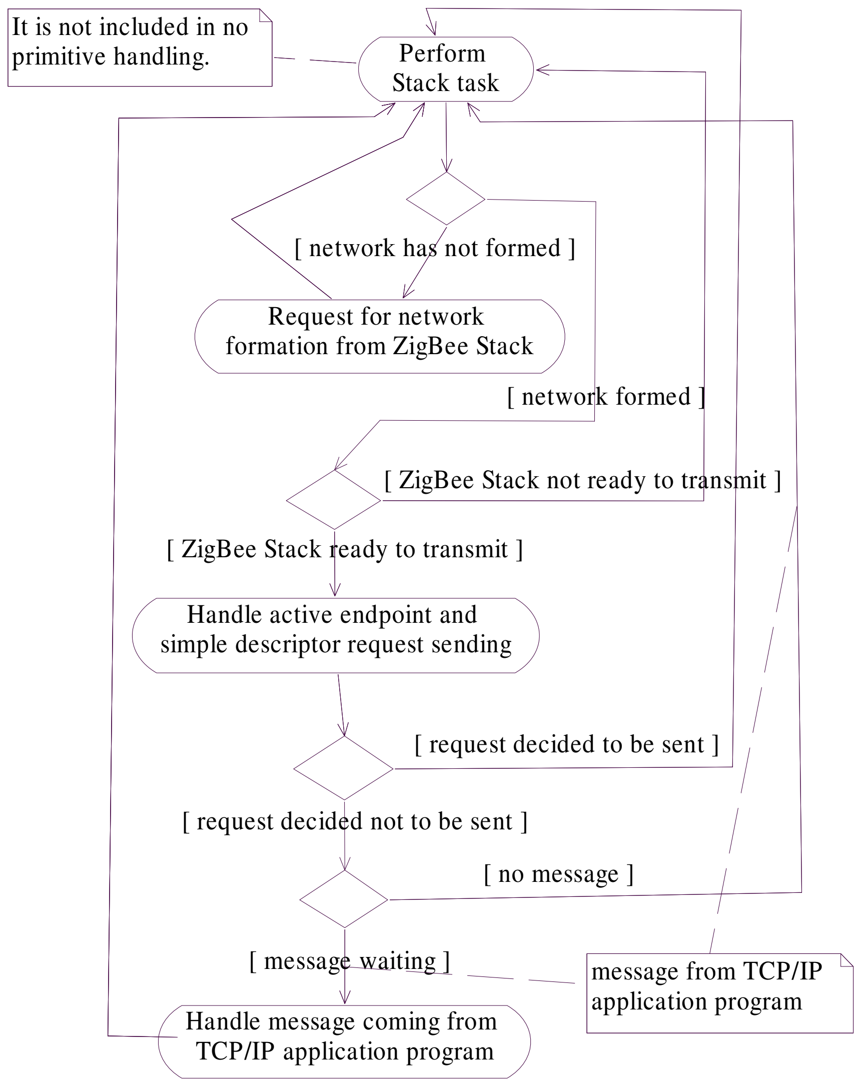

It is not included in no primitive handling.
Perform Stack task
[network has not formed]
Request for network formation from ZigBee Stack
[network formed]
[ZigBee Stack not ready to transmit]
[ZigBee Stack ready to transmit]
Handle active endpoint and simple descriptor request sending
[request decided to be sent]
[request decided not to be sent]
[no message]
[message waiting]
message from TCP/IP application program
Handle message coming from TCP/IP application program

Figure 4.11: The actions performed in the no primitive case for ZigBee application program

After requesting the ZigBee stack to send an active endpoint or simple descriptor request message to a ZigBee node, ZigBee application program exits from the current iteration of the infinite loop. It is vital to understand the issue that ZigBee application program requests the ZigBee stack to send an active endpoint or simple descriptor request message to a node only if it has received a response to the previous such message sent to any node. ZigBee application program requests the ZigBee stack to send at most one active endpoint or simple descriptor request message in an iteration of the infinite loop.

ZigBee application program may not request the ZigBee stack to send an active endpoint or simple descriptor request message to any ZigBee node in an iteration of the infinite loop because of any of the following three reasons:

- there is no ZigBee node connected to the network
- ZigBee application program has not received a response to the previous active endpoint or simple descriptor request message
- ZigBee application program has received responses to all the required request messages from all the ZigBee nodes

If ZigBee application program has not send an active endpoint or simple descriptor request message in an iteration of the infinite loop, it checks whether there is a message coming from TCP/IP application program and waiting to be processed. If there is no such message, ZigBee application program exits from current iteration of the infinite loop. Otherwise, it processes the received message which is compliant with the specific application protocol.

In processing the message received from TCP/IP application program, ZigBee application program first checks whether the device type and device identifier fields of the received message determines a valid ZigBee slave device connected to the network. (The device type value of “all” is also accepted for general purpose commands. It does not identify a specific ZigBee slave device.) If the ZigBee slave device targeted by the message is not found in the ZigBee master controller’s database of networked devices, a response indicating a wrong device is sent to TCP/IP application program. Otherwise, the following actions are performed.

If the message received from TCP/IP application program conveys a general purpose get device list command, ZigBee application program sends a list of ZigBee slave devices, connected to the network, to TCP/IP application program.

If the message received from TCP/IP application program contains attribute set or get commands, ZigBee application program prepares a ZigBee message containing the attribute set or get transactions corresponding to the attribute set or get commands in the received message. Then, ZigBee application program determines a timeout period for a ZigBee response message based on the number of attribute set or get commands in the received message, starts a timer, sends the prepared ZigBee message to the target ZigBee slave device (determined from the received message) and exits from the current iteration of the infinite loop. This finishes the description of the actions performed when the current primitive equals to no primitive.

Another important part of ZigBee application program is the handling of a ZigBee response message, received from a ZigBee slave device, in the infinite loop. The reception of a ZigBee response message is indicated to ZigBee application program by a specific primitive generated and assigned to the current primitive by the ZigBee stack. The actions performed in handling of a ZigBee response message in the infinite loop are shown in the below figure.

Handling of a ZigBee response message by ZigBee application program starts by checking the endpoint of the ZigBee master controller the ZigBee response message is targeted to, namely the destination endpoint of the ZigBee response message.

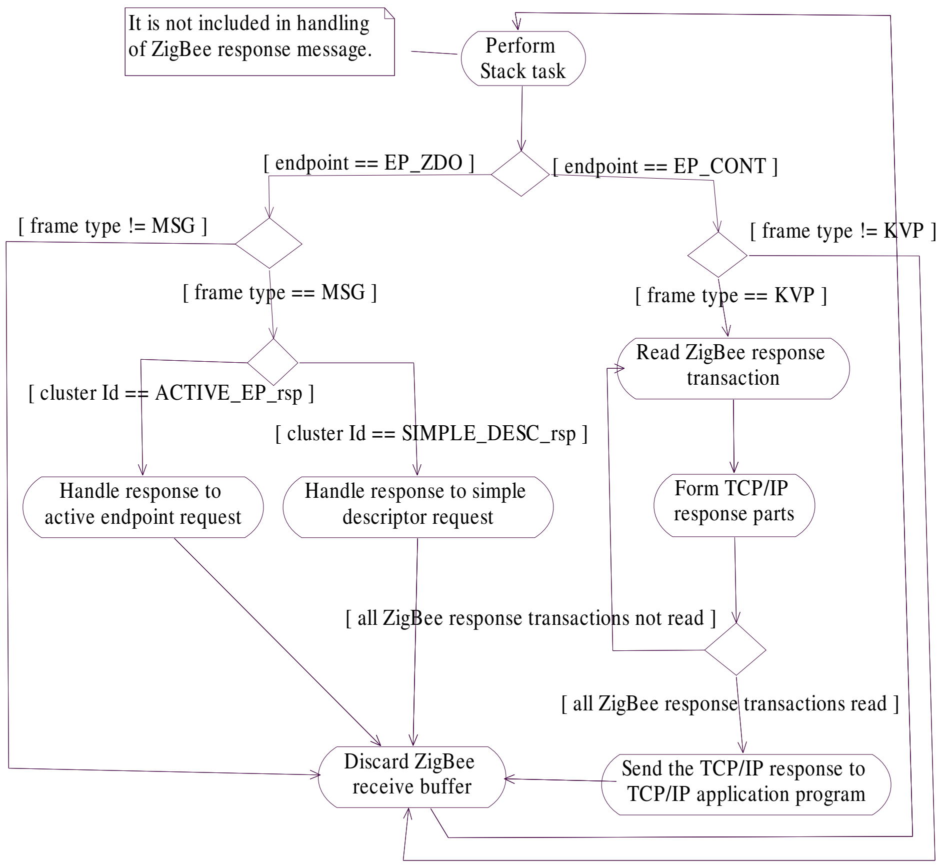

It is not included in handling of ZigBee response message.
Perform Stack task
[endpoint == EP_ZDO]
[endpoint == EP_CONT]
[frame type != MSG]
[frame type == MSG]
[frame type != KVP]
[frame type == KVP]
[cluster Id == ACTIVE_EP_rsp]
[cluster Id == SIMPLE_DESC_rsp]
Read ZigBee response transaction
Handle response to active endpoint request
Handle response to simple descriptor request
Form TCP/IP response parts
[all ZigBee response transactions not read]
[all ZigBee response transactions read]
Discard ZigBee receive buffer
Send the TCP/IP response to TCP/IP application program

Figure 4.12: The actions performed in handling a ZigBee response message in ZigBee application program

If the destination endpoint of the ZigBee response message equals to EP_ZDO, the ZigBee response message is targeted to the ZDO of the ZigBee master controller. In this case, if the frame type of the ZigBee response message is not MSG, ZigBee application program discards the ZigBee receive buffer and exits from the current iteration of the infinite loop. Otherwise, the cluster identifier of the ZigBee response message is checked.

If the cluster identifier of the received ZigBee response message equals to ACTIVE_EP_rsp, then the ZigBee response message is a response to an active endpoint request message sent to a ZigBee node. In this case, the active endpoints in the ZigBee node (the endpoint numbers of ZigBee slave devices residing on that node) are extracted from the ZigBee response message and saved to a record, related with that ZigBee node, in ZigBee master controller's database of networked devices. Then, ZigBee application program discards the ZigBee receive buffer and exits from the current iteration of the infinite loop.

If the cluster identifier of the received ZigBee response message equals to SIMPLE_DESC_rsp, then the ZigBee response message is a response to a simple descriptor request message sent to a ZigBee node for a specific endpoint. In this case, the device type of the ZigBee slave device residing on this specific endpoint of the ZigBee node is extracted from the ZigBee response message and saved to a record, related with the ZigBee node, in the ZigBee master controller's database of networked devices. Then, ZigBee application program discards the ZigBee receive buffer and exits from the current iteration of the infinite loop.

If the destination endpoint of the ZigBee response message equals to EP_CONT, the ZigBee response message is targeted to the user-defined endpoint of the ZigBee master controller. The ZigBee master controller has one user-defined endpoint. In this case, if the frame type of the ZigBee response message is not KVP, ZigBee application program discards the ZigBee receive buffer and exits from the current iteration of the infinite loop. Otherwise, each transaction in the ZigBee response message is processed as follows.

Firstly, whether the transaction contains the value of an attribute of a ZigBee slave device or the result of setting an attribute of a ZigBee slave device is determined. If the transaction contains the value of an attribute of a ZigBee slave device, this value is converted to a string and appended to the buffer containing the TCP/IP response (the response which will be sent to TCP/IP application program). If the transaction contains the result of setting an attribute of a ZigBee slave device to a specific value, the string "Yes" or "No" is appended to the TCP/IP response buffer according to this result.

After processing all the transactions in the ZigBee response message, the TCP/IP response message formed is sent to TCP/IP application program. Then, ZigBee application program discards the ZigBee receive buffer and exits from the current iteration of the infinite loop. This finishes the handling of a ZigBee response message by ZigBee application program.

Another important sequence of actions performed in the infinite loop of ZigBee application program handle the joining of a ZigBee node to the network coordinated by the ZigBee master controller. The joining of a new ZigBee node to the network is indicated to ZigBee application program by a specific primitive generated and assigned to the current primitive by the ZigBee stack. The actions performed in handling of a newly joined ZigBee node in the infinite loop are shown in the below figure.

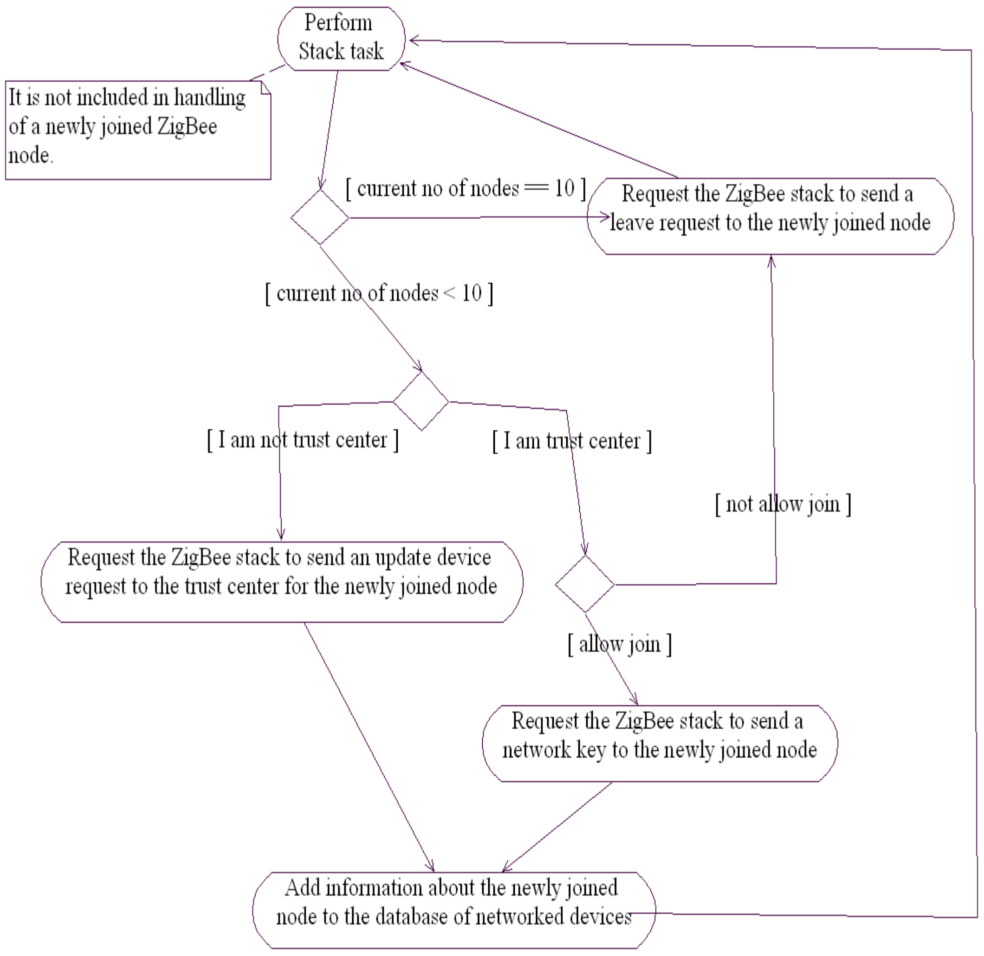

Perform Stack task
It is not included in handling of a newly joined ZigBee node.
[current no of nodes == 10]
Request the ZigBee stack to send a leave request to the newly joined node
[current no of nodes < 10]
[I am not trust center]
[I am trust center]
[not allow join]
Request the ZigBee stack to send an update device request to the trust center for the newly joined node
[allow join]
Request the ZigBee stack to send a network key to the newly joined node
Add information about the newly joined node to the database of networked devices

Figure 4.13: The actions performed in handling a newly joined ZigBee node in ZigBee application program

The number of ZigBee nodes that can connect to the network coordinated by the ZigBee master controller is limited to 10. The reason behind this limitation is avoiding memory overrun resulting from storing information about too many nodes in the ZigBee master controller's database of networked devices. Handling of a newly joined ZigBee node starts by checking the current number of nodes connected to the network. If this number is 10, ZigBee application program requests the ZigBee stack to send a leave request to the newly joined node and exits from the current iteration of the infinite loop. Otherwise, the following actions are performed.

If the ZigBee master controller is the trust center of the network (it is the trust center according to the current configuration of the network), ZigBee application program must decide if it can allow the newly joined node to be added to the network. This decision may depend on security requirements of the network. In ZigBee application program's current configuration, if the newly joined node has joined to the network using secure join, namely if the underlying network association is secure, then the newly joined node is allowed. Otherwise, it is not allowed to be added to the network. However, in other configurations, the decision to add the newly joined device to the network may depend on other conditions. If the newly joined node is not allowed to be added to the network, ZigBee application program requests the ZigBee stack to send a leave request to the newly joined node and exits from the current iteration of the infinite loop. Otherwise, the following actions are performed.

ZigBee application program requests the ZigBee stack to send a network key to the newly joined node. If the newly joined node has joined to the network using secure join, all zeros are sent to the newly joined node. (In this case, the actual network key is not all zeros and is preconfigured in both the ZigBee master controller and the newly joined node.) If the newly joined node has joined to the network not using secure join, a network key stored by the ZigBee stack of the ZigBee master controller must be sent to the newly joined node unprotected. This is a security leak. So as mentioned in the above paragraph, ZigBee application program does not allow the newly joined node to be added to the network if it has not joined to the network using secure join. After requesting the ZigBee stack to send all zeros or the actual network key to the newly joined node, ZigBee application program adds information about the newly joined node (such as short address and extended address of the newly joined node) to its database of networked devices and exits from the current iteration of the infinite loop.

If the ZigBee master controller is not the trust center of the network, ZigBee application program requests the ZigBee stack to send an update device command to the trust center of the network for the newly joined node. This update device command contains information about the newly joined node such as its short and extended address, an indication of whether it has joined to the network using secure join or not etc.. After sending the update device command, ZigBee application program adds information about the newly joined node to its database of networked devices and exits from the current iteration of the infinite loop. This finishes the handling of a newly joined node by ZigBee application program in the infinite loop.

4.2.2 ZigBee Slave Device Software

In this thesis work, there are two ZigBee slave devices implemented. The first ZigBee slave device implements the logic of light defined in the Home Control, Lighting Profile. The second ZigBee slave device implements the logic of occupancy sensor defined in the Home Control, Lighting Profile. A ZigBee slave device receives commands from the ZigBee master controller and processes these commands according to the attributes that these commands refer to. After processing these commands, the ZigBee slave device sends the appropriate responses to the ZigBee master controller. Each of the ZigBee slave devices uses Microchip Stack for the ZigBee Protocol as its ZigBee stack.

The application program for a ZigBee slave device is generally called slave application program throughout the thesis. The application program for the light is specifically called light slave application program and the application program for the occupancy sensor is specifically called occupancy sensor slave application program throughout the thesis. The following section describes the details of slave application program of a ZigBee slave device referring to the specific slave application programs where necessary.

4.2.2.1 Slave Application Program

Slave application program is loaded on the program memory of the PIC18LF4620 microcontroller on a ZigBee slave device. One file is produced for the slave application program for the light. This is Slave1Light.c. Also, one file is produced for the slave application program for the occupancy sensor. This is Slave2OS.c. In addition to appropriate one of these files, seven files are added to each ZigBee slave device application project. These seven files are UART.c, UART.h, Delay.c, Delay.h, myZigBee.c, zigbee.def, zLink.lkr.

The files UART.c and UART.h are taken from the Microchip TCP/IP Stack and contain the definitions and declarations of UART-related routines that are used to initialize the UART module on the PIC18LF4620 microcontroller (on PICDEM Z motherboard on each ZigBee slave device) and write/read some data to/from this UART module. Slave application program uses this UART module to send debug messages to a PC connected to PICDEM Z motherboard on the ZigBee slave device. The PC is connected to PICDEM Z motherboard on the ZigBee slave device via an RS-232 serial cable. The cable connects the PC’s COM port to the RS-232 port of PICDEM Z motherboard. The PC also contains a serial port monitor software to fetch and display the data sent to its COM port by the ZigBee slave device. The PC is used to display debug messages produced by slave application program.

The files Delay.c and Delay.h are taken from the Microchip TCP/IP Stack and include the definitions and declarations of general delay routines. The files myZigBee.c, zigbee.def and zLink.lkr are generated by the ZENA Network Analyzer software. The files myZigBee.c and zigbee.def contain declarations of some parameters and constants determining the role and characteristics of the ZigBee slave device, these files are related with, in a ZigBee network. The file zLink.lkr is a linker script for the PIC18LF4620 microcontroller used to implement the ZigBee slave device.

There are three important data variables defined by slave application program. Slave application program defines a variable called network descriptor list pointing to a link list containing descriptions of available networks that the ZigBee slave device can join. The information about available networks is collected by performing a network discovery process. There is a variable called current network descriptor pointing to a description of the network selected by the ZigBee slave device for joining. There is also a variable called current primitive that stores the primitive to be executed next by the Microchip ZigBee Stack, or generated last by the Stack via the interrupt mechanism, or the constant NO_PRIMITIVE. In addition to these three variables, there are application variables defined specifically by each of the specific slave application programs.

The main workflow of slave application program is very similar to the main workflow of ZigBee application program of the ZigBee master controller. There is an initialization section at the beginning in which application variables, UART subsystem of the PIC18LF4620 microcontroller, application specific hardware and the ZigBee stack are initialized. After the initialization section, there is an infinite loop in which slave application program performs normal ZigBee stack task and then performs specific actions according to the value of current primitive.

Additionally, there is a user interrupt handler that can be called asynchronously to the main workflow of slave application program and handles application interrupts. This interrupt handler mainly handles PORTB change interrupts. When RB4 button on PICDEM Z motherboard in a ZigBee slave node is pressed by the user when the node is part of the network coordinated by the ZigBee master controller, this interrupt handler adjusts some flags so that slave application program requests the ZigBee stack to leave from the network. When RB4 button on the node is pressed again by the user after the node leaves the network, the interrupt handler adjusts some flags so that slave application program requests the ZigBee stack to perform network discovery.

In the remainder of this section, the actions performed by slave application program in the infinite loop for specific values of current primitive are described. The below figure shows the actions performed in the infinite loop when current primitive is equal to no primitive.

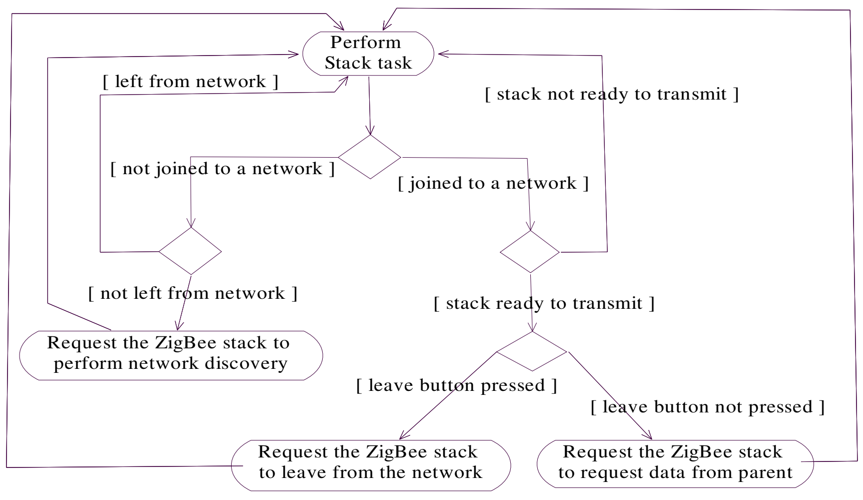

Perform Stack task
[left from network]
[stack not ready to transmit]
[not joined to a network]
[joined to a network]
[not left from network]
[stack ready to transmit]
Request the ZigBee stack to perform network discovery
[leave button pressed]
[leave button not pressed]
Request the ZigBee stack to leave from the network
Request the ZigBee stack to request data from parent

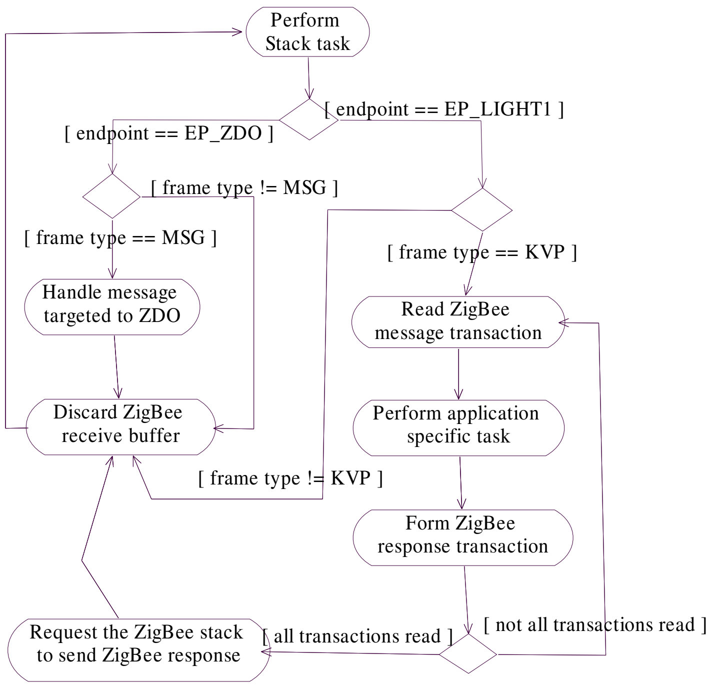
Figure 4.14: Handling of no primitive in the infinite loop in slave application program

Performing stack task shown in the above figure is not included in the actions performed in handling no primitive case by slave application program. Slave application program starts handling no primitive case by checking whether the ZigBee node is joined to a network or not. If the ZigBee node is not joined to a network now and has not left from a network before in the current execution of the node, then slave application program requests the ZigBee stack to perform network discovery to discover available networks for joining. If the ZigBee node is not joined to a network now and has left from a network before (with the user pressing RB4 button on the node) in the current execution of the node, then slave application program exits from the current iteration of the infinite loop without doing anything. If the ZigBee node is joined to a network now, the following actions are performed.

Slave application program checks if the stack is ready to transmit data. If it is not ready to transmit data, slave application program exits from the current iteration of the infinite loop. If the stack is ready to transmit data and leave button (RB4 button) has been already pressed, slave application program requests the stack to leave from network and exits from the current iteration of the infinite loop. If the stack is ready to transmit data and leave button has not been pressed, slave application program requests the stack to retrieve data from its parent (ZigBee master controller) to perform synchronization with its parent, and exits from the current iteration of the infinite loop. This finishes the handling of no primitive case in the infinite loop.

Another important algorithm in the infinite loop is the one handling the ZigBee message received from the ZigBee master controller. Upon reception of a new ZigBee message by performing stack task, current primitive is set to a specific primitive indicating the reception of a ZigBee message. The actions performed by slave application program in the infinite loop after reception of a new ZigBee message are shown in the below figure.

Slave application program checks the ZigBee node endpoint to which the received ZigBee message is targeted. If the target endpoint of the received message is ZDO, slave application program checks the frame type of the received message. If the frame type of the received message is not MSG, the receive buffer of the ZigBee stack is discarded and the current iteration of the infinite loop is exited. If the frame type of the received message is MSG, slave application program handles the received message according to its cluster identifier, discards the receive buffer of the stack and exits from the current iteration of the infinite loop.



Perform Stack task
[endpoint == EP_LIGHT1]
[endpoint == EP_ZDO]
[frame type != MSG]
[frame type == MSG]
[frame type == KVP]
Handle message targeted to ZDO
Read ZigBee message transaction
Discard ZigBee receive buffer
Perform application specific task
[frame type != KVP]
Form ZigBee response transaction
Request the ZigBee stack to send ZigBee response
[all transactions read]
[not all transactions read]

Figure 4.15: The actions performed by slave application program after receiving a new ZigBee message

If the target endpoint of the received ZigBee message is EP_LIGHT1, the identifier for the only user-defined endpoint, then slave application program checks the frame type of the received message. If the frame type of the received message is not KVP, slave application program discards the receive buffer of the stack and exits from the current iteration of the infinite loop. If the frame type of the received message is KVP, slave application program reads the transactions of the received ZigBee message one by one and performs the following actions for each transaction as follows.

Slave application program performs application specific tasks and forms a ZigBee response transaction according to the cluster identifier of the received ZigBee message, the identifier of the attribute contained in the received transaction and the command contained in the received transaction. For example, when light slave application program receives a ZigBee message with a specific cluster identifier and containing a transaction which contains a specific attribute and a command setting this attribute to a value indicating turn on, light slave application program turns on one of the LEDs on the PICDEM Z motherboard on the light slave device, sets an application variable indicating on/off status of the LED to an appropriate value and prepares a response transaction indicating that the light is turned on successfully.

After processing all the transactions of the received ZigBee message, slave application program requests the stack to send a single ZigBee response message, containing the prepared ZigBee response transactions, to the ZigBee master controller. After that, slave application program discards the receive buffer of the stack and exits from the current iteration of the infinite loop. This finishes the sequence of actions performed in the infinite loop after receiving a new ZigBee message.

The last important algorithm in the infinite loop defines the sequence of actions performed by slave application program after receiving the result of network discovery operation. The result of network discovery operation includes the status of the network discovery operation performed. If the status indicates success, the result of network discovery operation also includes the descriptors of networks available for joining in the radio sphere of influence of the ZigBee node. The actions performed in the infinite loop after receiving the result of network discovery operation are shown in the below figure.

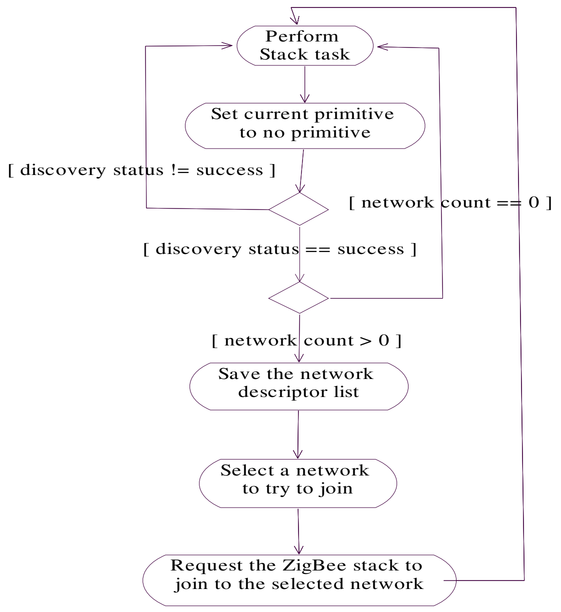

Perform Stack task
Set current primitive to no primitive
[discovery status != success]
[network count == 0]
[discovery status == success]
[network count > 0]
Save the network descriptor list
Select a network to try to join
Request the ZigBee stack to join to the selected network

Figure 4.16: The actions performed by slave application program after receiving the result of network discovery operation

Slave application program sets current primitive to no primitive. After that, it checks the status of the network discovery operation performed. If the status does not indicate success, it exits from the current iteration of the infinite loop. If the status indicates success but there is no available network to join, slave application program again exits from the current iteration of the infinite loop. If the status indicates success and there are available networks to join, then the following actions are performed.

The descriptors of found networks available for joining are saved to network descriptor list. From this list, an arbitrary network can be selected for joining. In the current implementation of slave application program, the first network in the list (the first network found as a result of network discovery operation) is selected for joining. After that, slave application program requests the ZigBee stack to join to the selected network and exits from the current iteration of the infinite loop. This finishes the handling of the result of network discovery operation in the infinite loop.

If the join attempt to the first network in network descriptor list is unsuccessful after receiving the result of the join attempt, the second network in the list is tried. If there is no such network in the list (the list has one network descriptor), slave application program requests the ZigBee stack to perform a network discovery operation again.

4.2.3 Web Application Software

The ZigBee master controller can be controlled and commanded from a PC connected to the Internet through the Internet. This is made possible via a web application stored in a web server connected to the Internet, and whose pages can be downloaded into a browser in a PC by connecting to the web server. The web application implemented in this thesis for this purpose has two web pages. This web application is used by the end user to send commands to the ZigBee slave devices indirectly through the ZigBee master controller. It is written using the ASP.NET technology and C# programming language.

The first page of the web application is the login page. It is comprised of two parts. The first part contains user interface elements written in ASP.NET markup language. This part is called user interface part. The user interface part of the login page contains ASP.NET web and HTML controls used to collect the login name, password and other login-related information for the end user. The second part is called event-handling part and contains event-handling routines. These routines handle events caused by activating some controls in the user interface part, and login page lifetime events.

The security between the web application and the end user's browser is based on SSL. The web server in which the web application is loaded shall be configured to use SSL for the web application. Upon SSL, the event-handling part of the login page implements forms-based authentication logic. It does this by using the forms-based authentication module provided by .NET. The forms-based authentication logic implemented by the event-handling part of the login page verifies the login name and password of the end user against the records found in an LDAP server. If the result of the verification is success, the end user is authenticated and a forms authentication ticket is sent to the end user. The end user can send the received ticket to the web server in the future accesses of the end user to the web application and the web application authenticates the end user automatically by inspecting the ticket. The ticket contains encrypted login name of the end user and a MAC. If the result of the above verification is not success, the end user is not authenticated and accepted as an anonymous user. The stand-alone LDAP server provided by the OpenLDAP project (slapd) is used as the LDAP server. All this forms-based authentication logic resides in the event handler of a button activating the login process.

There is a specific LDAP client software working along with the forms-based authentication logic in the event-handling part of the login page. The forms-based authentication logic relays the verification of user credentials to this software. This LDAP client software verifies the user credentials against the LDAP server by binding to the LDAP server using the supplied credentials. The connection between the LDAP client software and the LDAP server is secured via TLS. The LDAP server stores the user records in the LDAP directory maintained by it. The passwords of the users are encrypted before they are stored in the LDAP directory. The forms-based authentication logic also relays other user-related operations such as getting information about a user to the specific LDAP client software used in this thesis. This specific LDAP client software is implemented using .NET Directory Services library.

Each user record stored in the LDAP directory in the LDAP server contains mainly four attributes representing the login name, password, common name and email address of the user represented by the record. These four attributes are defined by specific classes defined in InetOrgPerson schema provided by the OpenLDAP project. This schema is included in the global definitions portion of the specific configuration file for the LDAP server. The user records are assigned to use the specific classes in this schema.

The second page of the web application is called the default web page. The default web page is comprised of two parts, again one of them is the user interface part and the other is the event-handling part. The user interface part of the default web page contains ASP.NET web and HTML controls used by the end user to select a ZigBee slave device and give commands to it, and used to display the results of these commands to the end user. The event-handling part of the default web page contains handlers for the page lifetime events and the events caused by activating some controls on the user interface part of the default web page.

The user interface part of the default web page is written using ASP.NET markup language. By using the user interface part of the default web page, the end user can:

- give a command to the ZigBee master controller to get the list of networked ZigBee slave devices
- see the list of networked ZigBee slave devices
- select a ZigBee slave device and see the clusters associated with the selected ZigBee slave device

- select a cluster and see the attributes associated with the selected cluster
- give commands to set or get the values of the attributes in the selected cluster associated with the selected ZigBee slave device
- see the results of the commands given to the selected ZigBee slave device (these results are the values of attributes targeted by the commands or the results of set operations on attributes targeted by the commands)

The event-handling part of the default web page is written in C# language. The most important page lifetime event handled in the event-handling part of the default web page is the page load event. (The page lifetime events are raised by the ASP.NET runtime.) The sequence of actions performed in the page load event handler is shown in the below figure.

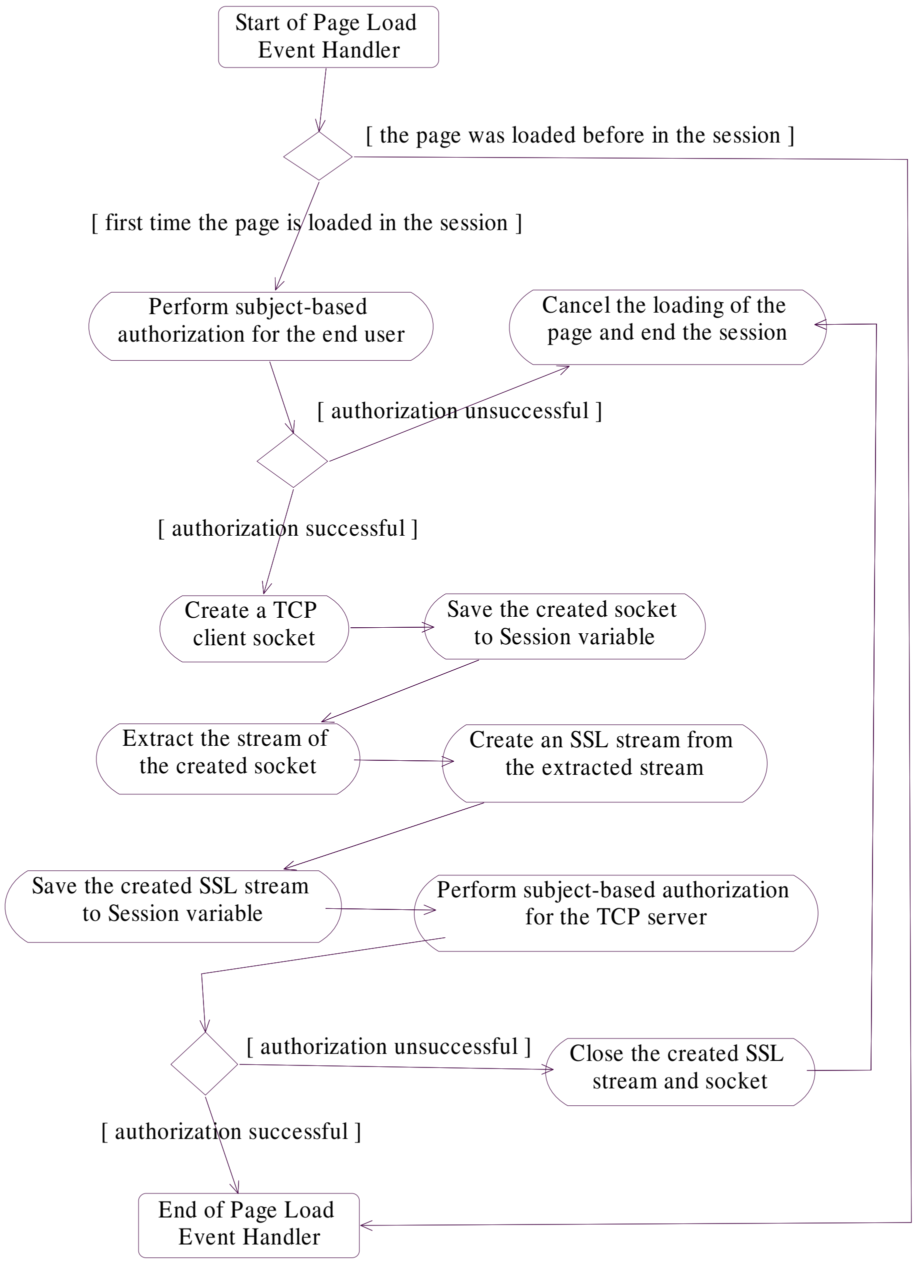

Start of Page Load Event Handler
[the page was loaded before in the session]
[first time the page is loaded in the session]
Perform subject-based authorization for the end user
Cancel the loading of the page and end the session
[authorization unsuccessful]
[authorization successful]
Create a TCP client socket
Save the created socket to Session variable
Extract the stream of the created socket
Create an SSL stream from the extracted stream
Save the created SSL stream to Session variable
Perform subject-based authorization for the TCP server
[authorization unsuccessful]
Close the created SSL stream and socket
[authorization successful]
End of Page Load Event Handler

Figure 4.17: The actions performed in the page load event handler of the default web page

The page load event handler first checks if it is the first time the default web page is loaded (requested) in the session. If the page was loaded before in the session, then the end user has been already authorized to access the page, so nothing is done in the page load event handler. If it is the first time the page is loaded in the session, the following actions are performed.

The page load event handler performs subject-based authorization for the end user in which it compares the subject name field of the X509 certificate of the end user with the allowed user name. If they are not equal, authorization is unsuccessful and the page load event handler cancels the loading of the page and ends the session. If they are equal, authorization is successful and the page load event handler:

- creates a TCP client socket, connected to the server socket opened by the ZigBee master controller, and stores this client socket in Session variable
- extracts the data stream of the created client socket
- creates an SSL stream from the extracted stream and stores the created SSL stream in Session variable
- performs subject-based authorization for the TCP server in which it compares the subject name field of the X509 certificate of the TCP server with the allowed subject name string indicating the ZigBee master controller

If subject-based authorization for the TCP server is unsuccessful, the page load event handler closes the created SSL stream and the client socket, cancels the loading of the default web page and ends the session.

The created objects above are stored in Session variable since they are used in subsequent handling of the events caused by activating some controls in the user interface part of the default web page.

In addition to subject-based authorization performed for the end user in the page load event handler, a user-based authorization for the end user is also performed before the end user accesses the default web page. User-based authorization is performed by the URL authorization module provided by .NET Framework. User-based authorization authorizes access to the default web page according to the login name of the end user. It prevents the anonymous and unauthorized end users from accessing the default web page. The URL authorization module receives the login name of the end user from the forms-based authentication module also provided by .NET Framework.

The security between the TCP client socket opened by the default web page and the TCP server socket opened by the ZigBee master controller is based on SSL. The default web page uses the SSL implementation provided by .NET base class libraries. After authenticating the TCP server socket with SSL, the page load event handler of the default web page performs subject-based authorization for the TCP server socket as mentioned above.

Event-handling part of the default web page also contains handlers for the events caused by activating some of the controls in the user interface part of the default web page. All of these controls are buttons. These buttons are used by the end user to send commands to the ZigBee master controller. The sequences of actions performed in these event handlers are similar in structure. The sequence of actions performed in such an event handler (a button press event handler) is shown in the below figure.

A button press event handler starts by reading the parameters entered by the end user to the appropriate user interface controls. It reads the parameters by using the server-side representations of the user interface controls found on the HTML page sent to the end user. By using the read parameters, the button press event handler forms a message containing the end user's commands and compliant with the specific application protocol used between the web server and the ZigBee master controller. After that, it writes the formed message to the SSL stream, created in the page load event handler and wrapping the TCP client socket, to send the message to the ZigBee master controller. Then, it waits for a response message.

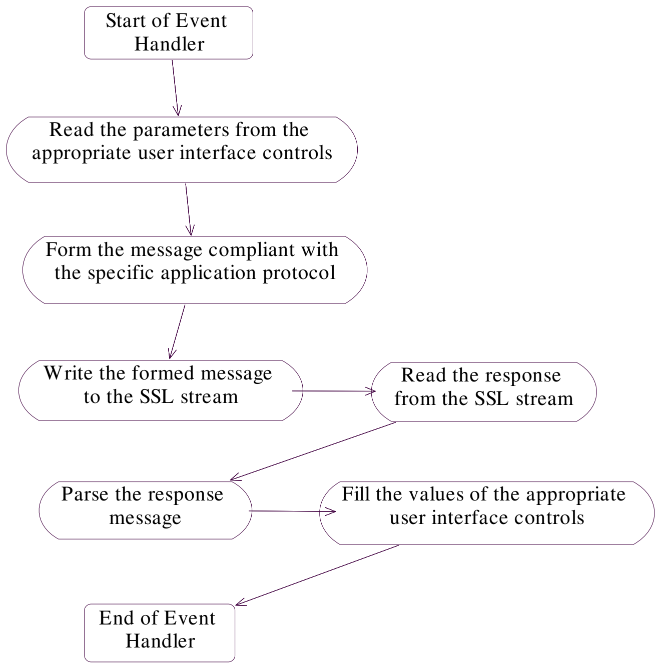

Start of Event Handler
Read the parameters from the appropriate user interface controls
Form the message compliant with the specific application protocol
Write the formed message to the SSL stream
Read the response from the SSL stream
Parse the response message
Fill the values of the appropriate user interface controls
End of Event Handler

Figure 4.18: The actions performed by a button press event handler of the default web page

While the response message is coming from the ZigBee master controller, the button press event handler reads the response message from the SSL stream. After reading the response message, the button press event handler tokenizes the response message. After that, it fills the contents of the appropriate user interface controls using the resulting tokens of the response message so that the end user can see the results of his/her commands to the ZigBee master controller.

4.2.4 Implementation Issues About the System-Wide Security

The security policies employed in specific parts of the system formed in this thesis are described in the relevant sections of this chapter. This section gives the details of security mechanisms used throughout the whole system in an implementation point of view. Security is provided for the following parts of the system:

- the communication between the end user's browser and the web application
- the communication between the web application and TCP/IP application program of the ZigBee master controller
- the communication between ZigBee application program of the ZigBee master controller and slave application program of a ZigBee slave device

The security for the communication between the end user's browser and the web application is based on SSL. The web server shall be configured to use SSL for incoming connections to the web application. The web server shall also be configured to send the certificate of the web application to the end user's browser in the SSL handshake. In the SSL handshake, the end user's browser (client) shall also send end user's certificate to the web application.

The actual key exchange and encryption mechanism used for the SSL communication between the end user's browser and the web application depends on the encryption algorithms supported by the end user's machine and the web server if it is assumed that both the end user's browser and the web server supports SSL. The preferred key exchange mechanism is RSA and the preferred encryption algorithm is AES for this SSL communication.

If the SSL protocol between the end user's browser and the web application succeeds, the end user will access to the login page of the web application, enter his/her credentials and submit the login page. Then, the web application tries to verify these credentials against an LDAP server by means of the LDAP client software implemented using Directory Services library provided by .NET Framework. This is called forms-based authentication. If it is not successful, the end user is redirected to the login page. Otherwise, the end user is redirected to the default web page of the web application. Upon loading the default web page, the web application performs two-way authorization.

The web application first performs a user-based authorization in which it compares the login name of the end user with an allowed login name. If they are not equal, the web application ends the session. Otherwise, it performs a subject-based authorization in which it compares the subject name of the X509 certificate of the end user with an allowed subject name. If they are not equal, the web application ends the session. Otherwise, it tries to connect to the ZigBee master controller.

The communication between the web application and the ZigBee master controller (TCP/IP application program of the ZigBee master controller) is secured via SSL. The web application (client) is configured to send its certificate to the ZigBee master controller. The ZigBee master controller (server) is also configured to send its certificate to the web application.

The key exchange and encryption algorithms used for the SSL communication between the web application and the ZigBee master controller depend on the encryption algorithms supported by the web server and the ZigBee master controller. The ZigBee master controller uses Microchip TCP/IP Stack version 4.55 implementing SSL. The SSL part of this stack supports RSA as the SSL key exchange algorithm and RC4 as the SSL encryption algorithm. Microchip also provides a data encryption library including implementations of AES, DES, 3DES and SkipJack. This data encryption library can be integrated with Microchip TCP/IP Stack version 4.55 to use AES as the SSL encryption algorithm for the SSL part of the stack. The preferred key exchange algorithm is RSA and the preferred encryption algorithm is RC4 for the SSL communication between the web application and the ZigBee master controller.

After the SSL communication is established between the web application and the ZigBee master controller, TCP/IP application program of the ZigBee master controller performs a subject-based authorization in which it compares the subject name field of the X509 certificate of the web application with an allowed subject name. If the comparison does not succeed, TCP/IP application program closes the connection between itself and the web application. Otherwise, it starts to wait for data from the web application.

Additionally, after the SSL communication is established between the web application and the ZigBee master controller, the web application performs a subject-based authorization in which it compares the subject name field of the X509 certificate of the ZigBee master controller with an allowed subject name. If the comparison does not succeed, the web application closes the connection between itself and the ZigBee master controller. Otherwise, it informs the end user of the established connection between itself and the ZigBee master controller.

The communication between the ZigBee master controller and a ZigBee slave device is secured by using the security mechanisms provided by the ZigBee protocol. All ZigBee devices shall use the same security mode provided by ZigBee. The ZigBee master controller is defined to be the trust center for the network coordinated by itself. It accepts a ZigBee slave device to the network only if the ZigBee slave device joins to the network using secure join. To securely join to the network, the ZigBee slave device must be preconfigured with a network key stored by the ZigBee master controller. After the ZigBee slave device joins to the network, all the communication between itself and the ZigBee master controller is secured with this network key. Each party adds a MIC at the end of a message to be transferred to the other party, encrypts the message and MIC with the network key (using AES) and sends the resulting data to the other party.

CHAPTER 5

PERFORMANCE

The performance of this thesis work is principally measured in terms of time in milliseconds that the processing of a specific command and/or response by a specific device takes. The specific command and/or response may be processed by the web server on which the web application is loaded, the PIC24FJ128GA010 microcontroller of the ZigBee master controller, the PIC18LF4620 microcontroller of the ZigBee master controller or the ZigBee slave device (the light or occupancy sensor).

Three commands are tested for measuring the performance. The first command ("get device list") is used by the end user to get the updated list of the ZigBee slave devices connected to the ZigBee master controller from the ZigBee master controller. This command starts from the browser of the end user, then goes to the web server from the browser through the Internet, then goes to the ZigBee master controller from the web server through the Internet and then a response to this command is sent from the ZigBee master controller to the web server through the Internet, then the response is sent from the web server to the browser of the end user through the Internet. This is the life cycle of this command. As mentioned, this command does not reach to any of the ZigBee slave devices.

The second command ("turn on") is used by the end user to turn on the light implemented by the first ZigBee slave device. This command starts from the browser of the end user, then goes to the web server from the browser through the Internet, then goes to the ZigBee master controller from the web server through the Internet, then is sent to the first ZigBee slave device by the ZigBee master controller using the ZigBee protocol and then a response to this command is sent from the first ZigBee slave device to the ZigBee master controller using the ZigBee protocol, then the response is sent from the ZigBee master controller to the web server through the Internet, then the response is sent from the web server to the browser of the end user through the Internet. This is the life cycle of this command.

The third command ("get current state") is used by the end user to get the current state of the occupancy sensor implemented by the second ZigBee slave device. This command starts from the browser of the end user, then goes to the web server from the browser through the Internet, then goes to the ZigBee master controller from the web server through the Internet, then is sent to the second ZigBee slave device by the ZigBee master controller using the ZigBee protocol and then a response to this command is sent from the second ZigBee slave device to the ZigBee master controller using the ZigBee protocol, then the response is sent from the ZigBee master controller to the web server through the Internet, then the response is sent from the web server to the browser of the end user through the Internet. This is the life cycle of this command.

The life cycles of these three commands represent all the possible life cycles that a command given by the end user can have. During the course of each of the above three commands, some processing on the command may be performed by each of the devices the command passes through. Besides, the ZigBee master controller has two microcontrollers, so both of them may perform some processing on the command. The average values of times for the processing of each of the three commands mentioned above and/or the response to each of the commands by each of the devices are shown in table A.1 in Appendix A in terms of milliseconds.

Another measure of the performance of this thesis work is the amount of RAM and ROM used by each of the following software:

- software loaded on the PIC24FJ128GA010 microcontroller of the ZigBee master controller
- software loaded on the PIC18LF4620 microcontroller of the ZigBee master controller
- software loaded on the PIC18LF4620 microcontroller of the first ZigBee slave device (light)
- software loaded on the PIC18LF4620 microcontroller of the second ZigBee slave device (occupancy sensor)

The RAM and ROM used by each of the software mentioned above belong to the microcontroller on which the software is loaded. The amount of RAM and ROM in bytes used by each of the software above is given in the below table.

Table 5.1: The amount of RAM and ROM in bytes used by the software loaded on each of the microcontrollers used in this thesis work

Software	**Amount of RAM used in bytes**	**Amount of ROM used in bytes**
software loaded on the PIC24FJ128GA010 microcontroller of the ZigBee master controller	2336	42002
software loaded on the PIC18LF4620 microcontroller of the ZigBee master controller	3602	64872
software loaded on the PIC18LF4620 microcontroller of the first ZigBee slave device (light)	2961	55784
software loaded on the PIC18LF4620 microcontroller of the second ZigBee slave device (occupancy sensor)	2948	50698

CHAPTER 6

CONCLUSION

In this chapter, firstly, the importance of the work, performed in this thesis, in the literature and the provisions of this work to the literature are discussed. Secondly, the weak and strong points of the solution suggested in the thesis are discussed. The weak points of the solution are mentioned along with the recommendations about the future improvements that can be made about the points.

The aim of the work performed in this thesis is to provide a solution to the problem of controlling a ZigBee wireless network by a PC (it is not necessarily a PC, it can be any device connected to the World Wide Web) through the Internet. The work performed in this thesis is important for real-world low-rate wireless personal area network (LR-WPAN) applications in the literature. The ZigBee network formed in this thesis is a small scale LR-WPAN. By taking the structure of the system formed in this thesis as an example, the developers of an LR-WPAN can connect their wireless network to the Internet and allow users to access their wireless network via the World Wide Web.

The work performed in this thesis gives an example of connecting a microcontroller-based embedded system simultaneously to an Ethernet and a ZigBee wireless network. This embedded system acts as a bridge between the Ethernet and the ZigBee network. It converts the TCP/IP packets received from the Ethernet to the ZigBee messages to be sent to the ZigBee network, and vice versa. It performs these conversions by means of a specific application protocol which is working over TCP/IP and contains information about the target ZigBee device and ZigBee commands or responses.

The work performed in this thesis realizes the connection of a PIC microcontroller-based embedded system to an Ethernet. This is done by using the Microchip TCP/IP stack and Microchip's Ethernet card containing Microchip's Ethernet controller. The work performed in this thesis also realizes the connection of a PIC microcontroller-based embedded system to a ZigBee network. This is done by using the Microchip ZigBee stack and Microchip's ZigBee RF card containing Microchip's IEEE802.15.4 compliant transceiver.

The strongest point of the solution provided by this thesis is that it allows the end user to control the ZigBee network using his/her browser through the World Wide Web. The end user may access to the ZigBee network from any machine which has connection to the Internet and the capability to interpret HTTP (namely, has a web browser). He/She does not have to bring with himself/herself an application program used to connect to the ZigBee network through the Internet.

Another strong point of the solution provided by this thesis is the security provided across the whole system. The security across the communication between the end user's browser and the web server, in which the web application is loaded, is based on SSL. Additionally, the web application performs forms-based authentication to authenticate the end user, and performs user-based authorization and subject-based authorization to authorize access to its default web page. So, an end user needs to have the correct X509 certificate and the knowledge of the correct login name and password to access to the default web page of the web application.

The security across the communication between the web application and the ZigBee master controller is also based on SSL. Both parties perform also subject-based authorization. Each party shall have the correct X509 certificate to access to the other party.

ZigBee’s own security mechanism is used to secure the communication between the ZigBee devices. To join to the ZigBee network, a ZigBee slave device shall be preconfigured with the network key used by the ZigBee master controller. This decreases the possibility of a malicious ZigBee device to join to the ZigBee network. Each of the messages transferred between the ZigBee devices contains a MIC and is encrypted. This avoids the possibility of a malicious ZigBee device to interpret these messages and modify these messages without being detected.

As mentioned in the related parts of the thesis, the ZigBee master controller is implemented by using two motherboards, each one with a microcontroller. The reason behind this implementation is to benefit from the plug-in compatibility of the Ethernet and RF cards (used also in the implementation of the ZigBee master controller) with the motherboards. However, using two motherboards in the implementation of the ZigBee master controller is a weak point in terms of space used and speed achieved. A future improvement over this point is using a single motherboard with a microcontroller with two SPI communication subunits. One of the SPI subunit of the microcontroller can be used for communicating with the Ethernet card and the other one can be used for communicating with the RF card. However, you are very likely need to manually connect one of the Ethernet and RF cards to the microcontroller in this case. Another improvement may be using a single board with a single microcontroller with one SPI subunit and one I^2C subunit, and connecting one of the Ethernet and RF cards to the microcontroller using SPI and the other one using I^2C. However, you are very likely need to manually connect one of the Ethernet and RF cards to the microcontroller, and must make changes in the TCP/IP or ZigBee stack to use I^2C instead of SPI communication in this case. In both of the improvements mentioned above:

- space used decrease, since a single board is used
- speed achieved throughout the system increase, since the present UART communication between the two microcontrollers of the ZigBee master controller is avoided, since there is one microcontroller

Another improvement over the weak point mentioned above may be using a single board with two microcontrollers, each one with a SPI subunit. Such a board can be found, but it is very likely that you need to manually connect one of the Ethernet and RF cards to the corresponding microcontroller in this case. Also, in this case there must be a communication mechanism between the two microcontrollers of the board so that speed disadvantage of the current design may not be eliminated. Space reduction can be achieved however, since there is a single board used.

The best thing you can do to eliminate the disadvantages of the current design of the ZigBee master controller is to design a new board with one microcontroller having two SPI subunits. The board shall contain an Ethernet controller connected to the first SPI subunit of the microcontroller, and IEEE802.15.4 transceiver connected to the second SPI subunit of the microcontroller. This improvement gives the most space reduction and speed increase among the improvements mentioned above.

Another weak point of the solution provided by the thesis is that the end user can send a message to the ZigBee master controller such that if the message contains commands setting or getting the values of some attributes, these attributes must pertain to a single cluster supported by a single ZigBee slave device. The transactions of a single ZigBee message must reference a single cluster of the target ZigBee slave device. So, the end user’s message corresponds to a single ZigBee message in the current design. It is a preference making it easy to program the devices of the system and resulting in a low memory consumption but slower system operation. A future improvement may be to allow an end user’s message to contain commands setting or getting the values of attributes pertaining to multiple different clusters of a single ZigBee slave device or pertaining to multiple different ZigBee slave devices. In this case, the end user’s message may correspond to multiple ZigBee messages. The web application may divide the end user’s message into n submessages each of which corresponds to a single ZigBee message. Then, the web application may send these submessages one by one to the

ZigBee master controller. After receiving a response to a submessage, it sends the next submessage to the ZigBee master controller. It also stores the received responses. After collecting all the responses from the ZigBee master controller, the web application combines them to a single response message and sends it to the end user. In this design, the web application may consume more memory than the current design in the thesis. The reason is that the end user's message and the response to it require more memory than their counterparts in the current design in the thesis. However, by this improvement, a greater speed throughout the system can be achieved. The reason is that the end user may send a message corresponding to multiple ZigBee messages to the web application by connecting to the web application only once. In the current design in the thesis, the end user can send a message corresponding to one ZigBee message to the web application by connecting to the web application only once.

Another weak point of the solution provided by the thesis may be that the ZigBee master controller and the web server in which the web application is loaded are separate devices communicating with each other through the Internet. An improvement over this design may be to implement web server functionality within the ZigBee master controller. This improvement increases the speed throughout the system since there is no need to connect the web server to the ZigBee master controller via the Internet. However, it increases the memory requirements and processing load on the embedded system implementing the ZigBee master controller.

Providing security across the entire system is considered to be a strong point of the solution provided by the thesis as mentioned above. The system-wide security depends mainly on two mechanisms: SSL and ZigBee security. There are attacks defined in the literature to both SSL and ZigBee's security mechanism at the protocol level and implementation level. These attacks can be considered weak points of the system. Firstly, the attacks to SSL and the countermeasures that can be taken against these attacks are discussed in the context of the system implemented in the thesis. Then, the same is done for the ZigBee's own security mechanism.

SSL is used to secure the communication between the end user and the web application and the communication between the web application and the ZigBee master controller. The first SSL communication mentioned above is preferred to use RSA key exchange in the SSL handshake layer, and AES in the SSL record layer. The second SSL communication mentioned above uses RSA key exchange in the SSL handshake layer, and RC4 stream cipher in the SSL record layer. Specifically, the second SSL communication is between .NET SSL library used by the web application and Microchip SSL implementation used by the ZigBee master controller. For the first communication, no assumption can be made about the specific SSL implementations used by the end user's PC and the web server containing the web application.

One of the most serious attacks against SSL protocol is the version rollback attack [15]. This is a potential attack that can effect the two SSL communications existing in the system implemented in the thesis. It can be fixed for the SSL communication between the end user and the web application by configuring the web server containing the web application to stop accepting SSL 2.0 connections. It can be fixed for the SSL communication between the web application and the ZigBee master controller by changing the source of Microchip SSL implementation so that it does not accept SSL 2.0 client connections.

Timing-based attacks are the important attacks targeted towards specific SSL implementations including OpenSSL, and using timing differences among the cryptographic operations performed by one of the parties (mostly the server) participated in the SSL conversation [16]. Three specific timing attacks are described in [18]. The timing attack which has the most serious consequences is the RSA timing attack since this attack reveals the RSA private key of the attacked party [18]. (The other two attacks reveal plaintext data from one particular session [18].)

All of three timing attacks described in [18] effect older versions of OpenSSL. Fixes for all three attacks are included in OpenSSL versions 0.9.6j and 0.9.7b. In the SSL communication between the end user and the web application, if the end user's PC or the web server containing the web application uses one of the older versions of OpenSSL (older than the ones listed above) as an SSL implementation, then at least one of the three timing attacks described in [18] is possible. In the SSL communication between the web application and the ZigBee master controller, the block padding timing attack (see [18]) is not possible since this communication uses RC4 stream cipher. This attack effects only block ciphers. Other two timing attacks are still possible for this SSL communication. [18]

The countermeasures that can be taken by specific SSL implementations to the three timing attacks mentioned above are described in [18]. These countermeasures can be used by the web server and end user's PC against these attacks. The same countermeasures, excluding the measure against the block padding timing attack, can be considered for .NET SSL library used by the web application and Microchip SSL implementation used by the ZigBee master controller. It is difficult to make any changes to the .NET SSL library by outsider. However, the source of Microchip SSL implementation can be changed to implement RSA blinding to defend the ZigBee master controller against RSA timing attack. Also, the source of Microchip SSL implementation can be changed to decrease the difference between the time taken by the operations performed when the SSL version number check on the incoming packet is successful, and the time taken by the operations performed on failure of the SSL version number check on the incoming packet. This second change can be performed to defend the ZigBee master controller against the second attack described in [18].

A chosen-plaintext attack to SSL protocol is explained in [17]. This attack is applicable when a block cipher is used for encryption in the SSL record layer. When using a block cipher for encryption, SSL mandates the use of cipher block chaining mode of encryption which requires an initialization vector in order to encrypt. The initialization vector for the first message to be encrypted in SSL is a pseudorandom string which is generated during the SSL handshake phase. However, initialization vectors for the subsequent messages are chosen in a deterministic way such that the initialization vector for a message is taken to be the final ciphertext block of the immediately preceding message. The chosen-plaintext attack described in [17] utilizes this property (or flaw) of SSL. By using this attack, it is possible to recover low-entropy strings such as passwords and PINs that have been encrypted. [17]

The feasibility of the chosen-plaintext attack mentioned above is also demonstrated by examples in [17]. The attack can be easily realized by installing a corrupted plug-in to a web browser [17]. So, if such a corrupted plug-in can be installed to the web browser used by the end user’s PC to connect to the web application, the SSL communication between the end user and the web application can be attacked in such a way that the end user’s login name and password can be recovered. If the web application is configured such that it allows adequately long password and login name to be chosen by an end user, then the possibility of this attack to recover the end user’s credentials can be reduced. A measure that stops this attack could be to prefer the use of stream ciphers for encryption in the SSL communication between the end user and the web application. The countermeasures that can be taken against this attack are described in [17]. Mostly, these measures require making changes to the SSL protocol [17].

The communication between the ZigBee devices is secured via ZigBee’s own security mechanism. The ZigBee master controller acts as the trust center of the ZigBee network. All of the ZigBee devices employ 128-bit AES encryption and 128-bit MIC on the transmitted ZigBee messages. AES encryption/decryption is performed in counter mode and MIC calculation/verification is performed in cipher block chaining mode as indicated in [1].

The first attack, to the ZigBee network, examined in the context of this thesis work is the Sybil attack. The Sybil attack is performed by a malicious node that illegally acquires multiple identities. After the malicious node joins to the network, it registers virtual identities and deceives the other nodes into thinking that the registered virtual identities and itself are normal nodes. As a result, the network starts to provide any possible services to the virtual identities and the malicious node, and consume information from them. By using this fact, the attacker can generate multiple legal messages having different sources and opposing to the legal messages generated by the legal nodes, and by this way it causes a DoS (Denial of Service) attack. [19]

In [19], a Sybil attack detection method and its response method to mitigate the effect of the attack in a ZigBee wireless network are described. The proposed detection method employs a challenge-response approach. A verifier node sends an encrypted request to a node and the node should respond the request properly within threshold time. In the system implemented in the thesis, the verifier node can be the ZigBee master controller. [19]

Microchip ZigBee Stack does not support commercial security mode [22]. It only supports residential security mode [22]. ZigBee specification does not mandate the use of end-to-end security using link keys for residential security mode [1]. Microchip ZigBee Stack provides support for only network keys [22]. When using only network key, only the default ACL entry is available for use in the MAC layer and ZigBee specification mentioned that the MAC layer security should not use optional external frame counter (i.e. no checking of incoming frame counter) [1]. This is because there is only one ACL entry available for use when using the network key and this entry contains only one optional external frame counter [1]. Therefore, replay attacks are possible on the MAC layer [21]. Network layer can discard replayed packets through checking of incoming frame counter of network packets, but MAC command packets are parsed only in MAC layer so that it is possible for an attacker to replay them [21].

The obvious solution to the attack replaying MAC command packets are, as stated in [21], to use different external frame counters in a ZigBee device for each neighbor device. This is possible by using a separate ACL entry and therefore a link key in a device for each neighbor device [21]. The source of the Microchip ZigBee Stack can be changed to allow for this solution.

ZigBee uses the procedure of CCM* in its security mechanism which is similar to stream cipher. CCM* uses counter mode for encryption of data packets. It uses a nonce and key to generate the key stream and produces ciphertext by XORing plaintext and key stream. So, it is obvious that the same nonce and key will generate the same key stream. If the attacker finds two encrypted packets that use the same nonce and performs XOR operation on these encrypted packets, he/she will obtain the XORed result of the decrypted version of the two packets. This is key stream reuse attack and it is applicable to the ZigBee network implemented in the thesis. [1, 21]

As mentioned in [21], by employing key stream reuse attack during network key update procedure started by the trust center, the attacker can reveal the network key used in the ZigBee network. Nonce used in the CCM* procedure of ZigBee security depends on the sender's address, frame counter, key counter and block counter values. (Nonce is used when encrypting packets using AES in counter mode by the sender.) If all of these values are the same for two packets, then there will be a nonce reuse in encrypting these two packets. Nonce reuse mainly occurs as a result of using the same frame counter value for two packets by the sender. The range of values for the frame counter field is extensive, but for instance after a power failure, the frame counter value is reset to zero and there is a possibility of using same the frame counter for encrypting a packet sent after the power failure and encrypting a packet sent before the power failure. [21]

There are some solutions suggested in the literature for the key stream reuse attack caused by the nonce reuse problem in ZigBee security. One solution suggested is to modify the format of the nonce in CCM* such that 8-byte sender (source) address in nonce is modified to an 8-byte random number when sending each one outgoing packet or when the device is rebooted [21]. This requires changing the security protocol proposed by ZigBee.

A second solution suggested is to store the ACL entries in non-volatile memory so that they persist their state after a power failure [20]. By using this solution, it is possible to determine the frame counter used for the last packet sent to a destination node before the power failure [20]. This solution has some disadvantages in terms of performance since transferring ACL entries between RAM and the non-volatile memory incurs a significant overhead in terms of speed [20]. However, this solution can be applied to the ZigBee network implemented in the thesis. The ZigBee devices in the system use PICDEM Z motherboard to perform ZigBee-related operations. ACL entries used by a ZigBee device can be stored in an EEPROM that can be installed to the PICDEM Z motherboard by using the prototype area on the PICDEM Z motherboard.

The MAC layer of the ZigBee protocol provided by IEEE 802.15.4 specification does not include any integrity or confidentiality protection for acknowledgement packets. The lack of a MIC protecting acknowledgements allows an adversary to forge an acknowledgement for any packet. An adversary need only create the forged acknowledgement with the appropriate sequence number from the original packet. Since the sequence number of a packet is sent in cleartext, this is an easy task for the adversary. An attacker can use this weakness. An attacker can identify a packet that he/she wishes is not received by the intended recipient. The attacker can transmit a short burst of interference while the packet is being sent, causing the

CRC to be invalid at the recipient and the packet to be discarded by the recipient. Then, the attacker can forge a valid-looking acknowledgment and send it to the sender, deceiving the sender into thinking that the intended recipient has received the packet. [20]

The acknowledgement forgery attack described above effects the frames parsed only by the MAC layer, such as MAC command frames. An application designer does not rely on acknowledgements since receiving an acknowledgement for a message does not mean that the recipient actually received the message. A solution can be to use authenticated acknowledgements. The recipient can compute a MIC over the received decrypted message and its own address, and attach this MIC to the acknowledgement packet. The sender can easily verify this MIC. An adversary can not compute the MIC since he/she does not have access to the decrypted message and the private cryptographic key used in the MIC calculations performed by the receiver and the sender. This solution requires changes to the source of the Microchip ZigBee stack and can be a future improvement for IEEE 802.15.4 specification. [20]

Another weak point of the system implemented in this thesis is the inadequate support for multiple connections to multiple ZigBee master controllers in the web application. In order to connect to multiple ZigBee master controllers, the end user must create multiple instances of the web application. A future improvement can be to make the web application multi-threaded and use one thread of execution for each connection to a ZigBee master controller. The ZigBee master controllers to be connected to by the web application can be determined by the end user during the life of the web application. The user interface of the web application must also be updated accordingly.

Each of the ZigBee maser controller to be connected to by the web application may control a different ZigBee network. These ZigBee networks may not be in the range of one another. The end user can collect information from multiple ZigBee master controllers controlling different networks and decide what actions to perform. Possibly, the end user's one command can change the status of multiple devices found on different ZigBee networks. Additionally, the web application can be used to transfer information between different ZigBee networks, that are not in range of one another, by communicating with the master controllers of these networks.

Another weak point of the system implemented in this thesis is the lack of autonomity for the ZigBee slave devices. A ZigBee slave device performs actions on receipt of a ZigBee message from the ZigBee master controller. A future improvement about this point can be to implement the ZigBee slave devices such that some of them sense some features of their environment, take appropriate actions when specific conditions on those features are met and inform the ZigBee master controller about the results of these actions. Alternatively, these sensing ZigBee slave devices can warn other ZigBee slave devices, behaving as actuators, to take the appropriate actions when specific conditions are met. These actuator slave devices can then inform the ZigBee master controller about the results of the performed actions. For example, in a home automation network, a ZigBee slave device can sense the temperature of a room periodically, and send a ZigBee message to another ZigBee slave device, controlling the air conditioner, when the temperature exceeds or falls below some threshold value. This ZigBee message instructs the ZigBee slave device, controlling the air conditioner, to adjust the room's temperature. After adjusting the room's temperature, the ZigBee slave device, controlling the air conditioner, can inform the ZigBee master controller about the performed operation on the room's temperature.

Alternatively, a sensing ZigBee slave device can only sense a feature of its environment and warns the ZigBee master controller when a specific condition on that feature is met. It does not take any actions or activate any slave actuators on its own. The ZigBee master controller can decide what actions to perform or which actuators to activate. This solution decreases the workload (hence processor and memory requirements) on sensing ZigBee slave devices. However, it increases the workload on the ZigBee master controller which must implement the decision mechanism in this solution.

Autonomity of ZigBee slave devices can be realized by assigning rules to those ZigBee slave devices. By assigning a rule, a ZigBee slave device can be programmed to autonomously perform specific actions when a specific condition is met or periodically. The overall system could be more flexible if the rules that can be assigned to the ZigBee slave devices can be formed and modified by the end user during the operation of the system.

A single rule can be related with more than one ZigBee slave device. For instance, a specific condition can be verified by collecting data from a set of ZigBee slave devices, and then another set of ZigBee slave devices can perform the appropriate operations. A rule engine containing an inference mechanism can be established. Its main responsibility can be collecting data from the ZigBee slave devices, verifying a set of specific conditions using those data, determining the rules to be realized using the verified conditions, and activating the related ZigBee slave devices to perform the specific actions related with the determined rules. It also verifies the correct realization of the rules by the ZigBee slave devices, and sends information about the status of the realization of the rules to the web application. This rule engine can be implemented within the ZigBee master controller or as a separate ZigBee device. It can store the rules in non-volatile memory and the rules can be loaded to the rule engine during the operation of the system.

REFERENCES

[1] ZigBee Alliance, Inc., 2006, “ZigBee Specification”

[2] Kinney Patrick, 2003, “ZigBee Technology: Wireless Control that Simply Works”, Kinney Consulting LLC

[3] Paillard Cedric, 2005, “Chips square off on ZigBee”, EE Times

[4] IEEE, Inc., 2003, “802.15.4 Specification”

[5] Wikimedia Foundation, Inc., 1 May 2009, “Serial Peripheral Interface Bus – Wikipedia, the free encyclopedia”, http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus , last visited on 6 May 2009

[6] Kalinsky David, Kalinsky Roee, 2002, “Introduction to Serial Peripheral Interface”, Embedded Development Community

[7] Freescale Semiconductor, Inc., 2004, “SPI Block Guide V04.01”

[8] Microchip Technology, Inc., 2007, “PICDEM Z Demonstration Kit User’s Guide”

[9] Microchip Technology, Inc., 2007, “Home Control, Lighting Profile Definition File”, the file comes with Microchip ZigBee Stack V1.0-3.8a, location of the file in Microchip ZigBee Stack V1.0-3.8a installation is MpZBee\ZigBeeStack\zHCLighting.h, name of the file is zHCLighting.h

[10] Microchip Technology, Inc., 2005, “Explorer 16 Development Board User’s Guide”

[11] Legg Gary, 2004, “ZigBee: Wireless Technology for Low-Power Sensor Networks”, TechOnline

[12] McCartney David, 2006, “ZigBee: Calling all software”, OpenSystems Publishing

[13] Cutler Tim, 2005, “Deploying ZigBee in existing industrial automation networks”, Industrial Embedded Systems Resource Guide

[14] Donnelly Michael, 2000, “An Introduction to LDAP”, Sendmail Inc.

[15] Schneier B., Wagner D., 1996, “Analysis of the SSL 3.0 Protocol”, USENIX Press

[16] Roehling Sigrid, 2004, “The SSL Protocol”, ECS

[17] Bard V. Gregory, 2004, “Vulnerability of SSL to Chosen-Plaintext Attack”, Cryptology ePrint Archive

[18] Phipps Colin, 2003, “Recent Attacks against OpenSSL likely to be applicable to other SSL implementations”, Netcraft Ltd.

[19] Gunhee Lee, Jaesung Lim, Dong-kyoo Kim, SungHyun Yang, MyungHyun Yoon, 2006, “An Approach Mitigating Sybil Attack in Wireless Networks using ZigBee”, Institute of Information Technology Advancement, Korea

[20] Sastry Naveen, Wagner David, 2004, “Security Considerations for IEEE 802.15.4 Networks”, ACM

[21] Ko Lee-Chun, 2006, “Security Considerations for Residential Mode on ZigBee Network”, Industrial Technology Research Institute, Taiwan

[22] Flowers David, Otten Kim, Yang Yifeng, Rajbharti Nilesh, 2007, “Microchip Stack for the ZigBee Protocol”, Microchip Technology Inc.

[23] Microchip Technology Inc., 2007, “PIC18F2525/2620/4525/4620 Data Sheet”

[24] Microchip Technology Inc., 2007, “PIC24FJ128GA010 Family Data Sheet”

APPENDIX A

Table A.1: Average time in milliseconds that the processing of a specific command and/or response to this command, by a specific device takes

Process	Time value in ms for specific commands		
	Get device list	**Turn on**	**Get current state**
The processing of the command throughout the whole system [1]	537	4366	4369
The processing of the command and response to the command in the web server[2]	423	4250	4255
The processing of the command in the web server[3]	16	17	18

[1] The process is started when the end user gives the command, and finished when the end user gets the response to the command.

[2] The process starts from the beginning of the event handler of the button activating the command, and finishes at the end of this event handler.

[3] The process starts from the beginning of the event handler of the button activating the command, and continues until the end of the transmission of the command to the PIC24FJ128GA010 microcontroller of the ZigBee master controller.

Table A.1: Average time in milliseconds that the processing of a specific command and/or response to this command, by a specific device takes (continued)

Process	**Time value in ms for specific commands**		
	Get device list	**Turn on**	**Get current state**
The processing of the command in the PIC24FJ128GA010 microcontroller of the ZigBee master controller[1]	40	43	45
The processing of the command in the PIC18LF4620 microcontroller of the ZigBee master controller[2]	70	74	75
The processing of the command in the ZigBee slave device[3]	----	3690	3688

[1] The process starts with the reception of the first byte of the command by the PIC24FJ128GA010 microcontroller from the web server, and continues till the end of the transmission of the command to the PIC18LF4620 microcontroller of the ZigBee master controller via UART mechanism.

[2] The process starts with the reception of the first byte of the command from the PIC24FJ128GA010 microcontroller by the PIC18LF4620 microcontroller, and continues till the beginning of the transmission of the command to the ZigBee slave device, or if the command does not require transmission to a ZigBee slave device, continues till the end of the processing of the command in the PIC18LF4620 microcontroller of the ZigBee master controller.

[3] The process starts with the reception of the first byte of the command from the PIC18LF4620 microcontroller of the ZigBee master controller by the ZigBee slave device, and continues until the beginning of the transmission of the response (to the command) to the PIC18LF4620 microcontroller of the ZigBee master controller.

Table A.1: Average time in milliseconds that the processing of a specific command and/or response to this command, by a specific device takes (continued)

Process	**Time value in ms for specific commands**		
	Get device list	**Turn on**	**Get current state**
The processing of the response to the command in the PIC18LF4620 microcontroller of the ZigBee master controller[1]	80	76	78
The processing of the command and response to the command in the PIC18LF4620 microcontroller of the ZigBee master controller[2]	154	3979	3975

[1] The process starts with the reception of the first byte of the response from the ZigBee slave device by the PIC18LF4620 microcontroller, or if the command does not require transmission to a ZigBee slave device, starts from the beginning of the preparation of the response in the PIC18LF4620 microcontroller, and continues until the end of the transmission of the response to the PIC24FJ128GA010 microcontroller of the ZigBee master controller.

[2] The process starts with the reception of the first byte of the command from the PIC24FJ128GA010 microcontroller of the ZigBee master controller by the PIC18LF4620 microcontroller, and continues until the end of the transmission of the response to the PIC24FJ128GA010 microcontroller of the ZigBee master controller.

Table A.1: Average time in milliseconds that the processing of a specific command and/or response to this command, by a specific device takes (continued)

Process	**Time value in ms for specific commands**		
	Get device list	**Turn on**	**Get current state**
The processing of the response to the command in the PIC24FJ128GA010 microcontroller of the ZigBee master controller[1]	49	46	47
The processing of the command and response in the PIC24FJ128GA010 microcontroller of the ZigBee master controller[2]	265	4089	4092
The processing of the response to the command in the web server[3]	28	27	27

[1] The process starts with the reception of the first byte of the response from the PIC18LF4620 microcontroller of the ZigBee master controller by the PIC24FJ128GA010 microcontroller, and continues until the end of the transmission of the response to the web server.

[2] The process starts with the reception of the first byte of the command from the web server by the PIC24FJ128GA010 microcontroller of the ZigBee master controller, and continues until the end of the transmission of the response to the web server.

[3] The process starts with the reception of the first byte of the response from the PIC24FJ128GA010 microcontroller of the ZigBee master controller by the web server, and finishes at the end of the event handler of the button activating the command.