AGH University of Science and Technology

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND ELECTRONICS

DEPARTMENT OF ELECTRONICS

Master's Thesis

Name and surname:	Dominik Nowak
Field of studies:	Electronics and Telecommunications
Title:	"Bridge between industrial wireless sensor network and wired Ethernet network"
Supervisor:	Ph. D. Cezary Worek

Kraków, 2012

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Acknowledgment

I would like to express gratitude to my supervisor, Cezary Worek, for his support, advices, patience during the course of this work, and to Łukasz Krzak for his time devoted to this thesis and for valuable advices. I would also like to thank the AGH team involved in the KIC InnoEnergy innovation projects, in area of “Intelligent energy-efficient buildings and cities” on the project “Active Sub-Station” for cooperation and support. I am glad that my thesis is a part of this project. Last but not least I would also like to thank my family, especially my parents for the support, that they provided me entire life.

Table of Contents

1. INTRODUCTION.

1.1. Background.

Originally developed in the Xerox research center in 1976 Ethernet has evolved over the years to become one of the most popular technologies for computer networks. Driven by the popularity of the Internet and local area networks Ethernet is massively used in almost every environment, including residential, commercial and industrial, providing a backbone for all communication services. Each field of application has different requirements, and different communication protocols, working on top Ethernet are used to fulfill them. Internet Protocol (IP) is probably the most common standard in the world used for example in the Internet itself and local area networking (LAN). However, due to more stringent requirements of industrial networks other communication protocols have been developed. One of them, which is in the scope of this work, is the IEC61850 standard used in electrical substation automation systems, that allows exchanging real-time data.

Although the Ethernet-based networks are considered a common standard, in recent years we can observe a shift towards wireless data transmission technologies that bring benefits to many applications. Both technologies have pros and cons but undoubtedly additional qualities can be achieved if both technologies are integrated. Such integration is also a challenge, because merging two different communication systems requires a thorough understanding of the combined technologies and their limitations. In this work aspects of combining Ethernet and ZigBee wireless networks will be discussed.

The issues and solutions presented in this work are related to the aspect of communication between new components of so called “smart grid”, which is the new approach to building power grid networks. Among other applications, such integration may be beneficial to electrical active substation automation systems, which are important components of the energy network of the future.

This work has been realized as a part of the KIC InnoEnergy “Active Sub-Station” project, dealing with, inter alia, the area of distribution sub-stations.

1.2. The goal.

The purpose of this work is to design a device used to connect Ethernet-based networks and ZigBee wireless network. Such device will be called a bridge. Two Ethernet-based networks are considered in this work, one using most common TCP/IP protocol and the second using IEC61850 industrial grade protocol, used widely in substation automation systems. For this reason two versions of the bridge have been built and they are described in this work.

The first version – the ZigBee-to-Ethernet bridge - uses TCP protocol and provides embedded HTTP server to visualize data from ZigBee nodes within the ZigBee Personal Area Network (PAN). Moreover Java web application is implemented and described, which allows to store data from many bridges into a MySQL database.

The second presented version is the ZigBee-to-IEC61850 bridge. It is designed to meet IEC61850 standard requirements and to connect ZigBee network to a substation automation system.

1.3. Layout.

The thesis is divided into following chapters:

1. INTRODUCTION – discussion of the purpose and scope of work.
2. THEORETICAL BACKGROUND – discussion of concepts and technologies used in the project of a ZigBee-to-Ethernet, and a ZigBee-to-IEC61850 bridge.
3. DESIGN OF A ZIGBEE-TO-ETHERNET BRIDGE – presentation of the tools and components used to design hardware and software, architecture of the system and key aspects of the design.
4. DESIGN OF A ZIGBEE-TO-IEC61850 BRIDGE – description of hardware components, used tools, software libraries, algorithms, and architecture of the device connecting ZigBee and IEC61850 network.
5. CONCLUSION – summary of the work.
6. REFERENCES – additional resources such as hardware schematics, SCL file and HTML files are included.

1.4. Abbreviations.

API – Application Program's Interface

CSMA/CD - Carrier Sense Multiple Access / Collision Detection

FPU – Floating Point Unit

GOOSE – Generic Object Oriented Substation Event

GPIO – General Purpose Input/Output

GSSE – Generic Substation Status Event

HAL – Hardware Abstraction Layer

HMI – Human Machine Interface

IDE – Integrated Development Environment

IED – Intelligent Electronic Device

IP – Internet Protocol

JSF – Java Server Faces

LAN – Local Area Network

LD – Logical Device

LN – Logical Node

LR-WPAN – Low Rate Wireless Personal Area Network

MAC – Media Access Control

MDI – Media Dependent Interface

MII – Media Independent Interface

MMS – Manufacturing Message Specification

MPU – Memory Protection Unit

NV – Non Volatile

OSAL – Operating System Abstraction Layer

OSI – Open System Interconnection

PCB – Printed Circuit Board

PER – Packet Error Rate

RTOS – Real Time Operating System

SAS – Substation Automation System

SCL – System Configuration description Language

SNTP – Simple Network Time Protocol

SoC – System-on-Chip

SV – Sampled Values

TCP – Transmission Control Protocol

UML – Unified Modeling Language

URL – Uniform Resource Locator

WSN – Wireless Sensor Networking

XML – Extensible Markup Language

2. THEORETICAL BACKGROUND.

2.1. ISO-OSI reference model and Service Access Points.

ISO's OSI model is the reference model used in the telecommunication. It specifies the path for information to move it from application running in one computer, through a network to a application running in another computer. OSI model consists of seven layers: physical, data link, network, transport, session, presentation and application. A single layer offers services for the upper one, and uses the services from the layer below it. Such layer-based approach allows to divide telecommunication system into smaller pieces, which could be better managed, updated and even interchanged. The responsibilities of each layer are as follows:

- Physical Layer (PHY): defines the mechanical, electrical, procedural and functional specifications for basic functions of the physical link between communicating network systems [1].
- Data Link Layer: is responsible for providing reliable transmit and reception of data over physical network link. Data link layer is divided by IEEE into logical link control (LLC), and medium access control (MAC) sublayer. MAC manages protocol access to the physical medium and provides MAC addresses, which are unique for each device [1], while LLC manages communication between devices over a single link of a network.
- Network Layer: is responsible for routing packets over networks, by using network addresses.
- Transport Layer: guaranties that data is delivered without errors in the correct sequence from source to the destination.
- Session Layer: establishes, manages and terminates communication sessions between devices.
- Presentation Layer: provides functions to convert data to common data representation formats to make it convenient for application layer.
- Application Layer: provides the interface for user and defines the functionality of the application, which uses protocols from the lower layers.

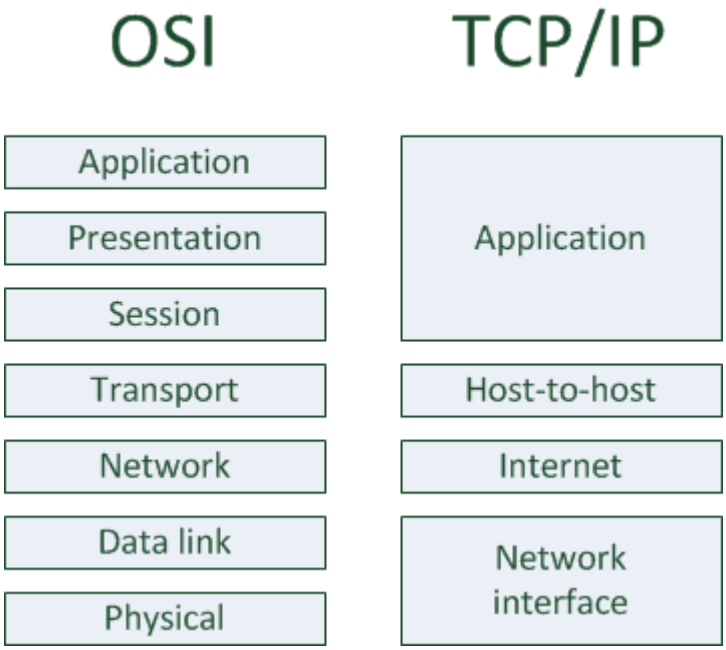


Figure 2.1. An illustration of the OSI protocol stack reference model and corresponding layers of the TCP/IP Protocol Suite stack[2].

Not all protocols define all OSI seven layers. For example TCP/IP defines only four of them (see figure 2.1):

- Network interface – corresponds to the Physical and Data Link layer.
- Internet layer – is responsible for task handled by OSI's Network layer.
- Host-to-host layer – as the name implies, provides correct data transfer from source to the destination – the same as the OSI's Transport layer.
- Application layer – merges Application, Presentation and Session layers described by the OSI model.

Figure 2.2 shows how the user data passes down the network stack. User data is created at the application layer. It may have to be divided into smaller pieces, because of the maximum length of the frame that is sent through the physical medium. Each layer adds to each piece of data some layer-specific information forming a protocol data unit (PDU). The name of this process is encapsulation.

For example in HTTP protocol user data is html document. Because in many cases this document is to large to send it as a whole, it has to be divided into smaller pieces called data. At the transport layer, the TCP protocol is used. It adds to the data a transport header and forms so-called segment (transport layer PDU). TCP is based on IP protocol, which corresponds to the network layer. At this layer, a network header is merged with the segment,

and after that it is called as a packet (network layer PDU). Finally the packet, a frame header and a frame trailer give a frame (data link layer PDU), which is then sent by a physical medium.

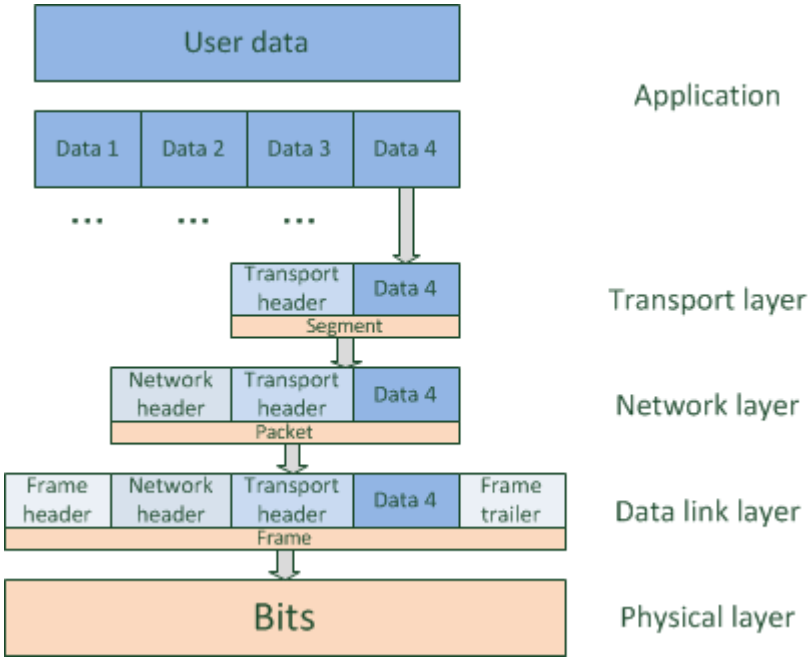


Figure 2.2. The path of a user data through a network protocol layers [3].

Exchange of data, control and configuration commands between layers is provided by Service Access Points (SAP). A SAP defines interface of the specific layer. Diagram illustrated in Figure 2.3 presents possible transaction types (so called primitives) between SAP of the lower layer, and an abstract user, which usually is the upper layer.

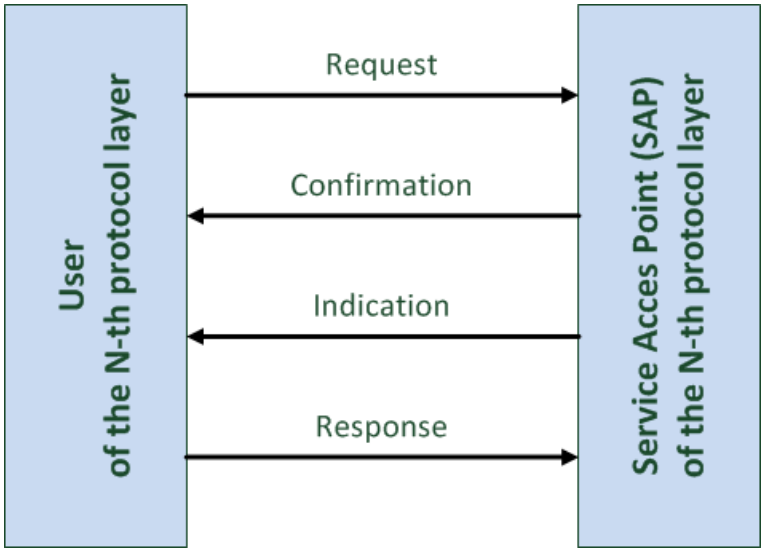


Figure 2.3. Service Access Point mechanism and its primitives.

If for example upper layer has data ready to send, a request primitive is used. When the requested service is completed, the lower layer SAP sends a confirmation. Such confirmations usually indicates whether the operation was successful or not. Asynchronous events can be indicated by a lower layer by an indication SAP primitive, which can be acknowledged by a response primitive.

2.2. IEEE 802.15.4 Standard overview.

2.2.1. Wireless sensor network.

Wireless sensor network (WSN) is a network which consists of multiple wirelessly connected elements called nodes. The basic parts of each node is a radio transceiver, microcontroller and sensors used to monitor various parameters, such as environmental, industrial, medical and other. One network can consist of a few to thousands of nodes. Depending on the requirements of a concrete application different communication protocols are used. When choosing type of WSN network, the most important parameters are channel access method, used frequency bands, possible network topologies, a throughput, a link budget, a power consumption and a price. There is no one, universal WSN network type suitable for all applications. In this work the ZigBee network based on IEEE 802.15.4 standard is used and is described in the following subsections.

2.2.2. Basic concepts of IEEE 802.15.4 standard.

The IEEE 802.15.4 standard has been developed to provide the solution for low-power and low data-rate Wireless Sensor Network requirements. The existence of standard decreases the cost of a WSN system, because manufacturers of integrated circuits, and developers do not need to develop their own low-level solutions and could focus on their application. Moreover the time to market is shorter and interoperability of radio devices between many manufacturers is simpler to provide.

The IEEE 802.15.4 standard provides lower network layers for a wireless personal area network. In contrast to IEEE 802.11 in which higher speeds are demanded and cost is not very critical, the discussed standard describes the wireless network in which devices are cheap, most of them are battery powered, send small amount of information and do not need high data-rates [4]. Output power of the transmitter is small to provide communications over distances from several to hundreds of meters with data-rates up to 250 kbps.

The standard is still under development, to adapt to new requirements. For this

reason, many releases are yet available, which are summarized in the table 2.1 [4] .

IEEE 802.15.4 version	Short description
IEEE 802.15.4 - 2003	Initial release of the IEEE 802.15.4 standard. Two versions of PHY are described: - for frequency band 868 and 915 MHz - for frequency band 2.4 GHz
IEEE 802.15.4 - 2006	This version is provided for an increase in the possible data rate for lower frequency bands and defines new modulation schemes: - three for 868 and 915 MHz - one for 2.4 GHz
IEEE 802.15.4a	Two new PHYs have been defined: - using UWB technology - using chirp spread spectrum at 2.4 GHz
IEEE 802.15.4c	Updates for PHYs
IEEE 802.15.4d	Updates for PHYs
IEEE 802.15.4e	MAC enhancements to IEEE 802.15.4 in support of the ISA 100.11a application have been defined.
IEEE 802.15.4f	Defines new PHYs for UWB, 2.4 GHz band and 433 MHz
IEEE 802.15.4g	Defines new PHYs for smart neighbourhood networks, which could be used by a smart grid application.

Table 2.1. The IEEE 802.15.4 standard releases [4].

2.2.3. Stack architecture.

IEEE 802.15.4 defines a low rate wireless personal area network (LR-WPAN) stack architecture, which is presented in Figure 2.4. Each block represents a specific layer. Arrows represent SAP interfaces between layers as described in the standard [5].

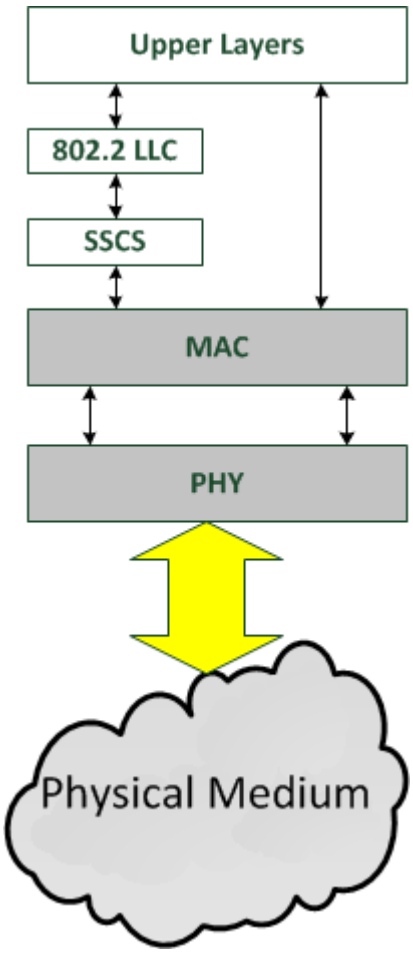


Figure 2.4. LR-WPAN device architecture [5].

Because IEEE 802.15.4 standard defines only a physical and a MAC layer, higher layers usually need to be developed. There is no one solution which fits all applications, so numerous of upper layer protocols have been developed. Some of them are presented in the table 2.2.

Standard	Short description
ZigBee	This standard, developed by ZigBee Alliance, is used in wireless sensor networks which have low power requirements in plenty of applications, such as: factory automation systems, home security systems, consumer electronics, health care [6]. The idea is to provide low power wireless connectivity and replacement of cable installations. To meet the varied, depending on application requirements, ZigBee introduces profiles of the wireless system.
RF4CE	Radio Frequency for Consumer Electronics (amalgamated with the ZigBee alliance) enables design of remote control devices using RF instead of traditional infra-red technology. It improves the communication with consumer electronics because of bidirectional transmission and better signal propagation properties.
6LoWPAN	IPv6 over Low power Wireless Personal Area Network. The idea is to apply the internet protocol event to the small, low power devices.
Wireless HART	Provides a wireless protocol for the process measurement, control, and asset management applications. It is based on industrial HART communication standard. It provides a self-organizing, and a self-healing mesh network.
ISA100.11a	This standard is deployed by ISA and is an open-standard intended to be used in industrial automation.
MiWi	Developed by Microchip P2P and designed for use in low data-rates, short distance low cost network. It is intended to be used in applications such as building automation, industrial monitoring and control, remote control and meter reading.

Table 2.2. 802.15.4 based higher layer protocols [4].

2.2.4. Device classes.

The IEEE 802.15.4 standard introduces two classes of devices operating in a wireless network – Full Function Device (FFD) and Reduced Function Device (RFD).

Full Function device can work in any topology, like star, tree or mesh. In most cases such device has to be powered all the time, and can not go into sleep mode. Therefore a

battery power supply is not recommended. FFD can talk to any other device and implements complete protocol set. Each FFD may work as a PAN coordinator, but only one coordinator can operate per one PAN network. The coordinator performs the same functions as a router, but also manages the network formation, device authorization etc.

The second class of IEEE 802.15.4 device is a RFD. A RFD has got reduced functionality and can only talk to FFDs, so it always acts as end device such as, sensor or actuator. Its use is limited to star topology or it can act as end-device in a peer-to-peer network. RFD cannot become a PAN coordinator and has got reduced protocol set, for example it does not route packets. Typically such node is in a sleep mode most of the time and wakes up periodically performing network tasks. Because of that, it usually consumes much less power then a FFD and can be battery powered.

2.2.5. Physical layer

PHY layer provides two services:

- PHY data service, which enables the transmission and reception of PHY protocol data units (PPDUs) across the radio channel.
- PHY management service.

The IEEE 802.15.4 systems operate in unlicensed radio bands. Some commonly used bands and corresponding transmission methods are presented in the table 2.3.

PHY	Frequency band [MHz]	Available channels	Bit rate (kbps)	Symbol rate (kbaud)	Modulation
868/915	868 – 868.6	1	20	20	BPSK
	902 – 928	10	40	40	BPSK
868/915 (optional)	868 – 868.6	1	250	12,5	ASK
	902 – 928	10	250	50	ASK
868/915 (optional)	868 – 868.6	1	100	25	O-QPSK
	902 – 928	10	250	62,5	O-QPSK
2450	2400– 2483,5	16	250	62.5	O-QPSK

Table 2.3. IEEE 802.15.4 frequency bands.

Direct Sequence Spread Spectrum technique is used making the transmission more resistant to interference from undesired narrow-band signals.

The PHY is responsible for the following tasks:

- activation and deactivation of the radio transceiver,
- energy detection (ED) within the current channel,

- link quality indicator (LQI) for received packets, that provides information for application about quality of wireless link,
- clear channel assessment (CCA) for carrier sense multiple access with collision avoidance (CSMA-CA),
- channel frequency selection,
- data transmission and reception.

It's worth to note that energy detection (ED) is used not only to select the best channel during network initialization, but also it provides possibility to adapt to a changing RF environment by selecting another channel if a link quality in the current one is causing transmission problems.

2.2.6. MAC layer and frame formats.

2.2.6.1. General description.

The MAC layer is responsible for

- data reception and transmission scheduling,
- data validity/integrity checking,
- acknowledgment of frame delivery,
- node addressing,
- time synchronization,
- association and disassociation of nodes,
- CSMA/CA multiple access,
- handling of so called guaranteed time slots (GTMs).

The IEEE 802.15.4 MAC layer provides interface for higher layer by defining two types of services: MAC data service and MAC management service. It also provides hooks that can be used by security mechanisms [5].

MAC data service enables the transmission and reception of MAC Protocol Data Units (MPDUs) across the PHY data service [5]. The name of the interface to higher layer is MLDE-SAP.

MAC management service is responsible for invoking the layer management function and it maintains a database of managed object associated with the MAC sublayer.

2.2.6.2. MAC frame formats.

There are four types of frames: beacon, command, data and acknowledgment. All frame types are based on a general MAC frame format, that is presented in the Figure 2.5.

This figure shows also how physical message transmitted through the radio channel looks like. Preamble and start of packet field allows a receiver to synchronize to a message. MAC Protocol Data Unit is divided into MAC header (MHR), MAC Service Data Unit (MSDU) and MAC footer (MFR).

MAC header contains three fields: frame control, sequence number and address information. Frame control field contains the information about a frame type, addressing field properties, and other specific for each network operation mode. It also indicates if the packet data is encrypted due to security reasons. Sequence number field is used for sending one acknowledgment per many frames being transmitted through a radio channel. It improves the effective application data throughput. Addressing field contains destination PAN identifier, destination address, source PAN identifier and source address. Frame payload has variable length and includes information specific for each frame type. FCS field with 16 bit ITU-T CRC is used to check data integrity and is calculated from MHR and MSDU fields.

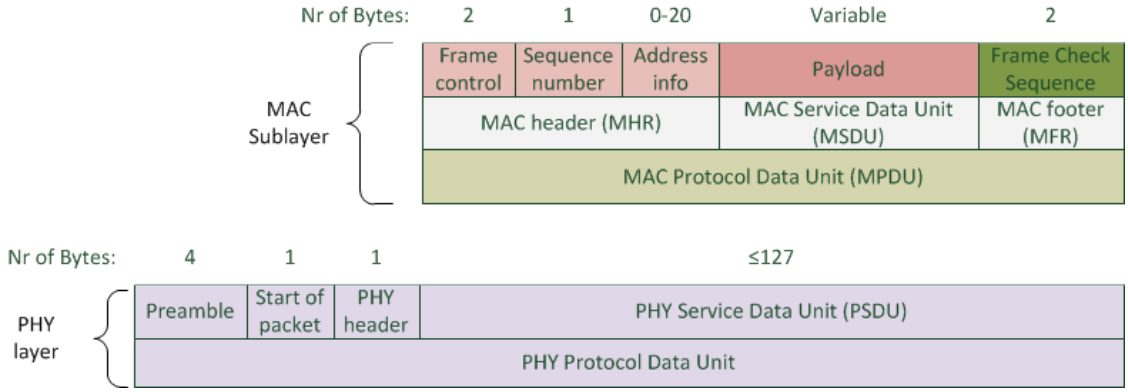


Figure 2.5. General MAC frame format.

Detailed description on possible MAC frames is presented in Appendix A.

2.2.6.3. Beacon and non-beacon networks.

There are two possible modes of network operation: using so called beacon frames or not. Example of beacon-enabled communication is presented in the Figure 2.6.

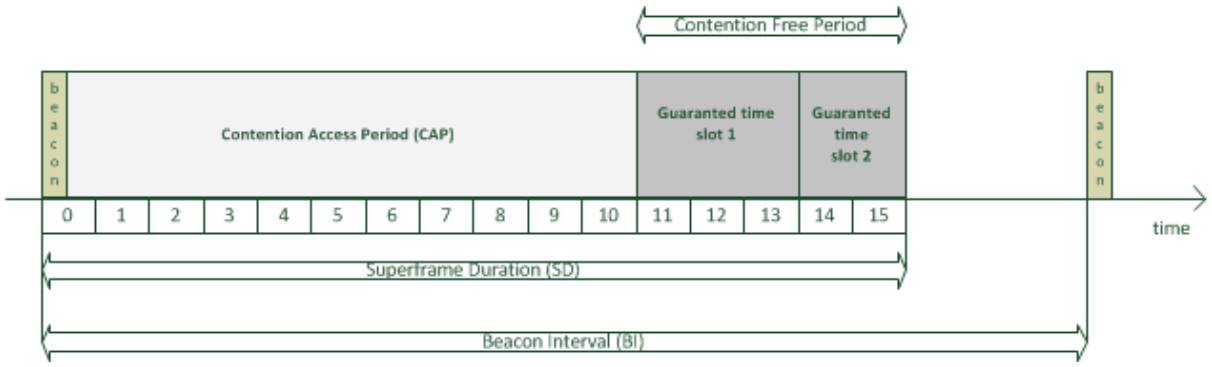


Figure 2.6. Example of beacon-enabled communication.

It has got so-called superframe structure in which network coordinator transmits beacons at predetermined time intervals. Superframe is divided into two characteristic periods with different channel access methods implemented.

During the contention access period slotted CSMA/CA channel access mechanism is used, and a device, which has data to be send, waits for the boundary of the next backoff slot, after random number of backoff slots period. If no energy is being detected the channel is regarded to be free, and device's message is being transmitted. If channel is found to be busy, device continues to wait random backoff slot number.

During the contention free period each device which requires constant bandwidth has got guaranteed time slot in which only this device can transmit it's messages.

Non-beacon transmission does not divide time into contention access and contention free period. Because there is not transmission synchronization frame, the contention access, based on CSMA/CA channel access mechanism is all the time.

2.2.6.4. Transactions in a beacon enabled networks.

Figures 2.7 and 2.8 demonstrate communication from a ZigBee device (Router or End -Device) to a coordinator and from a coordinator to a device.

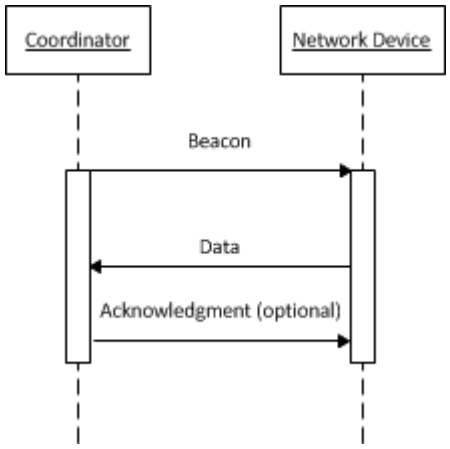


Figure 2.7. Sequence diagram of the transmission between a coordinator and a network device in a beacon-enabled network. Network device has data for the coordinator.

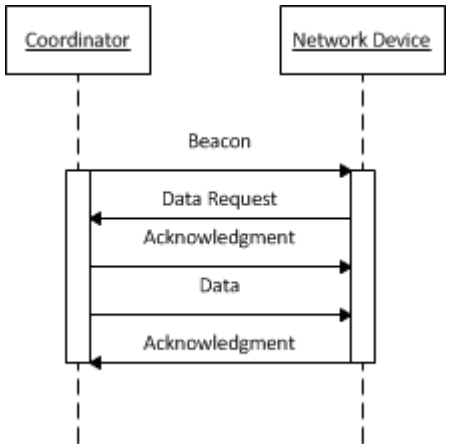


Figure 2.8. Sequence diagram of the transmission between a coordinator and a network device in a beacon-enabled network. The coordinator has data for a network device.

If a network device has data ready to send to the coordinator, at first it has to listen for the network beacon. Then device synchronizes itself to the superframe structure and by using slotted CSMA/CA mechanism it transmits data frame to the coordinator. After that coordinator can transmit an acknowledgment frame .

If the coordinator has got a pending message, then it indicates that fact in the network beacon frame. When the device, to which the coordinator wishes to send a packet, recognizes its address in the pending address field, it transmits a MAC command requesting the data using slotted CSMA/CA mechanism [7]. After receiving acknowledgment, coordinator transfers the requested data, and after reception of that data another acknowledgment is send by the network device [5].

Beacon enabled mode of transmission lowers the level of power consumed by a

coordinator, because it does not have to be active all the time, but only during superframe duration period. In the time between the end of superframe and the beginning of the next beacon frame the device can be switched into low power mode. Furthermore beacon-enabled mode can provide higher channel usage than non-beacon network, by utilizing slotted CSMA/CA mechanism.

2.2.6.5. Transactions in a non-beacon networks.

Another channel access method that does not use beacon frame is unslotted CSMA/CA. In this method before network device sends it's frame, it has to wait for a random period. If channel is free, the device transmits data, otherwise random wait period is generated before next attempt. Transmission is acknowledged by ACK frames, which are send without using the CSMA/CA mechanism. An example of this type of channel access method is presented in Figures 2.9 and 2.10.

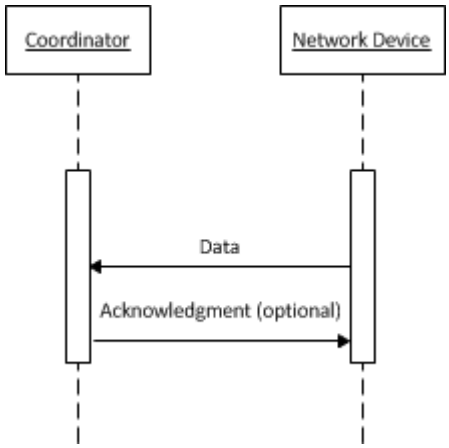


Figure 2.9. Sequence diagram of the transmission between a coordinator and a network device in a non-beacon network. Network device has data for the coordinator.

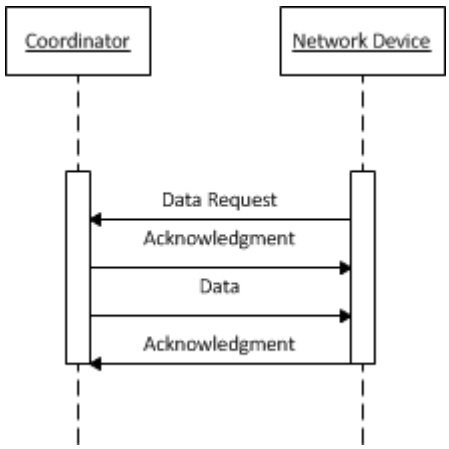


Figure 2.10. Sequence diagram of the transmission between a coordinator and a network device in a non-beacon network. The coordinator has data on request for a network device.

In Figure 2.9 a network device has data for a coordinator, and it simply sends it using CSMA/CA mechanism described above. Coordinator can acknowledge the successful reception of the packet. Figure 2.10 presents situation in which coordinator has pending data for the network device. At first the network device has to check if data is ready using polling mechanism. It is done by sending data request frame. If data is pending, the data frame is transmitted, else the coordinator transmits a data frame with a zero-length payload.

2.3. Analysis of the ZigBee Standard.

2.3.1. General description.

ZigBee is a IEEE 802.15.4 based protocol for wireless sensor network applications. Typical WSN network requires low power consumption of the node, which provides longer battery life. Moreover low cost, small footprint and other features like mesh networking, that allows communication between many devices in wide networks are required. All these features are characteristic for ZigBee technology.

ZigBee defines network and application layers on the top of the IEEE 802.15.4 MAC and PHY layers. ZigBee network is self-configuring, self-healing and provides interoperability between devices from different manufacturers, even if each of node performs different functions. ZigBee provides security mechanisms including node authorization, 128-bit AES data encoding with various key distribution options All this combined with flexibility and extendability make ZigBee good solution for WSN networks.

ZigBee standard is developed by the ZigBee Alliance, which is open, non-profit association of several hundred companies. Before the product can carry the ZigBee Alliance logo, the ZigBee Certification program has to be successfully completed. This ensures interoperatibility and proper coexistence between certified ZigBee devices. There are two types of ZigBee certified testing programs [8]:

- ZigBee Compliant Platform (ZCP) – for modules that are intended to be used as a part of end products. For example CC2530-ZNP provided by Texas Instruments along with software framework, which is used in devices described in Chapters 3 and 4, is a ZCP.
- ZigBee Certified Product – the certificate is for products that are built upon a ZigBee Compliant Platform, and they have to pass the test, if they are to be sold with ZigBee logo.

2.3.2. ZigBee application profiles.

In addition ZigBee Alliance defines several application profiles, suited for various purposes, in ex.:

- Smart Energy profile – intended for devices that provide information about energy usage, for example power meters.
- Home automation profile – suited for devices that control home appliances, for example remote control of heating, ventilation and air conditioning (HVAC) appliances as well as for home entertainment.
- Building Automation profile – allows integration of almost every system in a building

such as heating, lighting management and security.

- Light Link – suitable for light sources and light controllers

Other profiles are currently not made public:

- Remote Control profile,
- Health Care profile,
- Input Device profile,
- Retail Services profile,
- Telecom Services profile,
- 3D Sync profile.

Furthermore additional private profiles suited for specific applications can be defined by a ZigBee devices developers. The list of possible profiles is still open, and other public profiles may be provided by the ZigBee Alliance in the future.

2.3.3 ZigBee stack architecture.

ZigBee Stack Architecture is illustrated in Figure 2.11. Each main part of the stack is described in the next three subsections.

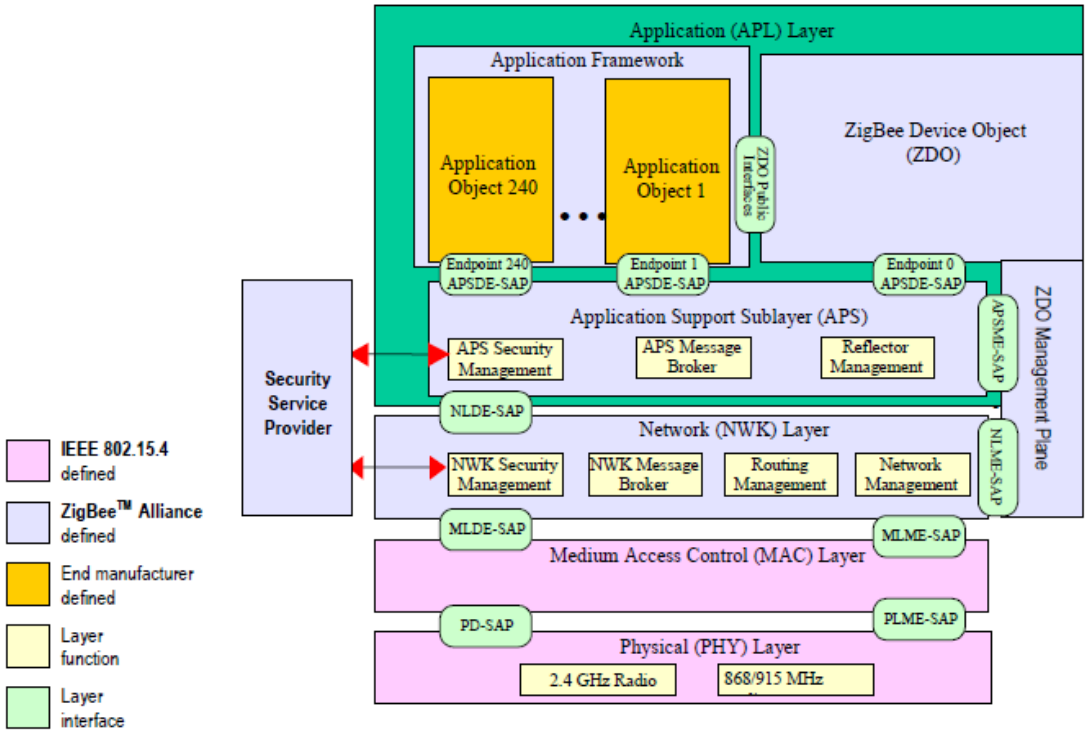


Figure 2.11. Outline of the ZigBee Stack Architecture [9].

2.3.3.1. Network (NWK) layer of the ZigBee stack architecture.

Network layer controls IEEE 802.15.4 MAC sub-layer and provides an interface to the application layer. To accomplish this, two service entities are used. The first one – NWK layer

data entity (NLDE) provides the data transmission service via the NLDE-SAP. The NWK layer management entity (NLME) is used to manage services via the NLME-SAP [9].

Network layer is responsible for the following functions:

- joining and leaving the network by ZigBee devices,
- providing cryptographic security for transmitted frames,
- routing frames to their destination,
- discovering and maintain route tables,
- discovering one-hop neighbors,
- storing of important information about neighbor devices.

Three types of devices are used in the network: a coordinator, router and end-device. A coordinator is responsible for starting a network and assigning addresses to newly associated devices. Router routes frames and is likely to be constantly powered on, in contrast to end-devices. Coordinator and router are Full Function Devices (FFD) and end-devices are Reduced Function Devices (RFD) as described in IEEE 802.15.4 standard.

ZigBee supports three topologies:

- star – in which network is controlled by a single device (coordinator) to which all other devices are directly connected,
- tree – which may employ beacon oriented communication because of hierarchical structure,
- mesh – which allows peer-to-peer communication and makes the network more resistant to network failures, by giving possibility of creating alternative paths.

General NWK Frame Format is presented in Figure 2.12.

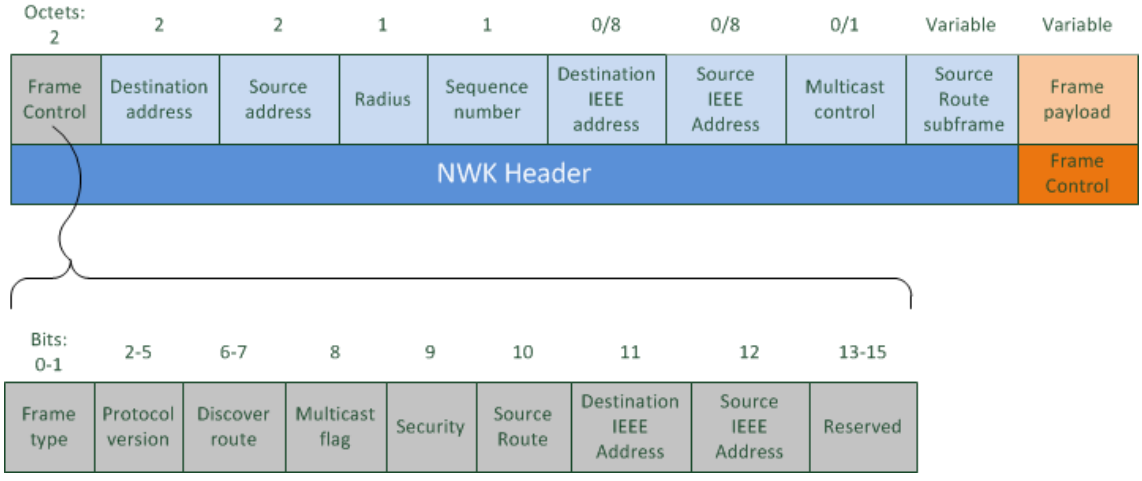


Figure 2.12. General NWK Frame Format.

NWK frame fills NWK frame payload and consists of the following fields:

- Frame Control – it determines the type of frame – data and command frames are supported and are presented in Figures 2.13 and 2.14. There are also sub-fields for discovery operations, indicating that security is used or the frame is a multicast or not. The rest of sub-fields ares used to include or not long IEEE address into the frame.
- Destination short network address.
- Source short network address.
- Radius – which specifies the range of a transmission, measured as the maximum number of hopes from the source to the destination. Each receiving device shall decrement this field by 1.
- Sequence number - which is incremented each time the NWK layer constructs a new NWK frame. This field is used to provide safety and to determine the order of the packets in a receiver node.
- Destination long (8-octet length) IEEE address.
- Source long IEEE address.
- Multicast control.
- Source Route subframe.
- Frame payload for higher layer data.

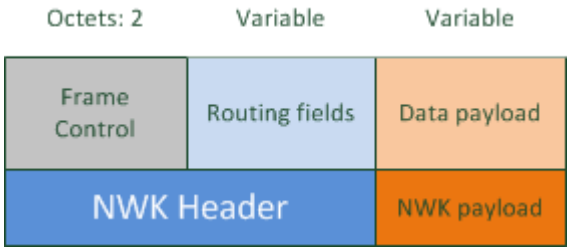


Figure 2.13. NWK Data Frame Format.

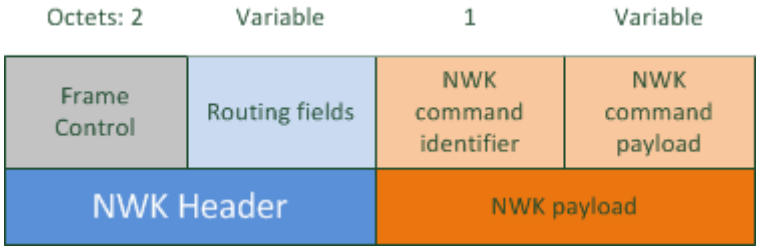


Figure 2.14. NWK Command Frame Format.

2.3.3.2 Application layer of the ZigBee stack architecture.

Application layer consists of Application Framework, ZigBee Device Object (ZDO) and Application Support Sublayer (APS).

Application Framework (AF) provides the functions, data types and data-frame formats for transporting data to facilitate the profile building process. It carries the environment in which application objects are built into ZigBee device. Application Object is responsible for initiation of standard network functions as well as controlling and managing of the protocol layers. AF communicates with APS via APSDE-SAP, which includes primitives for data transfer such as: request, response, indication and confirmation. Transfer is carried out between peer app object entities. Up to 240 distinct Application Objects could be defined. There are also two special endpoints: to interface data to the ZDO, and to interface data function to broadcast data to all app objects [8].

Zigbee Device Object (ZDO) is responsible for:

- establishing a secure high-level connection between ZigBee devices,
- discovering devices and their services in the network,
- determining whether the device is coordinator, end device or router,
- initiating a binding requests or replying to them,
- providing the interface to the lower portions of the ZigBee protocol stack via End Point 0 (EP0) [9].

Finally the APS sublayer provides:

- messages forwarding between bound devices,
- removal and filtering of messages,
- reliable data transport,
- mapping 64 bit IEEE addresses to 16 bit NWK addresses,
- maintaining tables for binding.

APS layer defines three types of frames: data frame, command frame and acknowledgment frame (figures 2.16, 2.17 and 2.18). The general APS Frame Format is illustrated in the Figure 2.15. Frame Control carries out information about frame type, delivery mode (normal unicast, indirect addressing, broadcast, group addressing), and other properties of the transmission. At this layer transmission occurs between endpoints. There are up to 240 application endpoints and one special endpoint for ZDO (EP 0) [9].

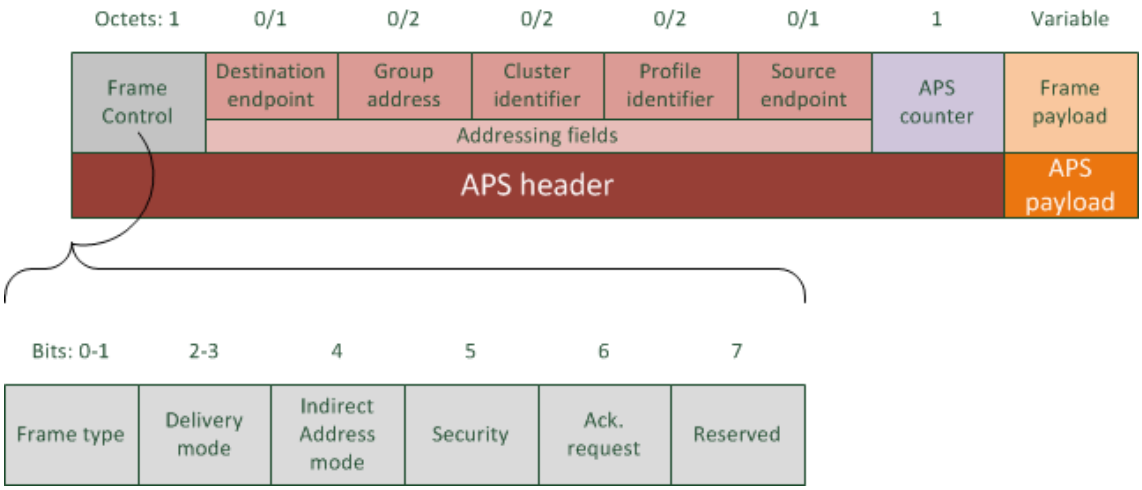


Figure 2.15. General APS Frame Format.

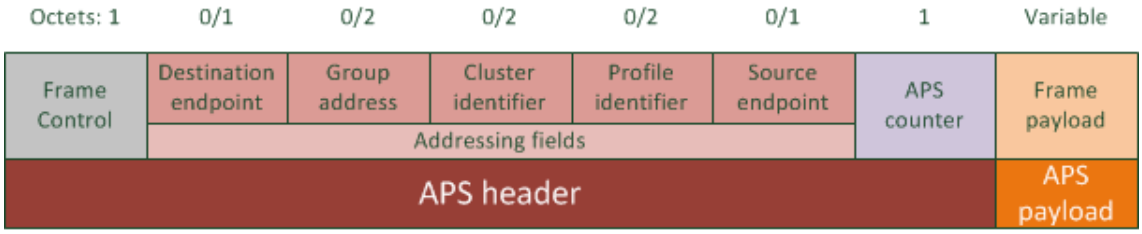


Figure 2.16. APS layer data-frame format

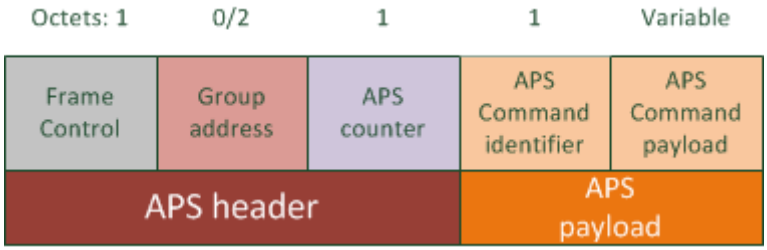


Figure 2.17. APS layer command frame format

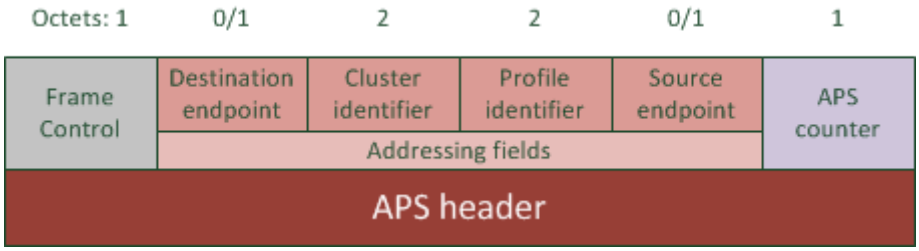


Figure 2.18. APS layer acknowledge frame format.

During design process of a ZigBee device the application profile has to be selected. An application profile defines device descriptions, application-level interfaces, message formats and standard practices for a given application type. This allows to create interoperable, distributed applications. Devices within an application profile communicate with each other by the means of clusters. A cluster is a related collection of commands and attributes, which together define an interface to specific functionality [10]. For example the "on/off" cluster and "alarms" cluster can be used to communicate a door lock with an alarm in a home automation profile compatible system. Some clusters are standardized and defined in the ZigBee Cluster Library (ZCL) which enables clusters re-usage across many profiles.

2.3.3.3 ZDO Management Plane of the ZigBee stack architecture.

ZDO Management Plane is used by APS and NWK layers in the process of communication with the ZDO[8]. APSME-SAP makes an interface between ZDO Management Plane and APS that is used for transport of management commands to:

- unbind or bind two devices together, by creating an entity in the binding table,
- communicate with AIB (APS Information Base),
- add and remove groups to endpoints.

NLME-SAP is an interface for NWK layer and is used to network discovery, access and formation [9].

2.3.4. Security of a ZigBee network.

Because data physically transmitted over a radio channel can be easily captured and recognized by a 802.15.4 compatible sniffer, security mechanisms are needed. In ZigBee unauthorized access is prevented by the following features and functions provided by this protocol [11] :

- AES - based Encryption with 128 bit key length provides both data integrity and security. Data integrity ensures that they have not been changed, removed or added by unauthorized node [12]. Data security prevents to view sent information by using ZigBee sniffer. It is achieved by encrypting the data payload field. ZDO initializes Security Service Provider (SSP) which is used by NWK and APS layers.
- Message Timeout – frame counter in the message frame is incremented after each transfer, which counteracts so called replay attacks. Even if some encrypted message has been sniffed, it cannot be re-used later.
- Access Control List - which contains MAC addresses of nodes which could be in the PAN network.

- Accept or Reject Join Request – additional security mechanism could be implemented. Because in most cases user decides if he want to add a new node to the PAN, there could be the hardware accept button which enables it – for example in the coordinator node or remote PAN management center.

2.4. 802.3: Ethernet Standard overview

Ethernet is one of the LAN technologies, intended for a high-speed communication over a small geographic area [13]. Ethernet is covered by the IEEE 802.3 standard. This standard describes the two lowest layers of the OSI reference model: Data link and Physical layer.

Data link layer is further divided into two sublayers: MAC-client and MAC. Without going into much detail, the Ethernet MAC sublayer is responsible for data encapsulation, initiation of frame transmission and detecting failures. During reception, frames are parsed and errors are detected. MAC Ethernet frame format is illustrated in Figure 2.19. It consists of the following fields:

- SFD – Start-of-Frame Delimiter.
- DA – Destination Address, hardware MAC address of the target station. Should be unique worldwide.
- SA – Source Address, hardware MAC address of the station sending frame.
- DFL – Data Field Length, the number of octets carried by the DF field.
- DF – Data Field, carries information from LLC sublayer.
- PAD – Padding, additional octets which are used, if the DF field has not minimal number of octets.
- FCS – Frame Check Sequence, used to detect errors in the receiver, calculated without Preamble and SFD fields.

Figure 2.19. Ethernet MAC frame format.

Physical layer is divided into Physical medium independent layer with MII interface, and Physical medium dependent layers with MDI interface. Physical medium independent layer is often integrated as a microcontroller peripheral. For example the STM32F4

microcontroller used in the project has the Ethernet peripheral with MII interface. Physical medium dependent layer is usually realized in a separate chip called: physical layer transceiver, and allows the system with MII interface to work over different kinds of cables such as twisted-pair and optical fiber.

Ethernet uses carrier sense multiple access with collision detection (CSMA/CD). A station can detect if there is a transmission in the channel, and sends its frames only when there is no traffic. Collision between frames from many sources can occur only during the beginning of transmission due to the propagation delay over the cable. Pure CSMA leads to waste of time, because after collision frames from many stations interfere with each other throughout their duration. Collision detection mechanism shortens collision period and allows faster return to the collision-free transmission.

There are many Ethernet versions available and their general naming standard is as follows:

Xbase-**Y**, where:

X – maximum data rate in Mbps,

Y – physical medium abbreviation.

For example:

- 10Base-T: offers 10 Mbps and uses twisted-pair cable,
- 100Base-T (Fast Ethernet): offers 100 Mbps, also using twisted-pair cable,
- 1000Base-SX (Gigabit Ethernet): provides 1Gbps over the Multi-mode fiber.

2.5. Smart grid concepts.

A smart grid is an emerging concept of a next-generation power grid network, which integrates information, telecommunication and power technologies to deliver energy in the optimal way from many sources to many destinations. The main goal of a smart grid is to improve reliability of the energy network by providing alternative routes after grid fail, and efficiency due to distributed generation and energy management systems. Distributed generation means that many small sources generate electricity and they are situated in residential and industrial areas. In such grid renewable energy sources, such as photovoltaic panels and wind turbines are connected to the existing network. Distribution algorithms find the most efficient way to reduce the transmission losses.

The very important aspect of a smart grid is interaction with energy consumer, who can monitor dynamically changing prices of electricity during the day and postpone some less important energy consuming activities to time when prices are lower. This leads to lower bills

and to compensation for the day energy demand curve. Excessive overproduced energy can be stored in batteries of electrical vehicles and gave back at the time of increased power demand.

To allow these features smart metering has to be applied. Smart meter recognizes and details electrical consumption and, for example, relays information to central monitoring station [14]. Wireless sensor networks such as ZigBee, are ideal for that purpose. The ZigBee Alliance is currently pushing their technology to become a standard solution both for smart metering, home and building automation systems. Thanks to constant grid monitoring as well as smart sensors placed along power lines and substations, when damage occurs, the problem is automatically reported, the damaged grid segment is isolated and distribution system may automatically re-route grid topology. The place of fail can be precisely identified and the relevant departments are informed about the problem to solve it as soon as possible. This feature is often called self-healing.

To fully deploy the smart grid idea, every component of the grid has to be prepared: power generators, distribution and transmission power lines, substations, and consumers. New applications emerge especially in substations, which are used to transform voltage (from high to low or low to high level) and often integrate grid fault protection and switching functions. There are international initiatives that focus on substation automation like the KIC-ActiveSubStations project [15].

The EU's “Third Energy Package” includes Electricity and Gas Directives which requires the EU Member States to ensure the implementation of intelligent metering systems. Smart metering should be fully deployed by 2022, and by 2020 80% energy consumers have to be equipped with smart meters [16].

2.6. IEC 61850 standard overview.

2.6.1. Basic concepts of IEC 61850 and standard documents description.

Global demand for electricity is constantly increasing. According to Frost & Sullivan research, the average annual growth rate in power generation is 2.7% to the 2020 [17]. Nowadays significant funds are invested into electrical industry to improve performance of power generation and distribution, reduce production costs, and increase reliability and safety of electric grid systems. Substation automation, which is in scope of IEC 61850 standard, is one of the aspects under consideration.

The main goal of the IEC 61850 standard is to provide interoperability between Intelligent Electronic Devices (IEDs) that are main building blocks of substation automation systems. Interoperability enables easier configuration, higher reliability, lower costs and safety.

IEC 61850 defines object oriented approach to device building and to their interoperation in a network. It defines abstract communication models that are independent from particular hardware platform and communication interfaces, that makes the standard resistant to future changes in communication protocols. It allows to use the standard with any transmission media, that meet the basic protocol requirements. The whole substation automation system and each device in the IEC 61850 network is described in a standardized way using Substation Configuration Language (SCL), which is described in IEC 61850-6 document [18].

IEC 61850 allows for interoperation between IEDs produced by different manufacturers. An IEDs should:

- provide possibility to connect to a common bus, using common interface,
- understand information posted by another IEDs,
- if required, work together on one task without disturbing each other.

Moreover interoperability in every configuration requires [19]:

- that communication allows any device to perform any function, but this does not mean that any device must serve any function,
- that Substation Automation System (SAS) functions and their behavior during communication should be described independently of a device,
- no redundancy in function description,
- openness to the future innovations.

2.6.2. Data and communication models.

IEC 61850 models the functions performed by real devices in SAS. The common information and functions available in the physical devices are described in the standard using the modeling approach. Figure 2.20 illustrates a real device, which is modeled as a logical device containing logical nodes (LN). In this example the logical device is the bay unit, and it contains multiple logical nodes, which are virtual representations of the concrete functions the bay unit performs. For example a circuit breaker is virtually represented by the XCBR class of a LN. Each logical node contains a list of data with dedicated data attributes [20], for instance XCBR class has data defining its position (on/off) using a boolean value.

An IEC 61850 device model is described by a configuration file written in the SCL language described in the reference [18]. It contains definitions of logical devices, logical nodes and information exchange settings. To provide interoperability each device, even from different manufacturers has to be defined by using the same semantic, for this reason standardized configuration language had to be defined.

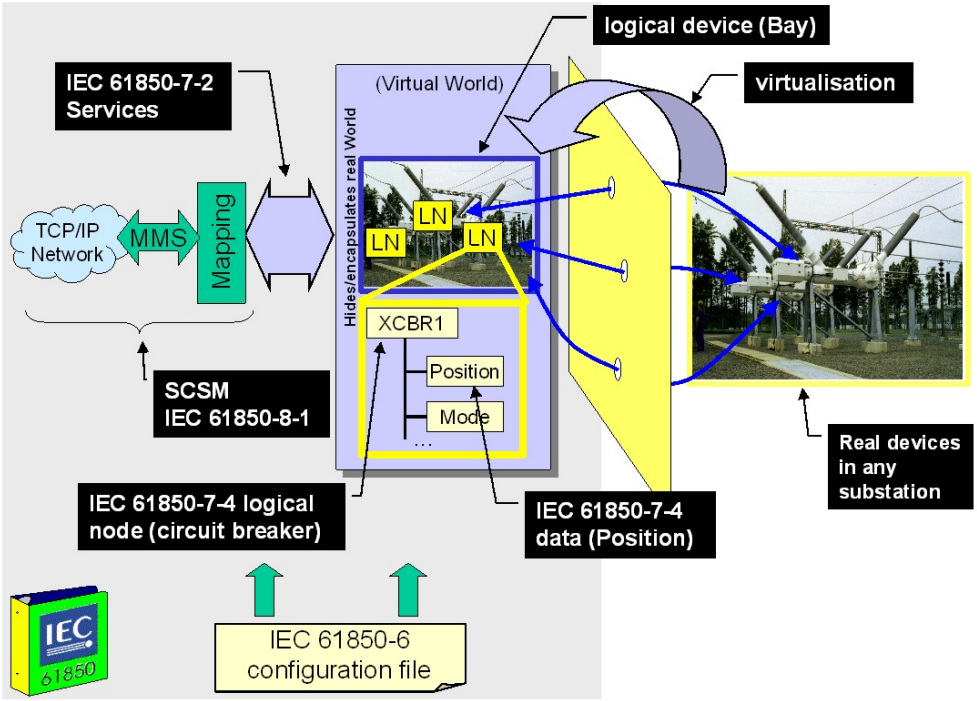


Figure 2.20. The modeling approach in the IEC61850. The Figure comes from [20].

The hierarchy in the data model is illustrated in Figure 2.21. Substation Automation System consists of physical devices (IEDs) connected to the transmission medium, for example Ethernet cable. Each of them contains unique IP address, and works as a server. IEDs are collections of logical devices, which further divide into logical nodes. Possible logical node classes are defined by the standard.

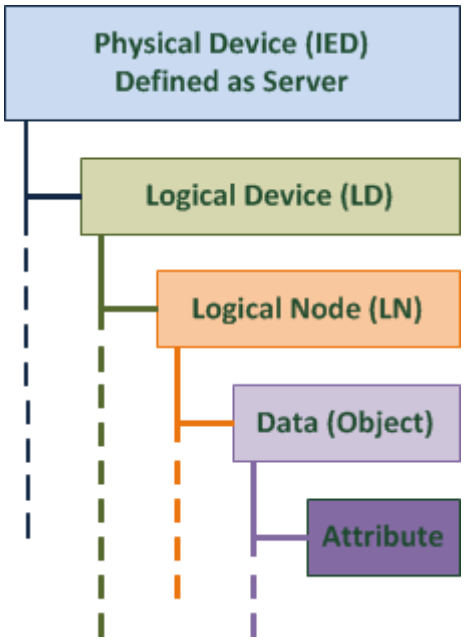


Figure 2.21. Hierarchy of the data model defined in the IEC 61850 standard.

The objects are referenced by their position in the hierarchy in the following way:

LD / LN $ DO,

for example:

BayUnit_2 / XCBR_1 $ Pos,

refers to a position (Pos) of the circuit breaker 1 (XCBR_1), which is localized in the Bay Unit 2 (BayUnit_2).

Apart from the data model IEC 61850 defines the information and information exchange models, which are described in the part 7-2 of the standard [21]: “Basic information and communication structure – Abstract communication service interface (ACSI)”. ACSI defines the abstract interface between a client and a remote server. It also describes fast event distribution method using publisher/subscriber mechanism. These interfaces are used for:

- real-time data exchange,
- file transfer,
- discovery of data types,
- group control,
- reporting the events,
- remote device control.

The ASCI information and information exchange models correspond with the Common data classes described in the part 7-3 [22] and 7-4 [23].

The information modeling classes are as follows:

- Server – is the container of other ACSI models. Represents the behavior of a device, which is seen by external systems. Server can communicate with the client using client/server mechanism and send real-time information to peer devices, such as Sampled Values and GOOSE.
- Logical device (LD) – represents a group of functions with their input and output information,
- Logical node (LN) – represents the information produced and consumed by a single function of a logical device,
- Data objects – provide parameters to describe a specific function, such as time stamp, analogue value etc.

Each information model is defined as a class.

Information exchange model describes service models, in addition to Logical Node and Data classes [21]:

- Data Set – groups data objects and their attributes from different logical nodes. This group can be directly accessed as a whole, and could be send using sampled values, GOOSE messages, reports and logs.
- Substitution – this model allows to replace a process value by another, for example more precise value.
- Setting group control – describes the switching from one set of logical node settings to another one.
- Report control and logging – allows to generate reports and logs from the specified data-sets. An ICD file (in the Report Control Blocks) describes parameters for this model such as: trigger options (on data change, update, integrity poll, quality change), time stamp, sequence number and information about reasons for report generation. Moreover reports can be sent in specified time interval.
- Control blocks for generic substation event (GSE) – defines the fast and reliable distribution of the input and output data values to the physical peer devices.
- Control blocks for transmission of sampled values – which are periodically transferred samples that have to be quickly delivered.
- Control – describes services to control devices in the SAS.
- Time and time synchronization.
- File system – definition of file management and transmission using client/server mechanism.
- Tracking – provides the interface to track control, configuration and exchange services. It is used for diagnosis the system.

Each model has different requirements for a transmission method (client-server or publisher-subscriber), delivery time and length of data, so they are mapped to different communication protocols described in the next subsection.

2.6.3. Real Protocols and IEC 61850.

IEC 61850 communication services are mapped into different communication protocols depending on their requirements. Two of the requirement parameters, which are used further are: transmission and generation time, which are illustrated in Figure 2.22.

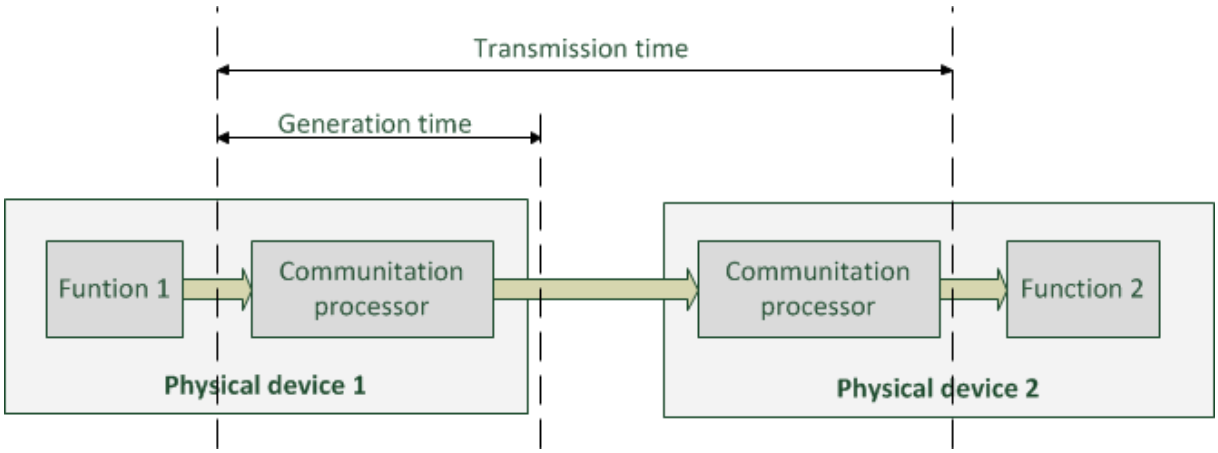


Figure 2.22. Illustration of transmission and generation time in IEC61850-5 [19]

In the part 5 of the standard [19] the seven types of messages are defined:

- Type 1 – fast messages which contain simple binary code or command such as: “turn on”, “turn off”, “close”, “start”, “stop” etc.
- Type 2 – medium speed messages, for which generation time is important, but the transmission time is less demanding.
- Type 3 – low speed messages which demand time stamps.
- Type 4 – raw data messages containing values from analog-to-digital converters and digital converters, which are generated by different IEDs.
- Type 5 – File transfer messages, to transfer big files divided into smaller pieces.
- Type 6 – Time synchronization messages used to synchronize internal IEDs clocks in the SAS.
- Type 7 – Command messages with access control used to transfer control commands, which requires high level of safety and data integrity.

Each type of message has different requirements on delivery time. Table 2.4 presents these requirements [19] :

Type of message		Time requirements – transmission time	
		P1	P2 / P3
1A	fast messages (off command)	<10 ms	<3 ms
1B	Fast messages (others)	<100 ms	<20 ms
2	medium speed messages	<100 ms	<100 ms
3	low speed messages	<500 ms	<500 ms
4	raw data messages	<10 ms	<3 ms
5	File transfer messages	Not critical (typically > 1000 ms)	
7A	Command messages with access control	<500 ms	<500 ms
7B	Command messages with access control (for special tasks)	<10 ms	<3 ms
		Time requirements – accuracy	
6A	Time synchronization messages (control and protection)	Accuracy: ±1 ms	±0.1 ms
6B	Time synchronization messages (measurements)	Accuracy: ±25 µs	Accuracy: ±4 µs / ±1 µs

Table 2.4. Time requirements for different types of the message [19].

The standard defines so called performance classes P1,P2, P3. Performance class P1 concerns distribution bay, P2 concerns transmission bay unless the client wants otherwise and P3 concerns transmission bay with elevated synchronization requirements.

Figure 2.23 illustrates communication stacks used by different types of IEC 61850 messages. GOOSE and SV messages are time-critical – to meet the requirements of a type 4 and 1, they have to be mapped directly to the low-level Ethernet layer. In this case frames can be shorter, and prepared more quickly [24]. The messages of type 2, 3 and 5 are mapped to MMS protocol based on TCP/IP protocol. The Time synchronization messages are sent using SNTP protocol based on UDP/IP protocol.

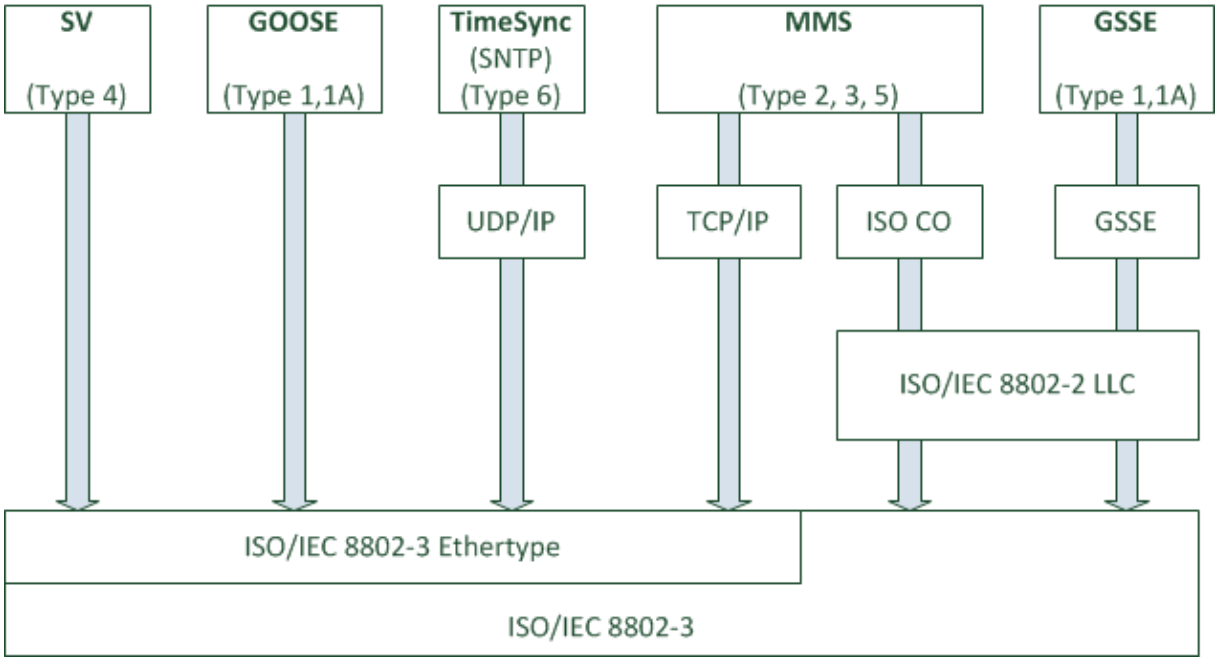


Figure 2.23. Communication stacks used by the IEC 61850 standard [24] .

2.6.4. Substation Configuration Language.

Part 6 of the IEC 61850 standard introduces file format for describing configuration, parameters, function and relations between IEDs, and for communication system configuration [18]. The language used in this file format is XML-based Substation Configuration Language (SCL). IEC 61850 defines six types of SCL files - they are shown in table 2.5.

Name	Extension	Description
System Specification Description	.SSD	It describes the functions of the SAS and required types of the LNs, but without specific IED description.
IED Capability Description	.ICD	It describes the functional capabilities of one IED type, so it should have only one IED section.
System Exchange Description	.SED	Description of the interface for another project to use the specified project.
Instantiated IED Description	.IID	Description for a single IED tailored to the specific project. Includes project specific name, addresses and data model. It can be used for example for IEDs which have number of LN instances dependent on the project settings.
System Configuration Description	.SCD	It describes which IEDs are used in the project, data flow between them and required DataTypes Templates.
Configured IED Description	.CID	Description of the communication part of an instantiated IED. It contains only the information about IED which IED configurator should know.

Table 2.5. Description of different types of SCL files.

The SCL file is divided into five sections:

- Header – identification of SCL configuration file, information about version, and options for the mapping of names to signals [18].
- Substation description – description of a functional structure of a substation.
- IED description – configuration of an IED, logical devices, logical nodes and their data objects descriptions.
- Communication system description – describes which IED access points are connected to a network.
- Data type templates – used to define instantiable logical node types.

2.6.5. Substation Model.

The IEC 61850 standard relates to a Substation Automation System (SAS). Substation Automation (SA) is a system that remotely monitors, controls and coordinates energy distribution components installed in a substation [25]. Main functions of SA are switch control, data monitoring and protection. These functions are broken by IEC 61850 into sub-functions which are associated with logical nodes. Each IED can perform one or many sub-functions, so each IED is a collection of logical nodes [25]. There are currently other protocols that can be used for substation automation, for example Modbus, Modbus Plus, DNP 3.0, IEC 60970, but they do not support interoperability among IEDs to such extent as IEC 61850.

An example of such SAS is presented in Figure 2.24. It shows an architecture with three levels: process, bay and station. Station bus is used to communicate between the station and the bay level as well as to communicate between equipment at the same level. Station level devices are for example station computer with HMI (Human Machine Interface) used as the interface to the operator and the station gateway to connect to, for example a network communication center [26]. Bay level equipment consists of protection and control IEDs, used to collect data from the process level equipment and to control its behavior.

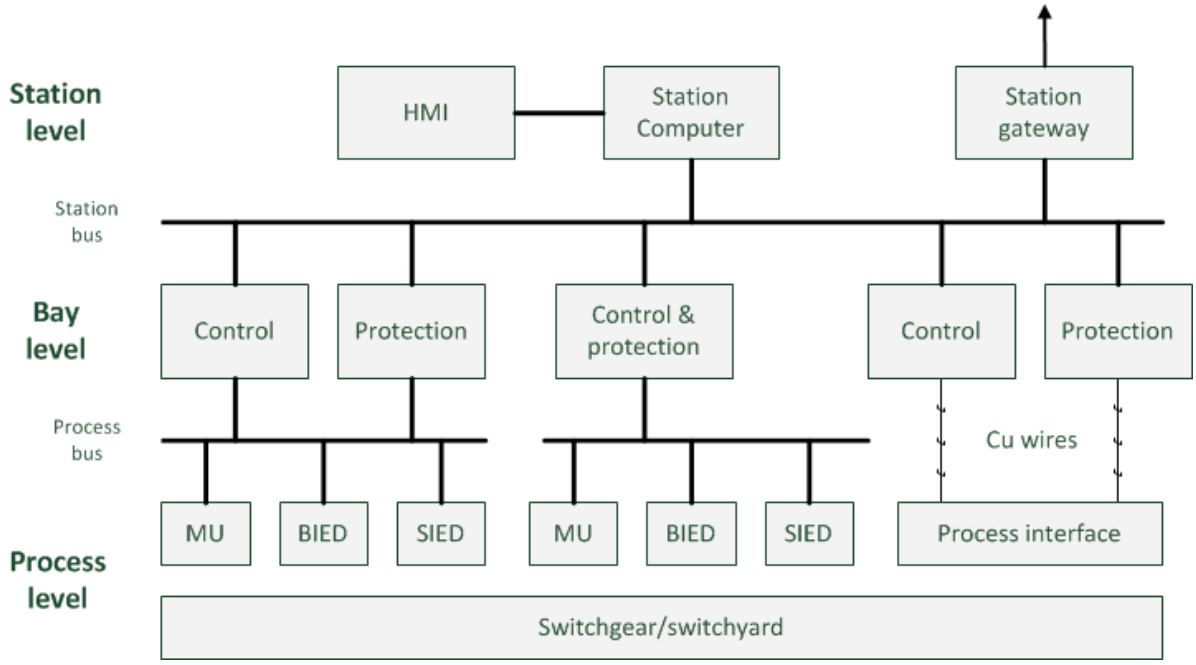


Figure 2.24. Example of a Substation Automation structure [26].

Process level equipment consists of merging unit IED (MU), breaker IED and switch IED. They serve as remote input / outputs, sensors and actuators [25]. Between the bay and the process level equipment there are process buses, which have higher real time requirements than station bus.

3. DESIGN OF A ZIGBEE-TO-ETHERNET BRIDGE.

3.1. Purpose of a ZigBee-to-Ethernet bridge.

The purpose of a ZigBee-to-Ethernet bridge is to connect ZigBee and Ethernet (TCP/IP based) network, as shown in Fig. 3.1.

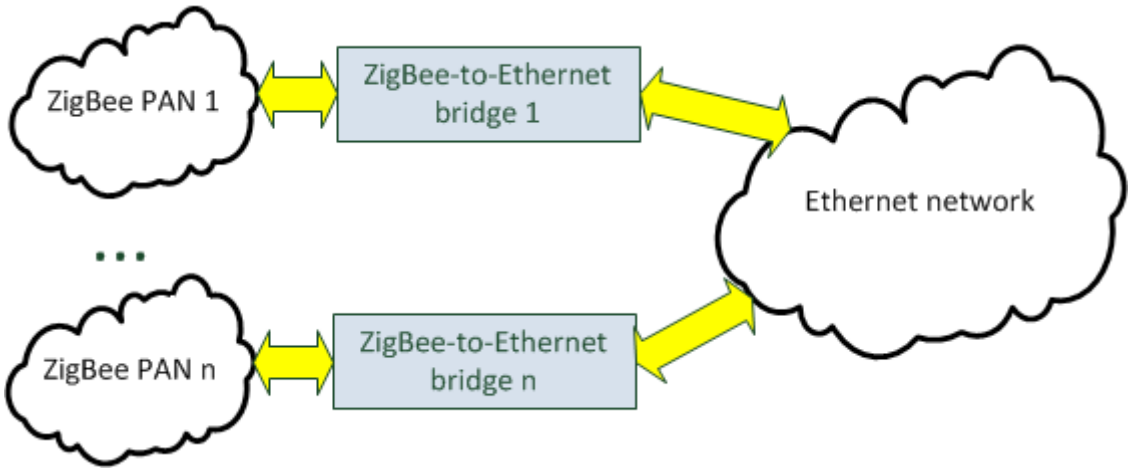


Figure 3.1. Vision of the system, which consist of many ZigBee-to-Ethernet bridges managing many PANs.

Such device should provide an interface for user to view data from ZigBee PAN network. It is assumed that the ZigBee nodes are equipped with sensors and actuators, that interact with the environment. The bridge should act as a data server, providing data from PAN network measurements as well as an interface to control the PAN nodes. This gives a possibility for example to store collected data in an external database to monitor changes of measured values.

A ZigBee-to-Ethernet bridge can meet requirements of industrial and home applications, for example:

1. ZigBee nodes can be used to monitor environmental and industrial process parameters inside a factory or even be used for controlling these processes. The nodes may be grouped into separate PAN networks, each covering for example a single process hall, a selected group of machines etc. The ZigBee-to-Ethernet bridge can be used to integrate these networks to existing factory intranet, which is often an Ethernet based TCP/IP network. Due to security reasons this data is usually not available outside the factory intranet [27].
2. In case when the data provided by the ZigBee nodes is not confidential, the ZigBee-to-Ethernet bridge can be used to connect PAN networks directly to the Internet. For example a city network gathering data about air pollution can connect directly to a

database hosting server, allowing visualization of the measurements.

3. ZigBee also becomes a standard in home automation, providing tools for controlling light, heating, ventilation and air conditioning (HVAC) and other appliances. For the end user the Internet connectivity allows to control such system from remote location, which actually can be any place in the world.

3.2. Requirements.

The bridge works as a PAN coordinator, so it has to process and maintain more data than other ZigBee nodes in the network. Data from PAN, that is interesting for other systems , such as network addresses of the nodes, state of the nodes, measurement data etc. are to be visible via TCP/IP based network. Taking all this into consideration, requirements for the bridge can be formulated:

- it should work in the unlicensed frequency band at 2.4 GHz, because of higher bit rate and 16 channels instead of 1 channel at 868 MHz. Although the sensitivity at 2.4 GHz is lower than at 868 MHz, but the distances between devices in typical PAN network are the order of several to tens of meters. 868 MHz band has got restrictions associated with the channel busy time. If there is a distortion in this frequency band, no other can be selected and communication between nodes in not possible. At 2.4 GHz another channel can be selected providing more reliable connection. Even if all WiFi channels at 2,4 GHz are occupied, there are four safe IEEE 802.15.4 channels at 2.425, 2.450, 2.475 and 2.480 GHz available.
- Output power of the ZigBee transceiver should be at least 3 dBm, and sensitivity at least -85 dBm, and hence a minimal link budget should be equal or greater than 88 dB.
- Both the microcontroller and radio module should be low power. Current consumption of the ZigBee transceiver should be lower than 50 mA, even at TX mode.
- Microcontroller may operate under the control of a RTOS, and should have enough processing power and flash memory for TCP/IP and ZigBee stack components, Web Server, and task for communication with MySQL client on the remote server.
- Ease of use for components and software tools.
- Free software tools.
- Low bill of materials cost.

3.3. Proposed solution overview.

3.3.1. Design choices.

The simplest way to provide web server, and data base software support is to use

single board computer with, for example, embedded Linux operating system. It provides API for TCP/IP stack, embedded data base clients, and full web server support, providing high-level API that allows to focus only on creating web page, instead of handling HTTP requests from a API level. However, this solution has two drawbacks: cost, and high energy consumption.

Taking all this into consideration, STM32F4 microcontroller and CC2530 ZigBee transceiver are selected as main hardware components of the Bridge. They are discussed in the subsections 3.4.1 and 3.4.2.

3.3.2. Architecture of the system with a ZigBee-to-Ethernet bridge.

Figure 3.2. shows architecture of a system in which a ZigBee-to-Ethernet bridge operates. This system may consist of multiple bridges connecting separate PAN networks, a web server, a MySQL database sever and a personal computer with the Internet access.

Each ZigBee-to-Ethernet bridge has its own IP address. Using web browser user can log into every bridge and change its settings. He can also see measurement results from sensors within the PAN. There can be also possibility to control PAN devices using standard ZigBee Cluster Libraries.

In the system there is an web server with Java web application, which enables user to view and control all ZigBee PAN networks in which there is a ZigBee-to-Ethernet bridge. This web application has MySQL client functionality, and can store data from each sensor, from each PAN, into the database. This enables to see changes of measured values during long periods, and to have documented all events which user cares about.

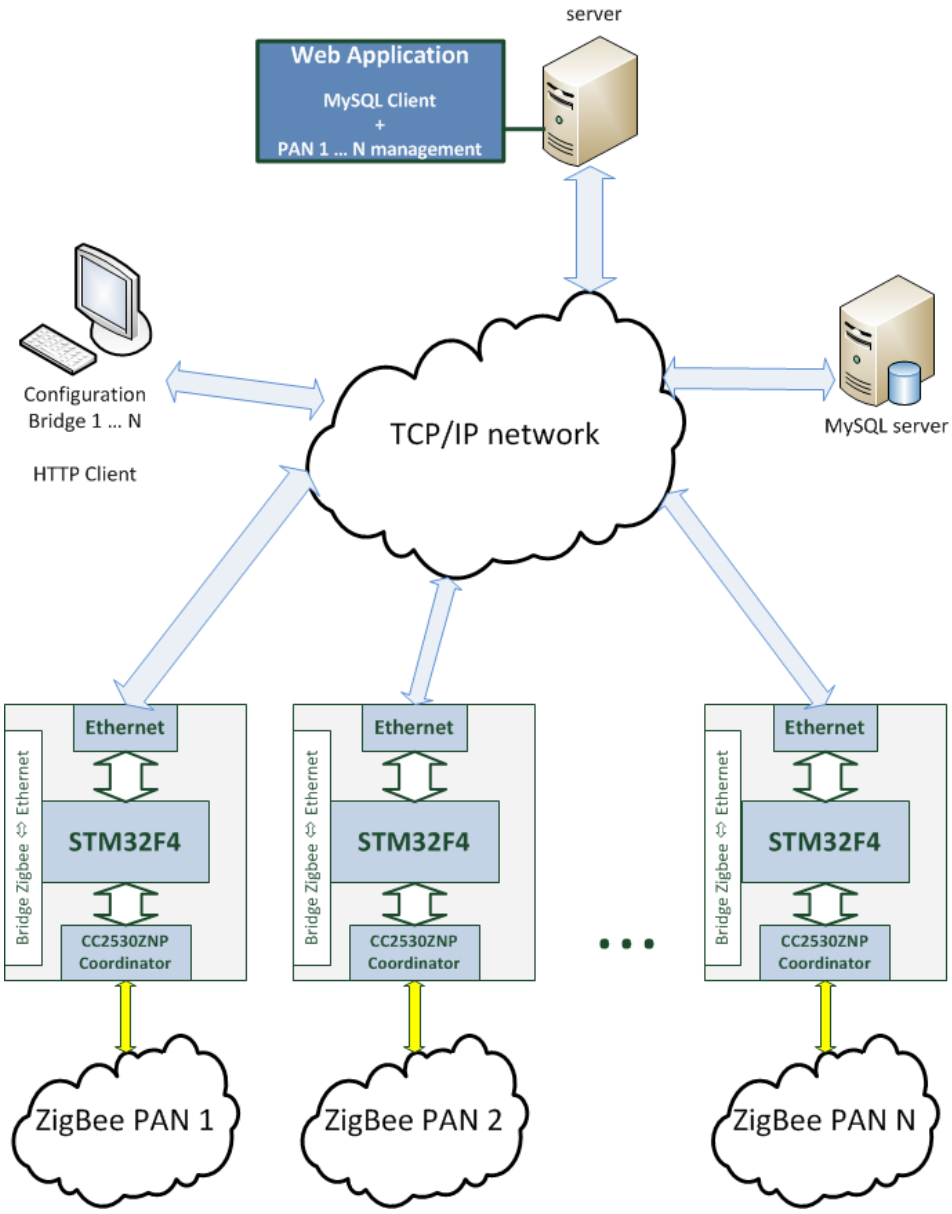


Figure 3.2. Architecture of the system, which consist of many ZigBee-to-Ethernet bridges managing many PANs.

3.3.3. Building blocks of a ZigBee-to-Ethernet bridge.

Figure 3.3 shows main software and hardware building blocks of the ZigBee-to-Ethernet bridge, which are described in sections 3.4 and 3.5.

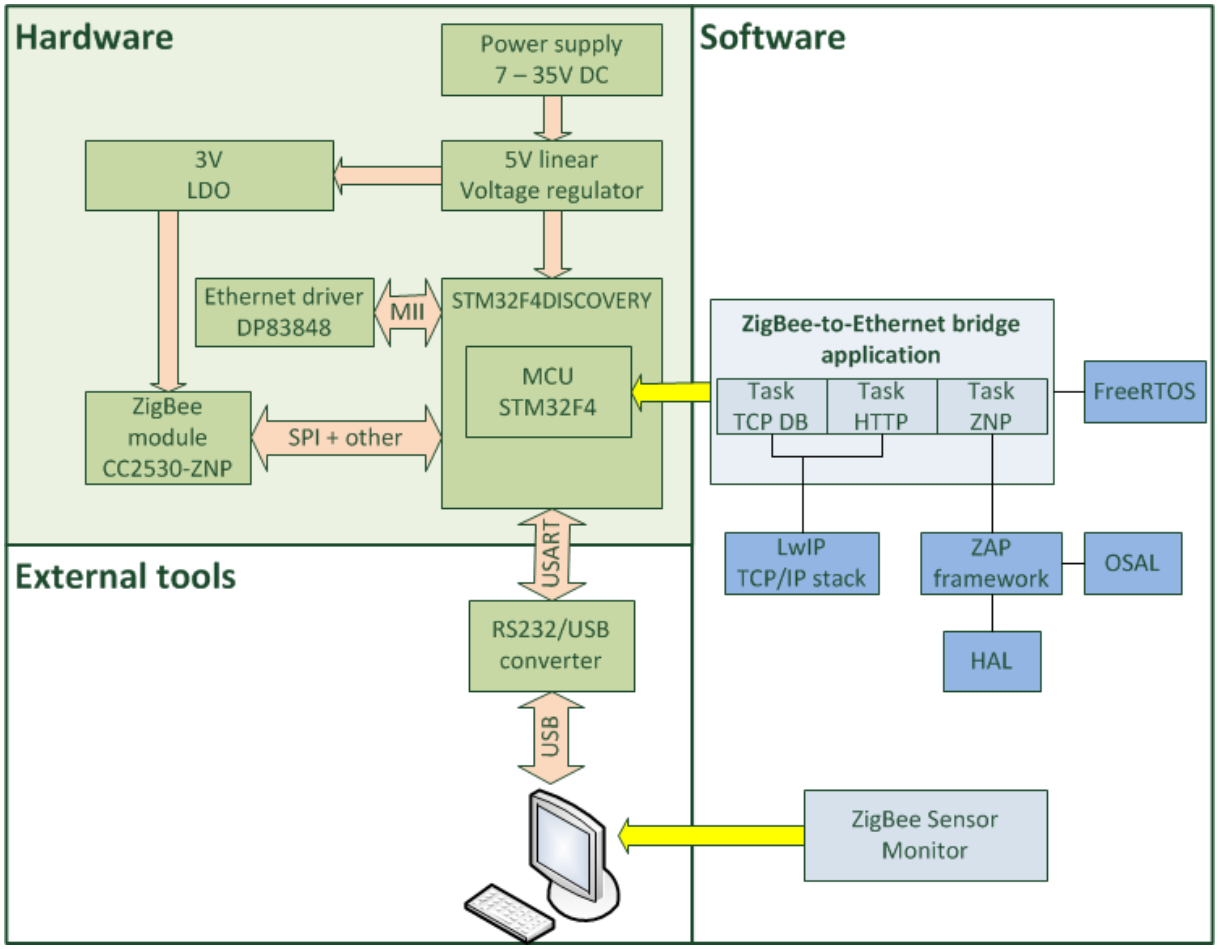


Figure 3.3. Block diagram of the hardware and software for a ZigBee-to-Ethernet bridge.

3.4. Hardware design.

3.4.1. STM32F4 microcontroller description.

STM32F407VGT6 microcontroller has been selected as a main controller of the bridge. It has got the latest ARM Cortex-M4 core with MPU, FPU, 1.25 DMIPS/MHz, 1 MB of Flash memory and 192 kB of SRAM memory[28]. It integrates low power capabilities, and powerful computing power in a single chip. Maximum core clock frequency is 168 MHz, which gives up to 210 DMIPS. Microcontroller has the following communication peripherals[29]:

- 10.5 Mbps USART,
- I2C,
- up to 37.5 Mbps SPI,
- CAN interface,
- SDIO interface,
- USB 2.0,
- 10/100 Ethernet MAC with dedicated DMA and MII/RMII interface.

Three of them are used in the project:

- SPI to communicate with the CC2530-ZNP module,
- USART to communicate with WSN visualization tool provided by the Texas Instruments company,
- Ethernet to connect to the Internet.

STM32F4 microcontroller integrates a 16-stream DMA controller, which can operate in three modes:

- memory to memory,
- memory to peripheral,
- peripheral to memory.

DMA is used in the bridge to offload the CPU from handling SPI transmission to and from CC2530-ZNP. For sending larger amount of data this method is better than using polling or interrupt-driven transmission, especially because the selected MCU has only 1-byte long hardware SPI buffers. The DMA transmission is used also while sending and receiving Ethernet frames.

STM32F4 microcontrollers contain so-called multi-AHB bus matrix. Several devices, such as Ethernet peripheral, DMA controller and the CPU act as bus masters, others such as FLASH and RAM memory controller and peripherals act as bus slaves [28]. Because of

separate buses and 3 SRAM memories with separate ports several high-speed peripherals can work simultaneously, and at the same time data processing can be performed by the CPU. Moreover it enables event driven architecture of the software in which CPU is in a low power mode most of the time, and is awaken by events from peripherals which carry out most time consuming processes of the program.

The capabilities offered by this microcontroller are sufficient to implement the expected functionality and the cost of a single chip is low.

3.4.2. CC2530-ZNP solution for ZigBee networking.

CC2530-ZNP is a low power ZigBee system-on-chip (SoC) module based on CC2530F256 Texas Instruments device which runs a dedicated ZigBee stack. It contains an IEEE 802.15.4 compatible radio transceiver and a 8051 microcontroller. The module can interfaced with SPI, UART or USB from an application processor. Architecture of such system is presented in Figure 3.4.

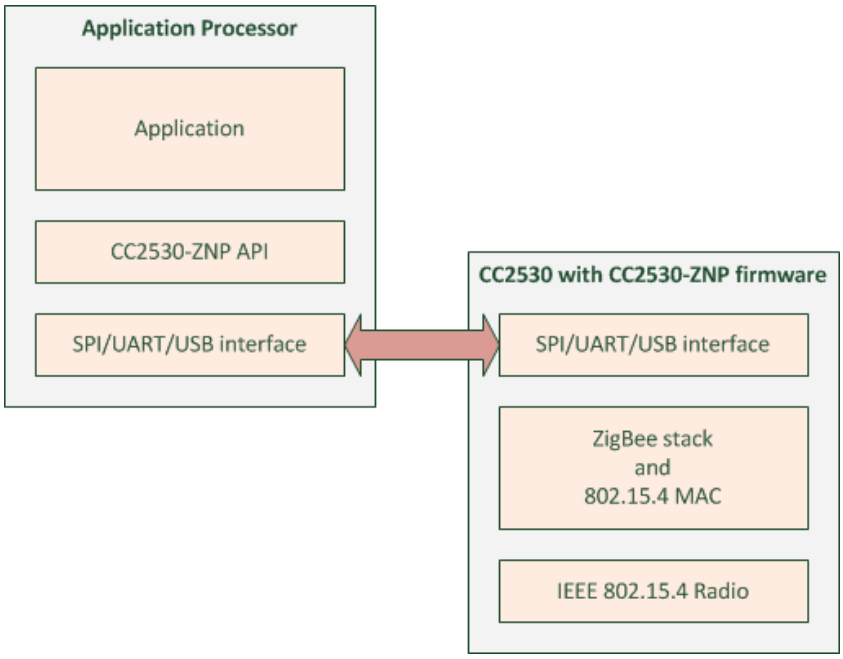


Figure 3.4. Architecture of a system with an application processor and CC2530-ZNP [30].

CC2530F256 has 256-KB Flash memory, which is appropriate value for implementing full ZigBee PRO stack and application layer using ZigBee cluster libraries provided by the Texas Instruments. This module integrates peripherals listed bellow:

- Five-channel DMA,
- Operational amplifier and ultra low-power comparator,
- IEEE 802.15.4 MAC timer, general-purpose 16-bit and 8-bit timers,
- CSMA/CA Hardware Support,

- RSSI and LQI indication,
- battery monitor,
- temperature sensor,
- 12-bit ADC,
- two USARTS,
- 21 GPIO pins.

Very few external components are needed to make it work. Typical power consumption in active transmission mode with 1 dBm output power and CPU in the idle state is 29 mA. A ZigBee device integrating this chip can be cheap, small and consuming very little energy.

Main radio features are as follows:

- programmable output power up to 4.5 dBm,
- frequency range: 2.394 to 2.507 GHz with 1 MHz steps and 5 MHz space between channels,
- baud rate: 250 kbps,
- maximum RX/TX and TX/RX switching time of 192 μs,
- receiver sensitivity (PER = 1%) is typically -97 dBm [31].

CC2530F256 hardware is appropriate for working as a part of the ZigBee-to-Ethernet bridge. To make it work as a ZigBee network processor, the ZStack-ZAP-MSP430 firmware must be downloaded from Texas Instruments ZigBee protocol stack website and installed in the CC2530 flash memory. This can be done by using either a SmartRF05 Evaluation board or CC-debugger – both provided by the chip vendor.

After uploading firmware CC2530-ZNP works as a peripheral processor handling ZigBee communication. That solution offloads the main application microcontroller – in this case STM32F4.

3.4.3. Schematic and Printed Circuit Board project.

Appendix D presents the electrical schematic of the ZigBee-to-Ethernet bridge. The schematic and the PCB have been designed using CadSoft EAGLE PCB Design Software. The schematic is divided into three sheets:

- main controller & CC2530-ZNP,
- Ethernet PHY,
- power supply.

3.4.3.1. STM32F4DISCOVERY hardware modifications.

The main application microcontroller is placed on a modified STM32F4DISCOVERY board, which is connected to the mother board through gold pin connectors. This is a cost effective solution and it integrates ST-link programmer and all components required to work, such as: crystal oscillator and linear voltage regulator. It allows for a fast prototype design.

However certain signals that needed outside of the module are already used by the on-board componets. That's why some of the integrated circuits had to be unsoldered:

- LIS30DL – 3-axis accelerometer,
- CS43L22 – audio DAC with integrated speaker driver,
- MP45DT02 – MEMS audio sensor.

This made the MII interface available to connect to an external Ethernet PHY.

3.4.3.2. Ethernet circuit description.

MII interface signals, made available after STM32F4DISCOVERY modification are listed and briefly described in the Table 3.1.

Name	Direction	Description
MDIO	↔	MII Data Input/Output
MDC	→	MII Data Clock
RxD0 - RxD3	←	Rx Data
Rx_DV	←	Rx Data Valid
Rx_CLK	←	Rx Clock
Rx_ER	←	Rx Error
TxD0 - RxD3	→	Tx Data
Tx_CLK	←	Tx Clock
Tx_ERR	→	Tx Error
Tx_EN	→	Tx Enable
COL	←	Collision
CRS	←	Carrier Sense

Table 3.1. MII signals description [32].

These pins are used to connect DP83848CVV chip, which is a single port Ethernet physical layer transceiver. It performs all the functions needed to receive and transmit data over twisted-pair cables. Some of the transceiver's features are [33]:

- supply voltage of 3.3V,
- typical power consumption < 270mW,
- auto-MDIX function (automatic selection of the appropriate MDI pair for reception and transmission),
- supports both 10BASE-T and 100BASE-TX Ethernet protocols,
- full MII and RMII interface support,
- small 48-pin LQFP package.

The block diagram of a system using DP83848CVV is illustrated in Figure 3.5. Generally three types of interfaces between microcontroller and a PHY transceiver can be used: MII, RMII, SNI. RMII requires less pins, but the needed clock frequency is two times higher than in MII case. SNI also needs less pins, but the highest possible bit-rate is 10 Mb/s. A ZigBee-to-Ethernet bridge uses MII interface.

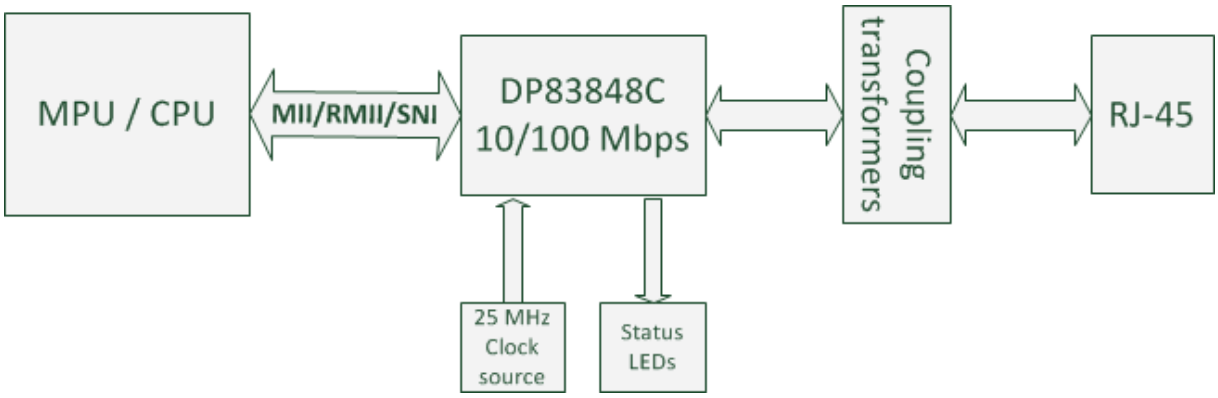


Figure 3.5. Ethernet physical layer system diagram [33].

To provide electrical isolation, which protects device connected to the Ethernet network, coupling transformers are needed between RJ-45 connector and DP83848 transceiver. In the design an integrated magnetic connector J011D21B is used, featuring a coupling transformer, RJ-45 connector, and status LEDs, all in a single element. The schematic diagram of the coupling circuit of a J011D21B is presented in Figure 3.6.

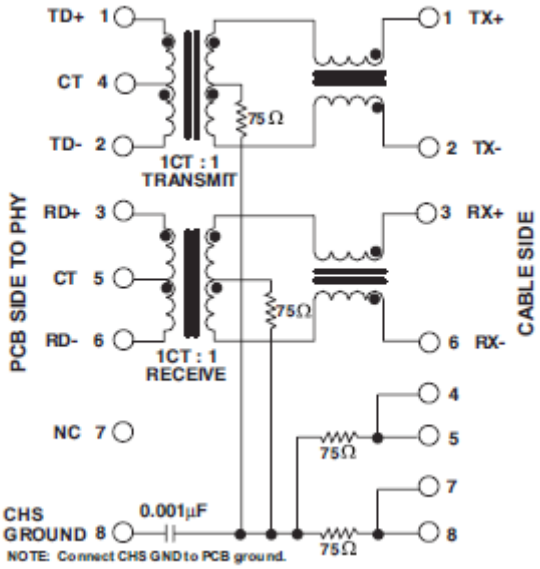


Figure 3.6. Coupling circuit of the J011D21B integrated RJ-45 connector [34].

In the Ethernet PHY circuit impedance matching is provided by precision 49.9 Ω resistors.

3.4.3.3. ZigBee transceiver circuit.

To facilitate the process of radio circuit for CC2530-ZNP design, the ready to use CC2530EM daughter board module has been used. It integrates CC2530F256 chip, antenna matching circuit with SMA connector, power supply decoupling components and two 2x10 pin, 50 mils raster connectors. The module is presented in Figure 3.7.

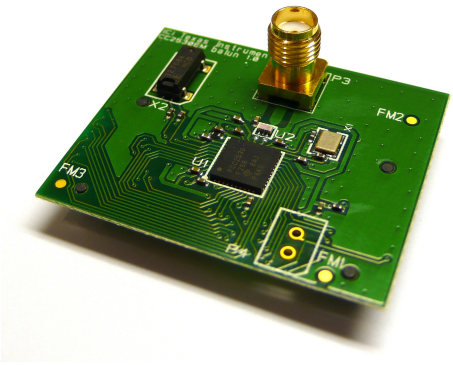


Figure 3.7. CC2530EM daughter board [35].

CC2530-ZNP is connected to a STM32F4 microcontroller via SPI interface with additional signals. SPI is a type of interface in which there is only one master and multiple slaves. Slaves can talk to the master only if he at first queries, therefore in many cases additional signals are added to indicate pending data in slaves. Figure 3.8 illustrates the

interface between CC2530-ZNP and STM32 microcontroller.

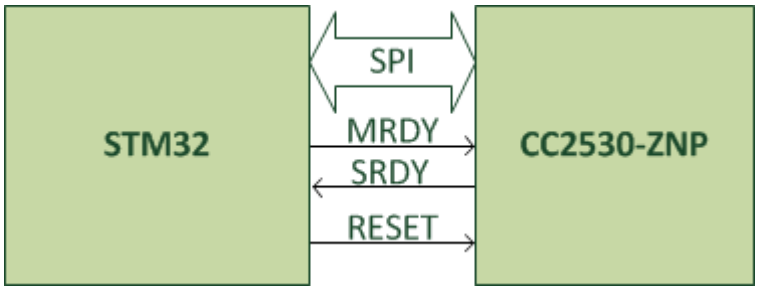


Figure 3.8. Interface between CC2530-ZNP and STM32 microcontroller.

Aside from standard SPI with MOSI, MISO, SCLK, CS signals, three additional lines are used: MRDY, SRDY and RESET. Additional signals allow to facilitate the higher layer SPI based protocol used to communicate with ZigBee module.

3.4.3.4. Power supply circuit.

The last schematic from appendix D presents the power supply circuit. The bridge is powered from 7 to 35V DC power supply. A diode between power input and L7805 linear voltage regulator protects against reverse polarity. As an alternative the power can be supplied from the USB connector via an onboard ST-link programmer. Because 3.3V linear voltage regulator integrated with STM32F4DISCOVERY board does not offer appropriate output current, the LF33CDT is used to provide a 3.3 V for the Ethernet PHY circuit. The PCB is designed so that MAX8860 can alternatively be used instead of LF33CDT to power the CC2530-ZNP transceiver circuit.

3.4.3.5. PCB layout of a ZigBee-to-Ethernet bridge.

Figures 3.9 and 3.10 show PCB layout of the bridge. The size of the board is 4700 x 3900 [mil]. Ethernet PHY circuit has been designed taking into account the length of the signal paths between DP83848CVV and J011D21B connector. These paths should not only be short, but also have equal length. Decoupling capacitors are placed close to the integrated circuits to better protect against voltage drops. The bridge hardware has been designed as an universal communication platform, which enables to connect not only to the Ethernet but also to CAN-bus, which is sometimes used in home-automation and industrial systems. The holes in the corners of the board are used to attach it to the chassis.

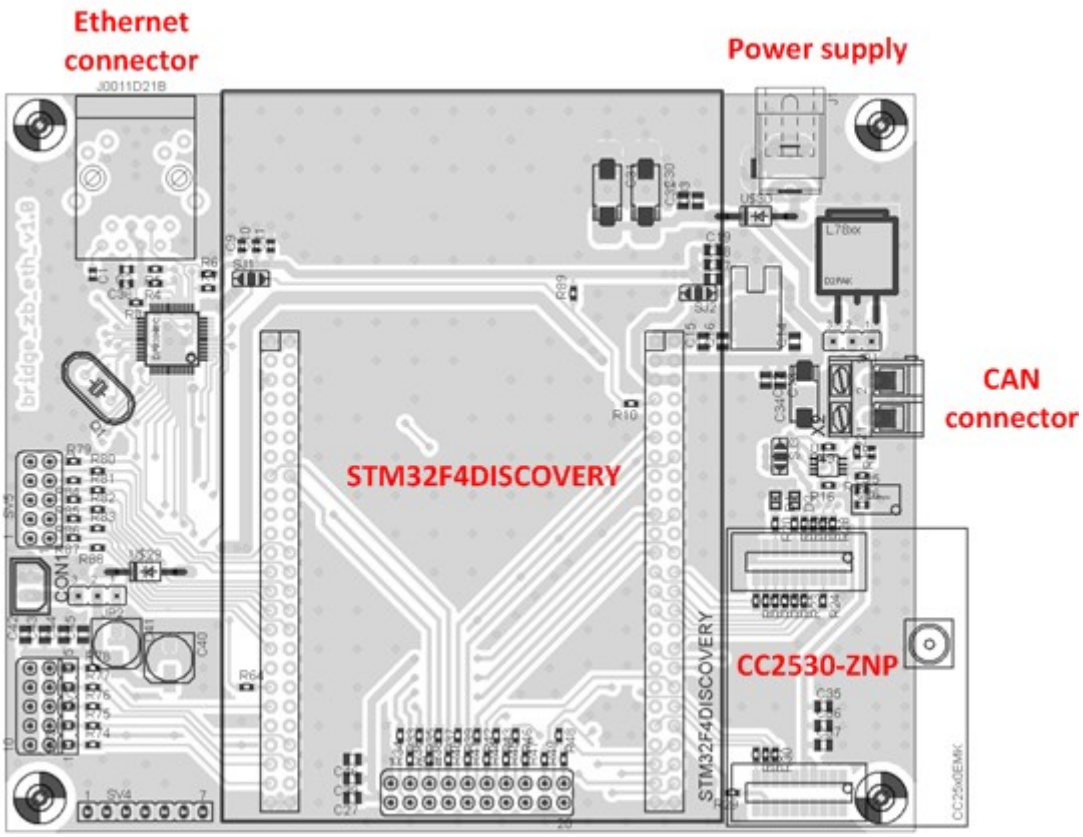


Figure 3.9. The ZigBee-to-Ethernet Bridge - PCB layout, top layer.

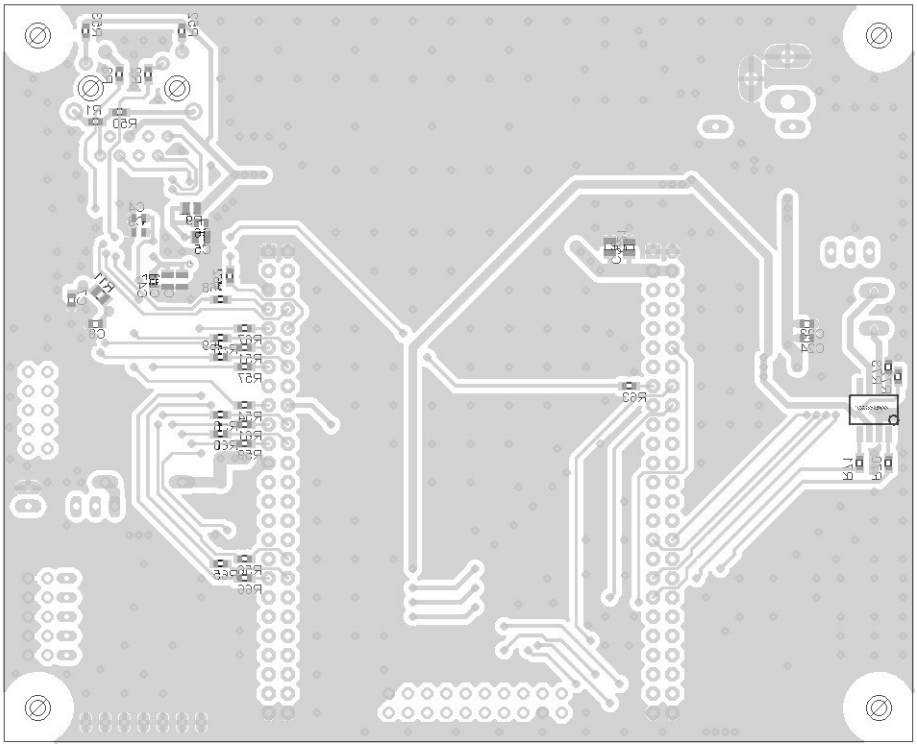


Figure 3.10. The ZigBee-to-Ethernet Bridge - PCB layout, bottom layer.

3.5. Software implementation.

Firmware for a ZigBee-to-Ethernet bridge has been developed using Atollic TrueSTUDIO for STM32 – embedded systems development tools [36]. It is an eclipse-based integrated development environment (IDE) that integrates the GCC toolchain. It supports ST-Link/V2 in-circuit debugger, which was used to program the STM32F4 microcontroller flash memory and for debugging.

The ZigBee-to-Ethernet bridge firmware relies on several software components, as illustrated in Figure 3.3. The main software building blocks are described in the following sections.

3.5.1. LwIP TCP/IP Stack.

To provide internet network connectivity TCP/IP stack libraries and drivers are required. LwIP is a open source TCP/IP stack designed for embedded systems, distributed under modified BSD license [37]. The main goal in designing LwIP was to make it suitable to run on machines with tens kilobytes of free RAM and about 40 kilobytes of Flash memory, providing full TCP/IP support. Hence the name – LwIP which means Lightweight IP. Protocols included in LwIP are as follows: IP, ICMP, IGMP, UDP, TCP, SNNP, DHCP, PPP and ARP.

LwIP is adapted to work both with and without an operating system. Three types of API are available:

- low-level API,
- netconn API,
- BSD-socket API.

3.5.1.1. Low-level LwIP API description.

Low-level API (also called “core”, “callback”, “native” or “raw” API) is designed to be used without an operating system and uses callback mechanism, which means that programmer has to write application callback functions that get called by the stack when events such as incoming data available, outgoing data sent, error notifications, poll timer expiration, connection closed, etc. occur [38].

3.5.1.2. Netconn LwIP API description.

Netconn API requires an operating system to work. Separate TCP/IP thread is created and it is responsible for processing all packets in the core of the stack. Programmer has to make sure that this thread is running and write his own application threads that communicate

with TCP/IP thread through message boxes and semaphores. In contrast to “raw” API, netconn API is sequential, which makes it easier to use the stack. Both netconn and raw API allow zero-copy functionality, which means that while sending or receiving data, only the address of a buffer with user data is sent to lower layers of the stack, without creating separate temporary buffers for communication. Because of that, the memory is saved and the program is faster, but the program has to be written carefully to prevent re-using created buffers before the end of the transmission.

3.5.1.3. BSD-socket LwIP API description.

Socket API is written using netconn API, so it requires more memory to work. Moreover it does not support zero-copy functionality. The main advantage of using Socket API is a portability of software with other posix operating systems stacks [38].

3.5.1.4. Selection of LwIP API for ZigBee-to-Ethernet bridge.

The netconn API has been selected to be used in the Bridge software, because it is well suited for embedded systems, requires less flash and RAM memory then socket API, can operate in multi-thread environment and due to its simplicity.

3.5.2. Texas Instruments ZigBee stack – Z-stack.

Z-Stack™ is TI's ZigBee protocol stack for their ZigBee compliant transceivers. It supports both ZigBee and ZigBee PRO feature sets, and allows to implement Smart Energy, Home Automation and Commissioning Cluster Application profiles. Z-Stack™ gives the opportunity to update node software by Over the Air Download (OAD). Z-Stack is free, but the license allows it to be used only with TI's IEEE 802.15.4 transceivers.

3.5.2.1. Texas Instruments solutions for ZigBee networking.

Z-Stack can be used on three different types of hardware/software platforms:

- CC2530 SoC, which integrates radio transceiver and a microcontroller running protocol stack and application layer, making the solution cheap, highly integrated with small PCB footprint. It is well suited for nodes that demand little computing power, and have to be small, cost effective and low-power.
- CC2530-ZNP, which is the same hardware as CC2530 SoC, but provides an interface to run the application layer on a main application processor, connected through the SPI, USART or USB interface. This solution facilitates the creation of reliable ZigBee

device which demands much computing power and is handling signals from many sources. Moreover time to market is reduced, because CC2530-ZNP is a ZigBee Compliant Platform (ZCP) [39].

- CC2520, which is only an IEEE 802.15.4 compliant radio transceiver and the higher layers: ZigBee protocol stack, and application are implemented in the host processor.

The ZigBee-to-Ethernet bridge uses the CC2530-ZNP and the Texas Instruments ZigBee Application Processor (ZAP) framework running on the application microcontroller (STM32F4). A ZAP framework is described in the following section.

3.5.2.2. Description of a ZAP framework.

The architecture of a ZAP framework is illustrated in the Figure 3.11. The ZAP framework uses TI's OSAL non-preemptive operating system, and provides the same API as TI's ZigBee stack. There are four layers, and each layer is implemented as a separate task within OSAL:

- HAL – hardware abstraction layer used to make upper-layers of software independent from the underlying hardware. To run the HAL-based code on a particular hardware a so-called port has to be written. The port is the implementation of low-level functions such as: communicating via SPI interface, running timers, reading and writing to GPIOs etc., specific for particular microcontroller.
- ZAP – is a translation layer between Z-Stack and ZNP APIs, thus application written using Z-Stack API can be simply moved to CC2530-ZNP based system.
- ZCL – handles functions from ZigBee Cluster Library.
- Application – an operating system task where programmer defines the top-level functionality of the ZigBee device using ZAP framework and ZCL libraries.

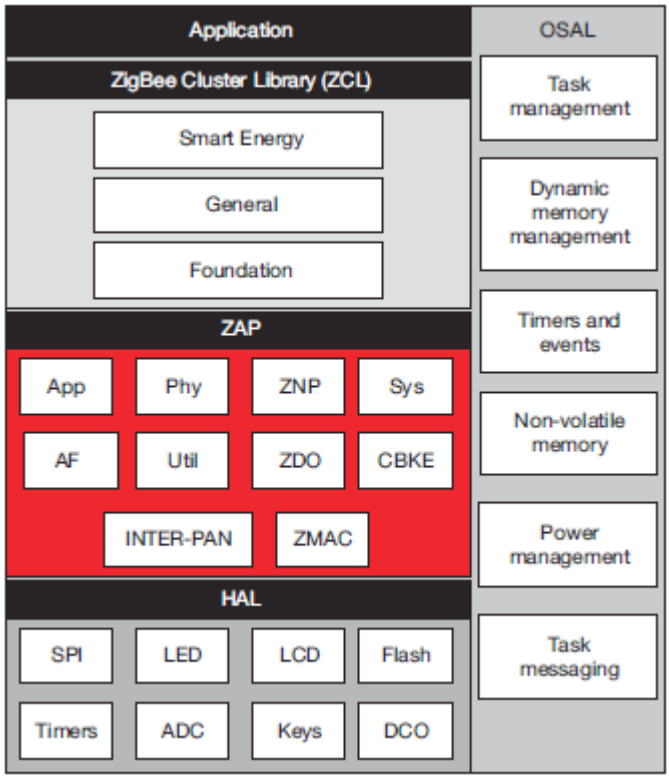


Figure 3.11. Layers of the ZAP framework used in the project [47].

Such layer based approach to writing software, and the well-defined framework allows programmer to focus mainly on the application layer and spend less time writing a program for the device. It also makes it easier to pass required tests in order to qualify for a ZigBee Certified Product logo.

3.5.3. FreeRTOS overview.

In order to improve the software architecture, and make it easily extendable, a real time operating system (RTOS) is used. FreeRTOS has been selected. It is free, even for commercial products and supports 31 hardware architectures. The FreeRTOS kernel can be configured to be preemptive or cooperative [40]. It is designed to have small footprint and requires less then 9 kilobytes of flash memory. It is easy to use, and to port to another hardware platforms.

3.5.4. Application-level software description.

The application software of a ZigBee-to-Ethernet bridge runs a multi-task environment and consists of three tasks:

- task_httpserver – is responsible for running HTTP server.
- task_tcp_db – which sends data from PAN network sensors to a remote server through a dedicated TCP port. The server runs a Java web application, that is used to control and acquire measurements from devices within PAN networks controlled by ZigBee-

to-Ethernet bridges. Measurement data is stored in an external MySQL database.

- task_znp – is responsible for ZigBee connectivity, by running OSAL operating system (described before) inside a single FreeRTOS task with appropriate OSAL-tasks defined.

3.5.4.1. Description of the RTOS task responsible for ZigBee communication.

The "task_znp" contains implementation of an application layer of ZAP framework described in the section 3.5.2.2, and runs TI's OSAL operating system.

A communication with another tasks is done by RTOS queues. These queues are used to send configuration commands for a ZigBee networks, as well as to provide measurement values – from ZigBee nodes within a ZigBee-to-Ethernet bridge PAN network – in a Ethernet-based TCP/IP network, by "task_httpserver" and "task_tcp_db".

ZAP layer of the ZAP framework (Figure 3.11) uses HAL functions defining methods to access a specific hardware platform. Texas Instruments provides only two kinds of HAL port for their microcontrollers, so it was required to write HAL port to access STM32F4 microcontroller peripherals. An interface between CC2530-ZNP and STM32F4 (application processor) is realized using SPI peripheral, and this interface is described in the following subsection.

3.5.4.1.1. SPI transmission between an application processor and CC2530-ZNP overview.

Figure 3.12 shows the general frame format used for SPI transmission between a CC2530-ZNP and an application processor. SPI has been selected because of higher bit-rate than in case of USART based transmission. CC2530-ZNP works as SPI slave, STM32F4 is SPI master and the bus clock is up to 4 MHz.

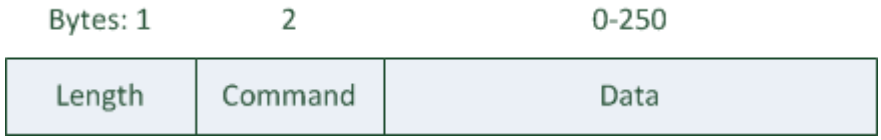


Figure 3.12. General ZNP frame format used for SPI transmission.

SPI mode of transmission in CC2530-ZNP uses three types of transactions:

- AREQ – Asynchronous Request - single direction transaction in which an application processor sends data and requires no acknowledgment from CC2530-ZNP. After sending this message application processor processes another instructions in the "task_znp" task without waiting for a response.
- SREQ – Synchronous Request – application processor in a single transaction not only

sends data, but also receives response from CC2530ZNP. SRSP – Synchronous Response – is the type of command send from ZigBee module.

- POLL – is the type of transaction in which CC2530-ZNP sends its queued data. When the POLL frame is ready, the CC2530-ZNP indicates this by pulling the SRDY (slave ready) line low, as long there is no pending transmission indicated by the MRDY line (master ready).

Figure 3.13 provides an illustration of how communication with SREQ command looks like. The rest of the commands are similar.

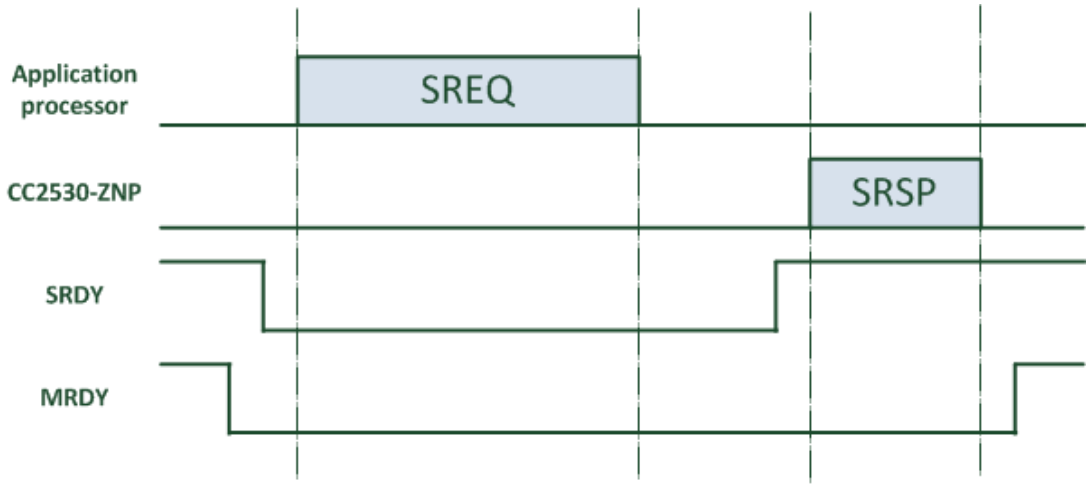


Figure 3.13. Illustration of SREQ – SRSP transaction, carried out between application processor and CC2530-ZNP.

When application processor has SREQ command to send, it holds MRDY pin low, and waits for low level on the SRDY line. This indicates that CC2530-ZNP is ready to receive command. Then SREQ is being transmitted, and after that application processor waits for SRDY line to go high. At this moment SRSP command from CC2530-ZNP to application processor is being transmitted. Because communication on the SPI bus is between master and slave, master has to generate the clock for the whole transmission. If all bytes specified by the length field in SRSP frame are received, the application processor sets its MRDY line high, ending the transaction process.

3.5.4.1.2. Description of DMA-based algorithm to handle SPI communication.

The formula used to calculate the time needed to send "x" bits, with f_{SPI} clock frequency is as follows:

$$T(x)=\frac{x}{f_{SPI}} \quad .$$

The longest possible frame, having 253 bytes therefore requires:

$$T(253\cdot 8)=\frac{253\cdot 8}{4\cdot 10^{6}}[s]=0.506[ms] \quad .$$

HAL functions for handling SPI transmission offer only polling mode, where the application must constantly check if the transmission has finished. There are situations when large amount of transmissions are carried out one by one, and the CPU can become blocked due to polling mode of transactions, which may affect other tasks requiring computing power. To overcome this problem higher-layer functions responsible for SPI communication were changed to allow DMA transfer, which saves CPU computing power.

An exemplary sequence diagram for SREQ type of transaction is presented in Figure 3.14. Other types of transaction are implemented in a similar way.

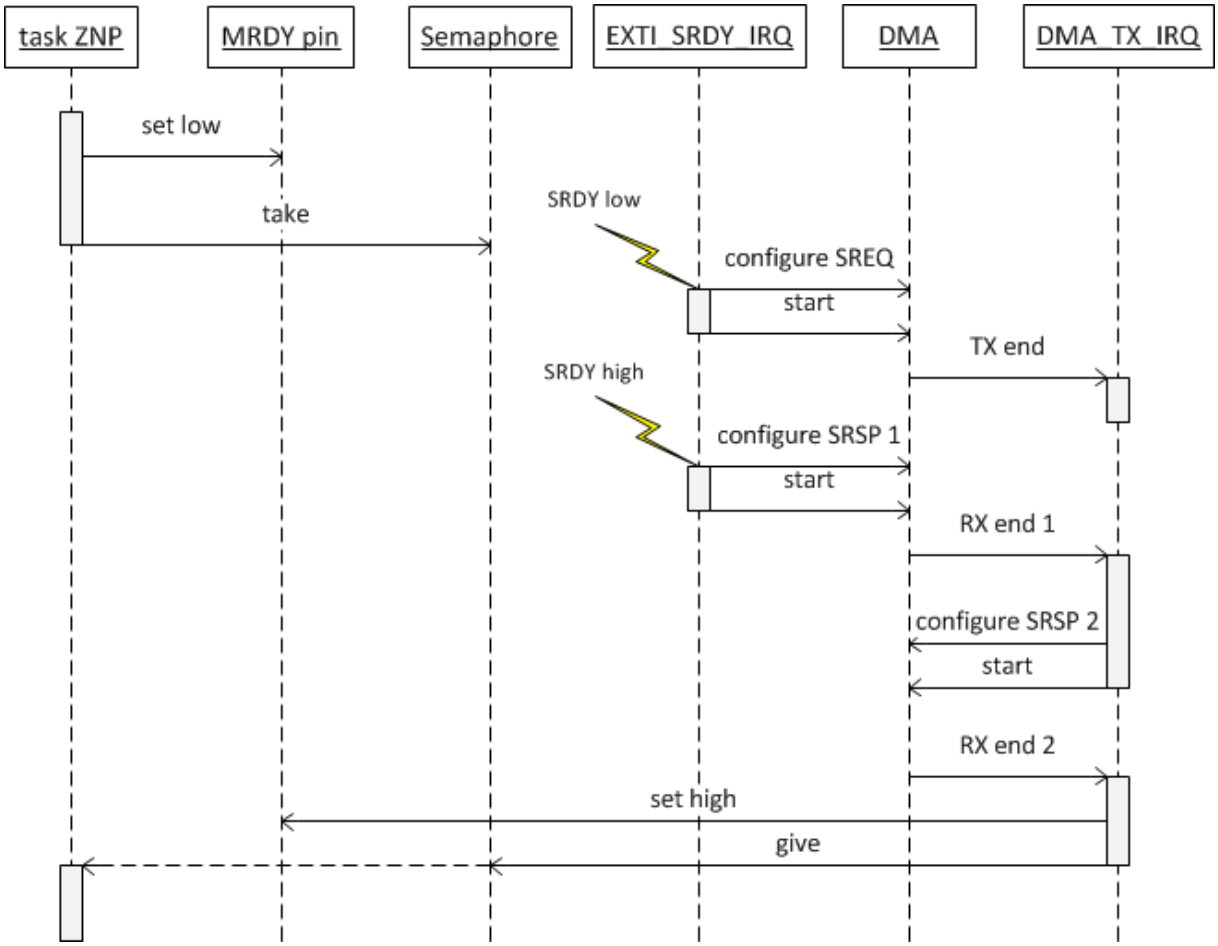


Figure 3.14. Sequence diagram of algorithm used to process SREQ command for the CC2530ZNP.

Sequence diagram shows SREQ command being executed within “task_znp”. The rectangle on the lifeline represents time in which CPU computing power is needed.

To indicate that an application processor has data for CC2530-ZNP, a MRDY pin is set high within the “task_znp”. Then the “task_znp” tries to take RTOS semaphore with timeout

parameter. Now the “task_znp” is blocked as long as semaphore is not taken, or timeout occurs.

When CC2530-ZNP is ready to receive SREQ frame, it indicates it by setting SRDY pin low, which is handled by external interrupt service routine (EXTI_SRDY_IRQ) in the application processor. In this routine DMA peripheral is configured to send SRDY frame as a whole, and is triggered to do that.

End of DMA transaction is indicated by DMA interrupt. CC2530-ZNP processes the received data, executes requested command, and when has SRSP frame ready sets SRDY line high, which triggers a next external interrupt in the application processor.

The application processor configures DMA to receive a first byte of a SRSP frame containing an information about a frame length. This length is known in the next DMA interrupt handler function, in which DMA peripheral is configured to receive the rest of the SRSP frame.

Reception of the frame is indicated by the application processor by setting MRDY line high, then RTOS semaphore is given. After that “task_znp” is in ready state.

Most of a transmission time the CPU is used to perform another RTOS tasks, which increases total performance of the microcontroller. To prevent conflicts, only one task has an access to SPI used for communication with CC2530-ZNP.

3.5.4.1.3. Software command interface for CC2530-ZNP.

The types of transactions mentioned in subsection 3.3.5.1.1 are used to process CC2530-ZNP software command interface, which consists of:

- SYS interface – used for low level functions for CC2530-ZNP hardware and software, such as non-volatile memory, GPIO pins or ADC.
- Configuration interface – “network specific” and “device specific” parameters can be set via this interface, and they are stored in a non-volatile memory. “Network specific” parameters should be the same within all devices in a network, for example: radio channel list, security key or PAN ID. “Device specific” parameters can vary depending on a ZigBee node: device type (coordinator, router, end-device), or user description.
- Simple API interface – used for very simple applications, with limited features of ZigBee stack.
- AF interface – used to interact with the Application Framework layer.
- ZDO interface – used to interact with the Zigbee Device Object layer.
- UTIL interface – additional functionality for ZigBee module, used for example to test connection between application processor and CC2530-ZNP.

3.5.4.1.4. Description of a ZAP application layer.

The application layer of the ZAP framework is based on an example provided by Texas Instruments.

Texas Instruments provides “ZigBee Sensor Monitor” MS Windows application, which visualizes ZigBee network. Each node is presented as a circle with network address, measured voltage and temperature. Nodes are connected by lines symbolizing radio links between them. Thanks to using sample ZAP application code supplied with “ZigBee Sensor Monitor” tool, the PAN network handled by the ZigBee-to-Ethernet bridge can be easily visualized. To do that the bridge must be connected to a computer by a serial port. It is however possible to redirect this communication to a TCP port, allowing to view the PAN network structure on-line.

In the network created for the demonstration of a bridge operation, there are ZigBee nodes working as routers and end-devices. In the constant time intervals they send their measurements to the central node (the bridge) which is connected to the TCP/IP network. Measurements, and network specific parameters are then send from “task_znp” to “task_http” and “task_tcp_db” by the FreeRTOS queues.

3.5.4.2. Description of the RTOS task responsible for handling HTTP requests.

The data sent by the “task_znp” is taken from the queue and stored on a local node list of the the HTTP task. The list implementation and data structure is available in appendix G. Among other parameters, a single message retrieved from the queue contains the MAC address of the sending node. When new message arrives, the corresponding entry in the node list is updated with measurement data or new node is registered on the list. The node list is visualized as a table within a web page served by the ZigBee-to-Ethernet bridge.

3.5.4.2.1. Handling HTTP requests using LwIP netconn API.

The LwIP stack does not support HTTP server functionality so it had to be implemented. Figure 3.15 shows transactions which occur between web browser and the task acting as HTTP server.

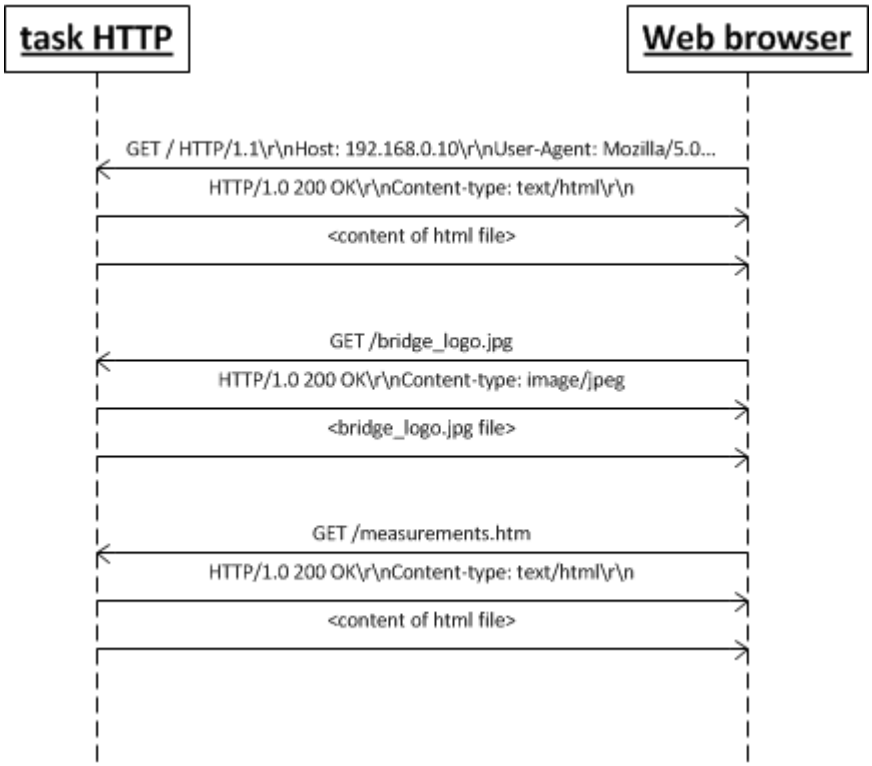


Figure 3.15. Transactions between web browser and "HTTP task".

When user wants to configure bridge and to see data from nodes within its PAN network, he opens a web browser (for example Mozilla Firefox) and enters IP address of the bridge for example 192.168.0.10. After pressing "enter" the connection with port 80 is established and "GET /HTTP/1.1\r\nHost: 192.168.0.10\r\nUser-Agent:Mozilla/5.0 … " command is sent from the web browser. The HTTP task reads data from the port using the following part of code:

```
...
inbuf = netconn_recv(conn);

if (inbuf != NULL) {
      if (netconn_err(conn) == ERR_OK) {
            netbuf_data(inbuf, (void**)&buf, &buflen);
            ...
      }
      ...
}
...
```

The netconn_recv() function blocks the task until data is in the network buffer, and then by netbuf_data() function, pointer to this data and its length is obtained. Now the "buf" is the pointer to "GET /HTTP/1.1\r\nHost: 192.168.0.10\r\nUser-Agent:Mozilla/5.0 … " c-string. The strcmp() function from standard stdio.h library is used to recognize a string and execute appropriate "if" statement.

When the request to load the main page is detected, then standard "GET" response is

sent: “HTTP/1.0 200 OK\r\nContent-type: text/html\r\n”. “Content-type” statement informs the web browser about the kind of data which will be sent. In this case the “Content-type” is HTML document. HTML document is dynamically generated, because it depends on the settings of the bridge and current state of a PAN network. Sample code illustrating the sending of data is given below:

```
strcpy(cDynamicPage,
      (const char*)
"<div id=\"menu\" style=\"background-color:#FFFFFF;float:left;\">"
);

strcat(cDynamicPage,(const char*)"<table border=\"1\">");

//row 1
strcat(cDynamicPage,(const char*)"<tr><td>ZB IEEE address</td>");
strcat(cDynamicPage,(const char*)"<td> 0x");
for (i = 0; i < 8; i++) {
      sprintf(temp_arr, "%x", zct.devIEEEaddr[i]);
      strcat(cDynamicPage," ");
      strcat(cDynamicPage,temp_arr);
}
strcat(cDynamicPage,(const char*)"</td></tr>");


//row 2
strcat(cDynamicPage,(const char*)"<tr><td>ZB short address</td>");
strcat(cDynamicPage,(const char*)"<td> 0x");

...

netconn_write(conn,cDynamicPage,u16_t)strlen(cDynamicPage),NETCONN_NOCOPY);
```

Because content of the web page is dynamically generated, the cDynamicPage[] table of type “char” is used to create c-string with HTML document. Example code presents creating the table, which is on the left, on the web page layout served by the bridge. First row of this table contains the IEEE address of the CC2530-ZNP ZigBee module integrated with a ZigBee-to-Ethernet bridge. When last line of HTML document is merged with c-string in cDynamicPage[] table, the netconn_write() function is executed to send finished HTML file to the web browser. Appendix B contains view of HTML file, that is being sent.

Browser receives HTML file content in which there are two statements informing that another data has to be uploaded:

```
…
<img src="bridge_logo.jpg" alt="bridge_logo" />
…
 <div id="measurements" style="background-color:#EEEEEE;float:left;">
  <iframe src="measurements.htm" width="800" height="600"></iframe>
 </div>
…
```

The first one is the “bridge_logo.jpg” image on top of the page. Web browser sends GET

request, then receives acknowledgment and the file, which it wanted. The same process occurs in the case of “measurements.htm” file, which is situated in the middle “iframe” .

3.5.4.2.2. Description of the layout of the web page served by the bridge.

The view of the web page served by the bridge is presented in Figure 3.16. The layout of the page is divided into four sections:

- the banner on the top,
- table with bridge settings on the left,
- data from sensors within the PAN on the right,
- footer at the bottom.

The table consists of parameters which user can change or parameters which only serve information. The first four fields of the table are read only:

- ZigBee IEEE address,
- ZigBee short, network address,
- current radio channel in which the bridge operates,
- 64-bit Extended PAN ID, which is automatically set by coordinator during network formation.

The next three fields can be changed – they are used to change network settings of the bridge, and are as follows:

- IP address,
- subnet mask,
- default gateway.

The following fields also can be changed. They are stored in the non-volatile memory, and are used for ZigBee networking:

- user description – short text, which can be seen by another nodes in the network. This also can be stored in the MySQL database in the table describing all managed Bridges.
- PAN ID - the address of the network within a specific radio channel with range of values between 0x0000 and 0x3fff,
- list of channels in which bridge can start network.

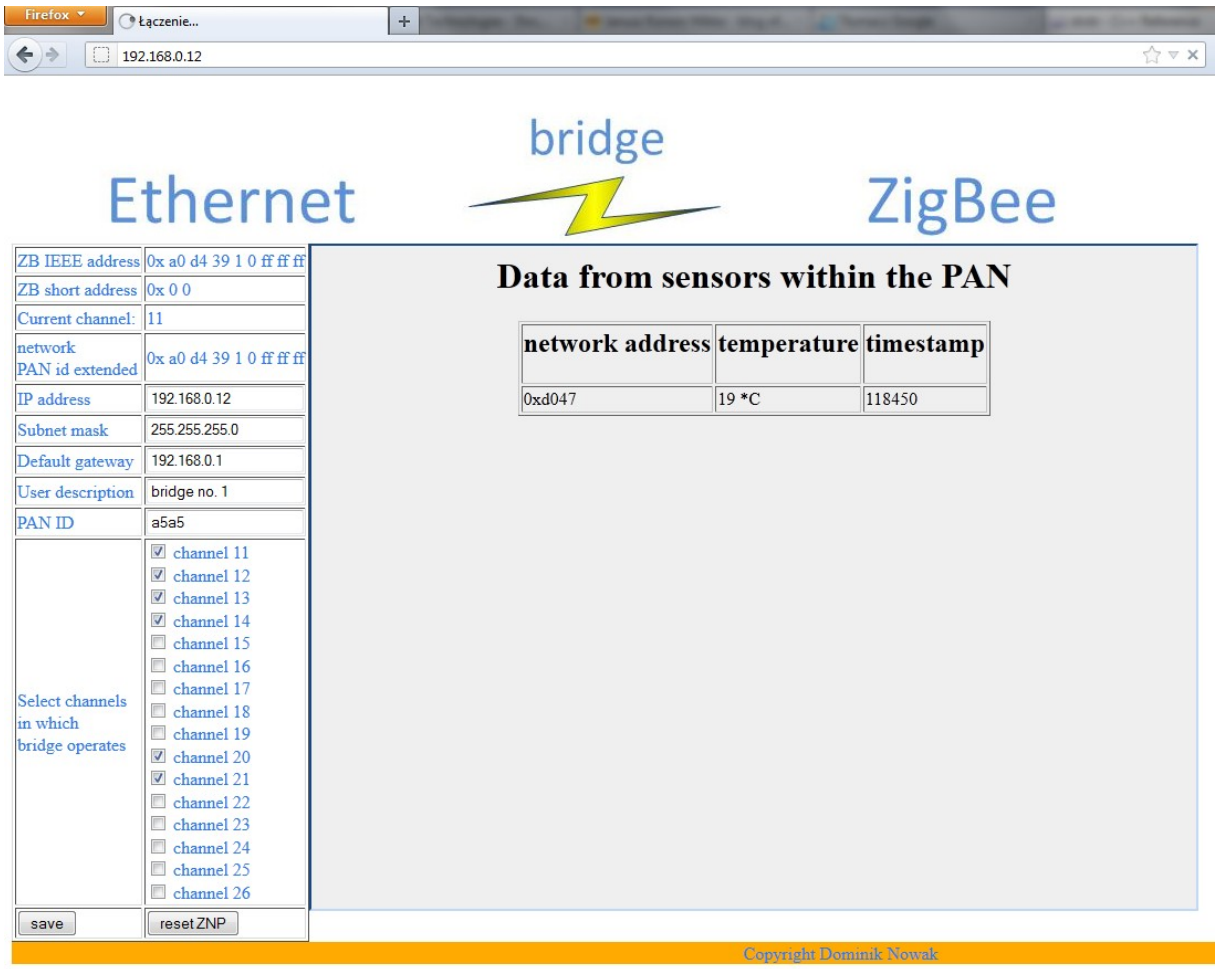


Figure 3.16. View of the web page served by the Bridge.

3.5.4.3. Description of the Java web application to manage ZigBee-to-Ethernet bridges.

The "WSN Manager v1" java web application runs on the server and allows to view data collected from sensors in the long term.

3.5.4.3.1. Use case UML diagram and main window of the "WSN Manager v1" web application.

Use case UML diagram of the "WSN Manager v1" web application is illustrated in Figure 3.17.

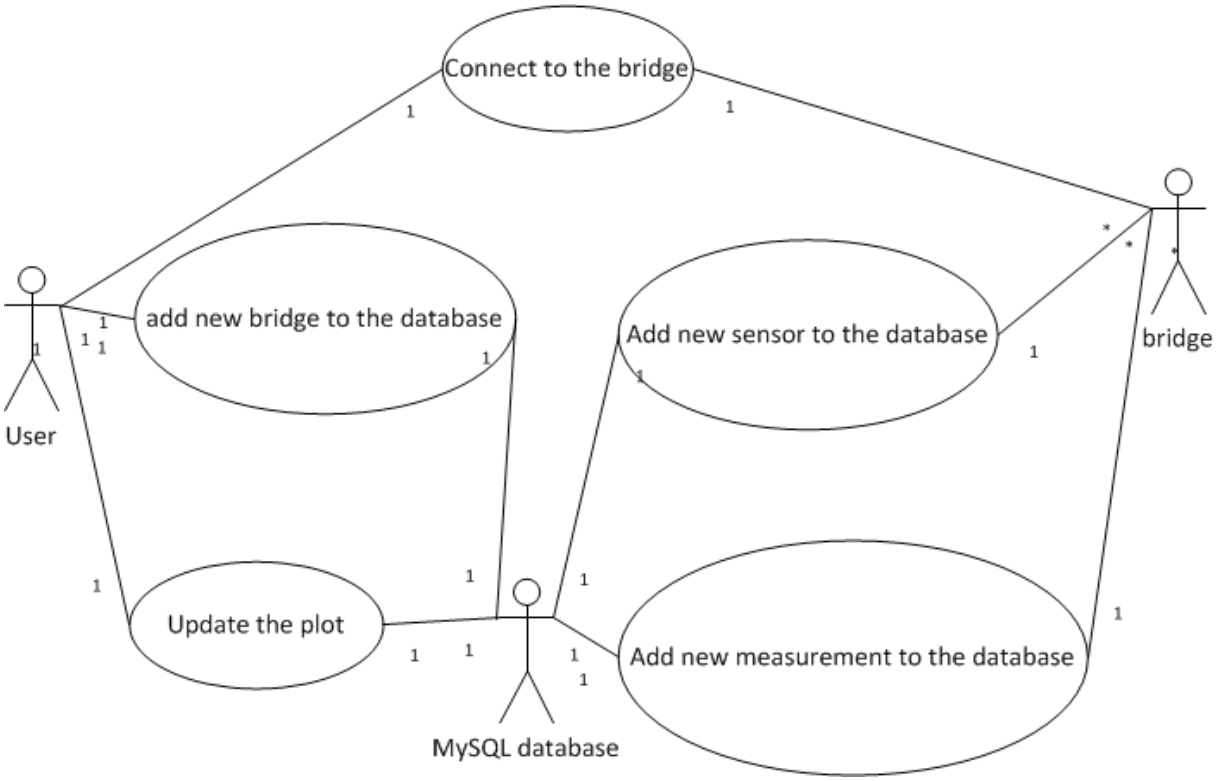


Figure 3.17. Use-case UML diagram of the "WSN Manager v1" web application.

In the first version of created web application only a few functions are possible. User can add new bridge to the database. He inputs unique bridge ID, IEEE address, and description, then this values are saved in a new record in the sens_location table on the SQL server. If the bridge is saved in the database, user can connect to it by typing the IP address, and the TCP port. After that, if new measurement is sent by any ZigBee node, its parameters are forwarded into web application and stored into database. When a new node is connected to the PAN network, it is automatically added to the corresponding table. In any moment user can click the "update plot" button, and the plot with data from all sensors is generated.

Figure 3.18 presents the main screen of the web application. It has been written in Java in the NetBeans Integrated Development Environment (IDE).

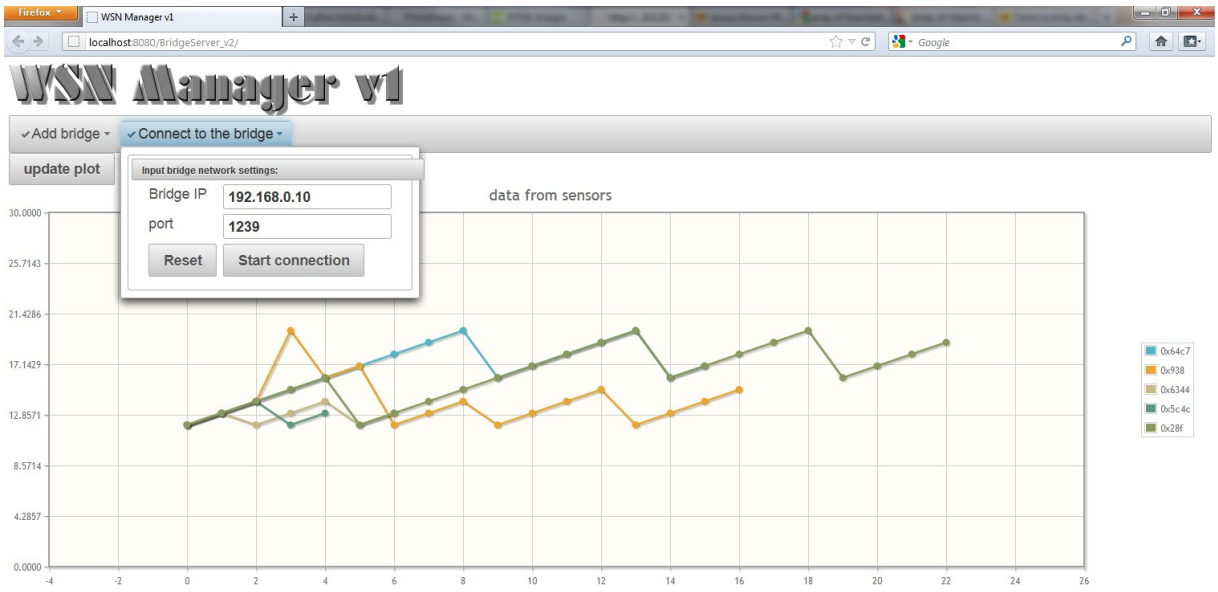


Figure 3.18. View of the "WSN Manager" web application.

3.5.4.3.2. Description of the development environment and software frameworks used in "WSN Manager v1" web application.

NetBeans IDE is well integrated with Java and services, which are necessary for writing web applications, such as: servers, databases and web services. Moreover popular frameworks are supported. Two of them have been used in the project: Java Server Faces (JSF) and Hibernate 3.2.5:

Java Server Faces technology is used to build user interface on the server side. It allows separation between the operation of the program and the presentation [41]. The Java code can be written, then tested, and when is done, its methods can be simply merged directly with XHTML document, that can be prepared by another developer. Without using any scripts, java methods are integrated with XHTML tags. Example of this is shown below:

```
<p:commandButton id="cmd1" value="Save bridge"
      action="#{bridgeZbEth.saveBridgeInDB()}" />
```

This tag creates command button, which can be identified by another elements of the web page by its "id". Value displayed in the button in the user interface is "Save bridge". When the user clicks it, the method "saveBridgeInDB()", which is the part of the "bridgeZbEth" object is executed.

There are JSF component libraries available, which can be used to provide a nice, modern look of the page. Some component libraries do not work well together, so it is recommended to select one of them. In the project, the PrimeFaces is used. The main web page XHTML code is situated in Appendix C.

Hibernate is an open source framework, which implements so called persistence layer – the layer, which translates object-oriented domain model into relational database [42]. Database tables are mapped into Java classes, hiding from the programmer point of view the methods and algorithms used to low-level database access. SQL calls are generated by the Hibernate automatically, and user do not need to manually convert received data into object domain [43]. Moreover it supports replacing the SQL database by another – only the little changes in mapping files are needed.

3.5.4.3.3. Connecting to a MySQL database by using hibernate framework.

To connect database tables with Java objects, the below listed files are needed:

- Hibernate Configuration File (hibernate.cfg.xml),
- Hibernate Helper File (HibernateUtil.java),
- Hibernate Mapping Files and Java classes.

Hibernate Configuration File contains information about the database connection, resource mappings and used SQL queries [44]. In this file SQL driver, URL to the database, user name and password are described. It also contains paths to the mapping files for each table in the specified database.

Helper File is used to access Hibernate's SessionFactory to obtain a Session object, according to the settings in the Configuration File.

To create mapping files and Java classes NetBeans IDE provides the so called: "Hibernate Reverse Engineering Wizard". IDE connects to the database specified in the Configuration File and allows user to select tables, which are to be mapped to the Java classes, then reverse engineering file is created (hibernate.reveng.xml).

Finally using hibernate.reveng.xml and hibernate.cfg.xml files, the mapping files and Java classes are created.

MySQL database is used in the project, and it consists of the following tables:

- sens_data – measurements from all sensors,
- sens_sensors – list of all sensors from all nodes,
- sens_nodes – list of all ZigBee nodes within bridges PAN networks,
- sens_location – list of location of each PAN, which is synonymous with the location of each Bridge,
- sens_types – types of measured values.

Corresponding to these tables files created by Hibernate wizards are:

- SensData.hbm.xml, SensData.java,
- SensSensors.hbm.xml, SensSensors.java,
- SensNodes.hbm.xml, SensNodes.java,
- SensLocation.hbm.xml, SensLocation.java,
- SensTypes.hbm.xml, SensTypes.java.

This example shows how function to insert new record into the table with data from the sensors looks like:

```
private void saveNewMeasurement(Double value, Date date, int sensorId)
{
   Session session = HibernateUtil.getSessionFactory().getCurrentSession();
   session.beginTransaction();
   SensData measurement = new SensData(date, sensorId, value);
   session.save(measurement);
   session.getTransaction().commit();
   HibernateUtil.getSessionFactory().close();
}
```

At first connection session is created, the new measurement is saved into it, then it is committed to the database and finally the session is closed.

3.5.4.4. Description of RTOS task responsible for communication with "WSN Manager v1" Java web application.

The task responsible for communication with the web application checks if the new message is available in the FreeRTOS queue. If so, the data is prescribed into the data structure which is then sent via TCP port 1239. The main loop of the task is as follows:

```
while(1) {
  if(connection_is_ok(newconn1)==0) {
    netconn_close(newconn1);
    netconn_delete(newconn1);
    newconn1 = netconn_accept(conn1);
  }
  if((newconn1 != NULL)) {
    if(connection_is_ok(newconn1)==1) {
      while((tmp2 = uxQueueMessagesWaiting(sensorData_queue_db)) > 0) {
        if(pdTRUE == xQueueReceive(sensorData_queue_db,&msg_static,100)) {
          sendSensorData(&msg_static,newconn1);
        }
      }
    }
    vTaskDelay(DLY10ms);
  }
}
```

In the loop the connection is checked. If it is closed, the current connection handle is deleted,

and the task is blocked, by the netconn_accept() function, until a new connection is requested.

3.6. Test of the system

View of the assembled prototype of the ZigBee-to-Ethernet bridge is given in Figure 3.19. Test network used to verify operation of the device is presented in the window of ZigBee Sensor Monitor Application provided by Texas Instruments, which is illustrated in Figure 3.20.

Figure 3.19. View of the assembled ZigBee-to-Ethernet bridge.

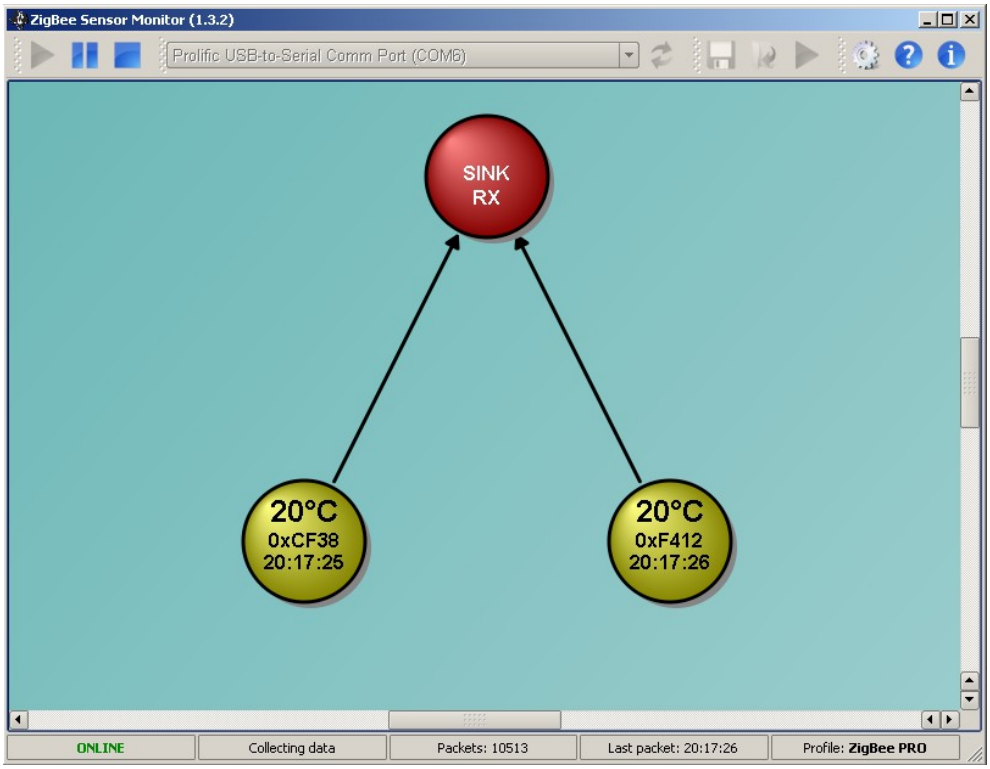


Figure 3.20. View of the tested network containing the Bridge (Red circle) and two nodes sending periodically generated sample values.

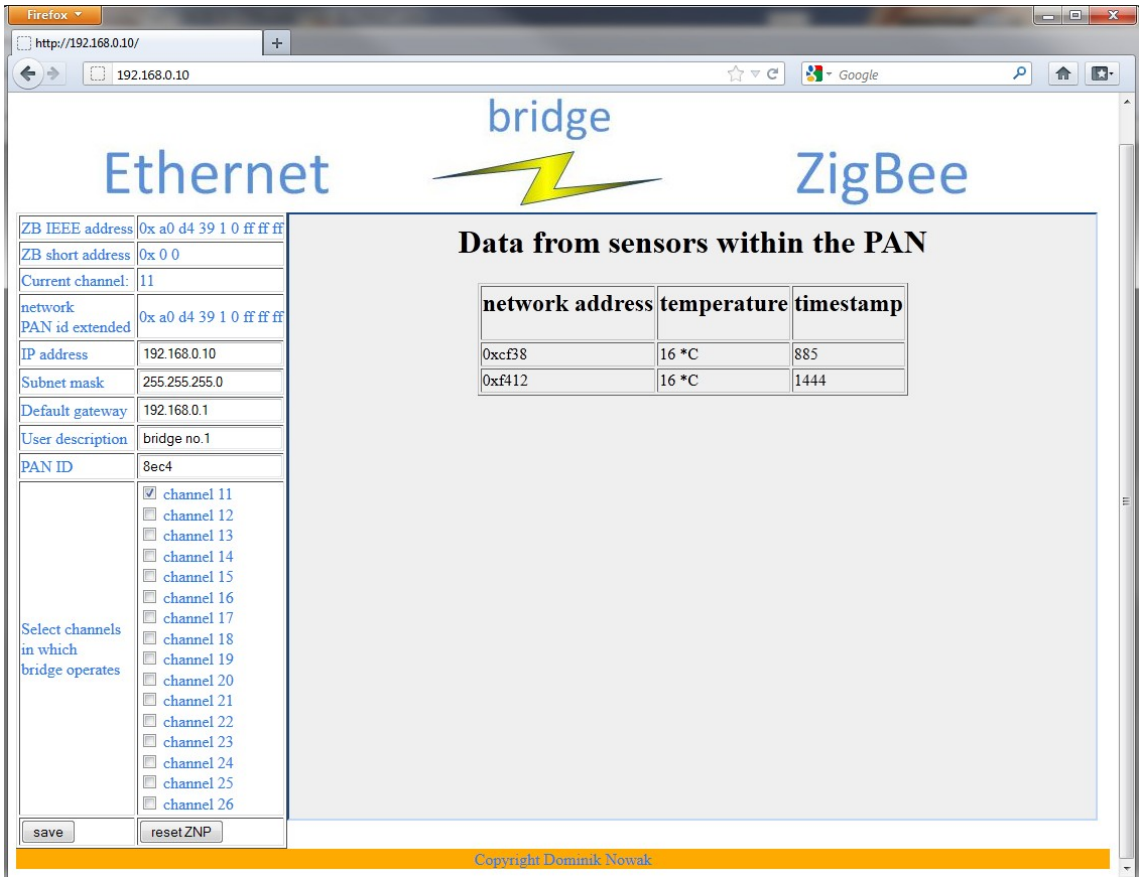


Figure 3.21. View of the web page served by the Bridge. PAN network contains two nodes.

This is a star network topology in which two nodes are directly connected to the bridge, but also another topology was tested, in which one of the nodes is intermediate node, that transfers data from the second one. Two ZigBee nodes periodically send sample measurements to the bridge, and the bridge sends reports via serial port to the computer, running Sensor Monitor Application. Yellow circles contain network addresses, measured values and time of last update.

Simultaneously the Bridges PAN network is monitored by the web page, which can be seen in Figure 3.21. Moreover the “WSN manager” Java Web Application runs and constantly stores measured values which can be displayed as is shown in Figure 3.18. During the test there were no errors, and the system worked reliably.

4. DESIGN OF A ZIGBEE-TO-IEC61850 BRIDGE.

4.1. System architecture.

4.1.1. A ZigBee-to-IEC61850 bridge from the IEC 61850 network point of view.

A ZigBee-to-IEC61850 bridge is seen from the IEC 61850 network point of view as an Intelligent Electronic Device (IED). Figure 4.1 shows an IEC 61850 network. Ethernet bus is the physical medium to which multiple IEDs are connected. One of the IEDs is named as "WSN network". This IED is a collection of Logical Devices (LDs) – in this case ZigBee nodes. Each LD corresponds to a physical ZigBee node in the PAN network. Depending on the function performed by the ZigBee nodes in the substation, appropriate logical nodes have to be selected to describe IEDs functions in the IEC 61850 semantics.

In the presented exemplary solution, the WSN network consists of a ZigBee coordinator (ZigBee-to-IEC61850 bridge) and a ZigBee router with a temperature sensor. Logical nodes are grouped and the first letter of their name determines to which group they belong. In IEC 61850-7-4 document [23] logical node classes are described. Temperature measurement can be described using "LN: Temperature supervision" node class – STMP, which belongs to the group "Supervision and monitoring", and includes a Tmp Data Object, that stores the temperature value. The "WSN network" IED acts as a server and contains a local database that stores measurements acquired from the ZigBee nodes within the PAN network. Then these measurements can be acquired by any other IEDs.

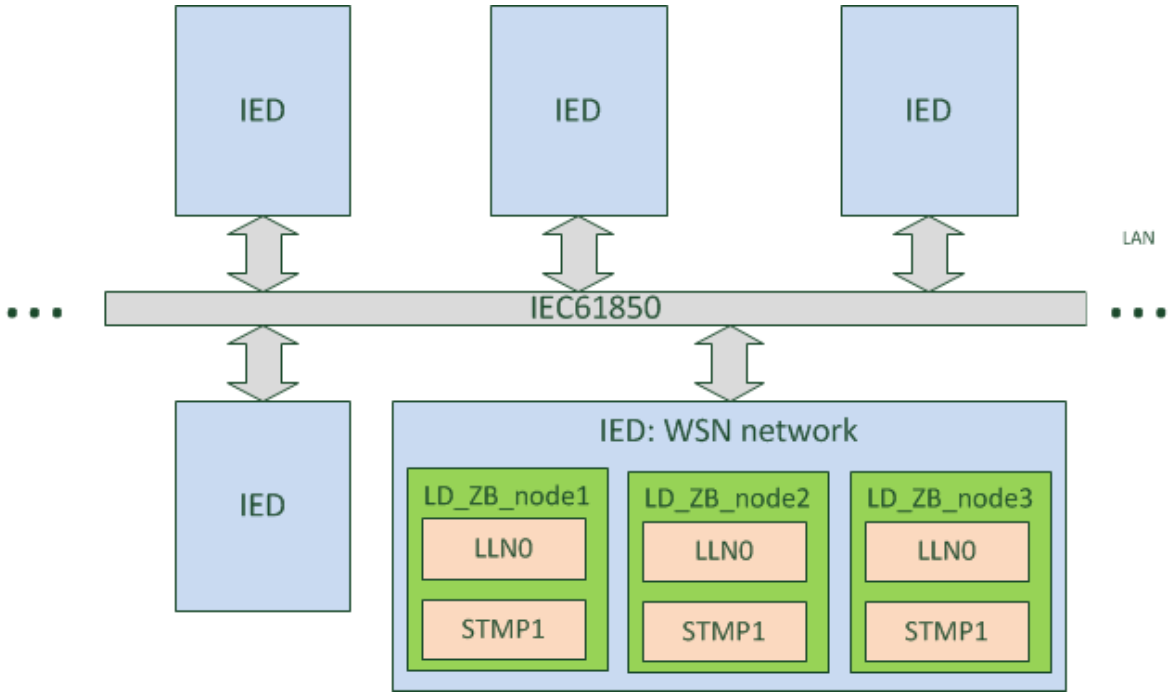


Figure 4.1. ZigBee-to-IEC61850 bridge acting as an Intelligent Electronic Device (IED).

ZigBee network does not give guaranties on delivery time of data packets. These times may vary with network size and topology, current network load etc. For this reason the data available in logical nodes is considered non-real-time. Thus reading this data does not require IEC 61850 real-time protocols such as GOOSE or SV, although it's possible to use them. However, the most adequate method of data exchange for ZigBee-toIEC61850 bridge appears to be MMS.

4.1.2. Hardware components of a ZigBee-to-IEC61850 bridge.

Figure 4.2 presents hardware components used in the "WSN network" IED. The most important part of the IED is a ZigBee-to-IEC61850 bridge which is in scope of this work. The prototype design is based on Development Kit DK61, provided by Beck IPC GmbH. It contains IPC@CHIP SC143 Embedded Web Controller.The kit is shipped with Paradigm C++ compiler and tools for developing applications. To make an IEC 61850 compliant device, it's best to use proven and reliable hardware and software components and tools. Beck IPC GmbH company offers reliable industrial control technology and communication products. They provide IEC61850 libraries suited to the SC143, which have been used in the project.

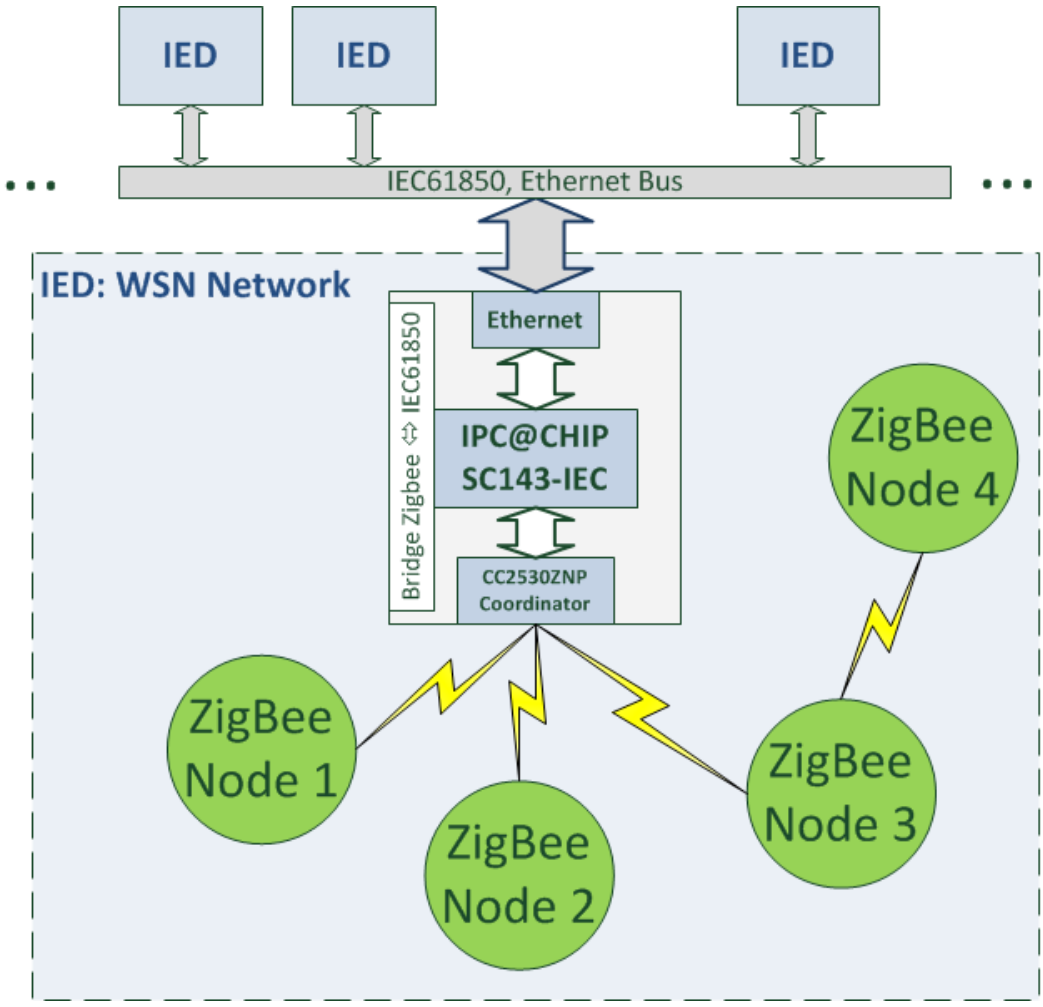


Figure 4.2. Components of the "WSN Network" IED.

To connect to the ZigBee network, the CC2530-ZNP module is used along with ZigBee software stack provided by Texas Instrument. The bridge acts as a ZigBee coordinator and a local database for other IEDs. The presented prototype uses remote temperature sensors only. Depending on the functions required by a substation, more sophisticated ZigBee nodes can be used.

The components of the bridge and the software implementation issues are presented in the next subsections.

4.2. BECK IPC@CHIP Development kit DK61.

The DK61 development kit contains the evaluation board with the SC143 Embedded Web Controller. It integrates 96 MHz SC186-EX, 16-bit 186 processor, 8 MB of RAM memory and 8 MB of flash memory. The following peripherals are provided by this controller:

- 2 x Ethernet,
- CAN 2.0,
- USB host and device,
- 34 GPIOs
- 16-bit address/data bus,
- SPI,
- I2C,
- RS232/422/485,
- DMA controller.

All of them are available on DK61 board connectors, so it is powerful platform, which can be used to integrate with many industrial communications standards. The SC143 controller is supplied in a BGA177 package. It uses preinstalled @CHIP-RTOS – real time operating system with full TCP/IP stack, Web server, FTP server, and Telnet server. Up to 12 DOS programs can run in that operating system simultaneously. They are loaded by the FTP connection.

4.3. BECK software tools used in the design.

The following software tools provided by Beck have been used in the project:

- Paradigm C++ Beck IPC Edition,
- IPC@CHIPTOOL,
- Postmake 2,

- ICD Designer,
- IPC@CHIP RTOS,
- IEC 61850 library.

4.3.1. Paradigm C++ development environment.

Paradigm C++ Beck IPC Edition is the development environment which provides tools to write, build and debug software for Beck's IPC@CHIP. To run it, an USB hardware license key is needed.

4.3.2. IPC@CHIPTOOL.

IPC@CHIPTOOL is a PC application that acts as a communication center for all IPC@CHIP – based products [45]. It is used to find IPC@CHIPs in the network, configure their serial numbers, IP addresses, network masks, gateway addresses and other network specific parameters. This software provides RTOS update functionality and allows to insert DOS programs into flash memory through a FTP connection. Typical IEC 61850 based DOS program requires additional files, which also can be loaded by the FTP protocol:

- AUTOEXEC.BAT – contains a script which is executed on the system start (after reset or power-up) with a list of DOS programs and their input parameters.
- CHIP.INI – contains IPC@CHIP system configuration settings, which are loaded at system startup. They relate to standard input devices, timers, network protocols, serial ports settings (for example to use DMA or not), power save mode, name of the device and many other.
- .ICD file – the defined by the IEC 61850 configuration file written in the SCL language.
- PIS10.key – the security key matching to the DK61 serial number. It is supplied with the IEC 61850 library.

IPC@CHIPTOOL integrates also serial and telnet clients, with are convenient during developing and testing applications for the IPC@CHIP.

4.3.3. Postmake 2.

Postmake 2 reduces the time needed to update software in the IPC@CHIP. It runs automatically after build process and sets a FTP connection to replace old version of DOS program. There is no need to manually set such connection using IPC@CHIPTOOL.

4.3.4. ICD Designer.

ICD Designer facilitates the process of building IEC 61850 SCL configuration files.

Instead of the XML based semantics, a hierarchical tree view is presented, as is shown in Figure 4.3.

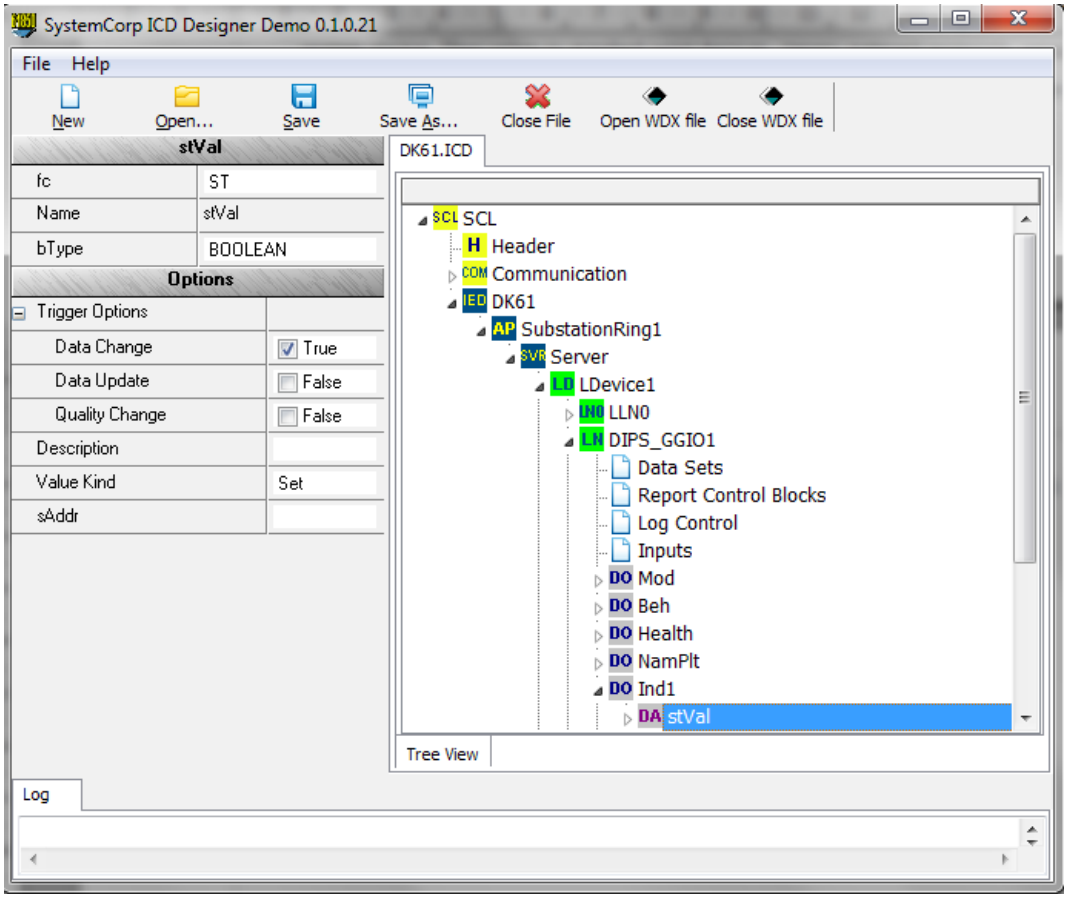


Figure 4.3. The view of ICD Designer main window.

All elements of the IED description are available through a graphical user interface in a clear and transparent manner, which greatly reduces SCL file development time.

4.3.5. IPC@CHIP-RTOS.

IPC@CHIP RTOS allows to execute up to 12 DOS multi-task programs simultaneously. IPC@CHIP RTOS provides abstraction layer for all hardware peripherals, serial ports, network services and supports FAT file systems and disk drivers. The architecture of the IPC@CHIP RTOS is illustrated in Figure 4.4.

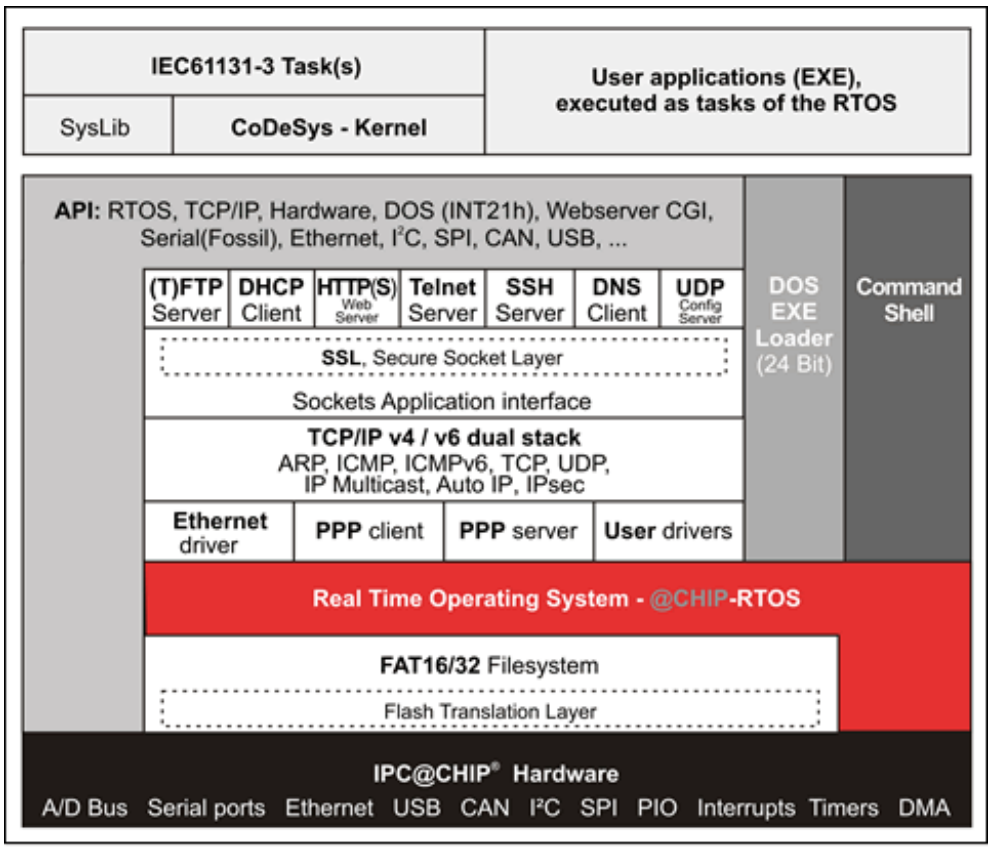


Figure 4.4. Architecture of the IPC@CHIP-RTOS used in the project [46].

The RTOS provides convenient, well documented API for task management, synchronization and communication. There can be 78 tasks running simultaneously, and the sum of available semaphores, timers, event groups is 128. To send data between tasks the mailbox mechanism can be used. IPC@CHIP-RTOS integrates also a command shell. The system configuration is stored in CHIP.INI file. On startup, the AUTOEXEC.BAT script is executed automatically to, for example, start default applications [46].

4.3.6. IEC 61850 Protocol Integration Stack.

IEC 61850 protocol library for IPC@CHIP is provided by the SystemCORP Pty Ltd. IEC 61850 Protocol Integration Stack (PIS) is used as shown in Figure 4.5.

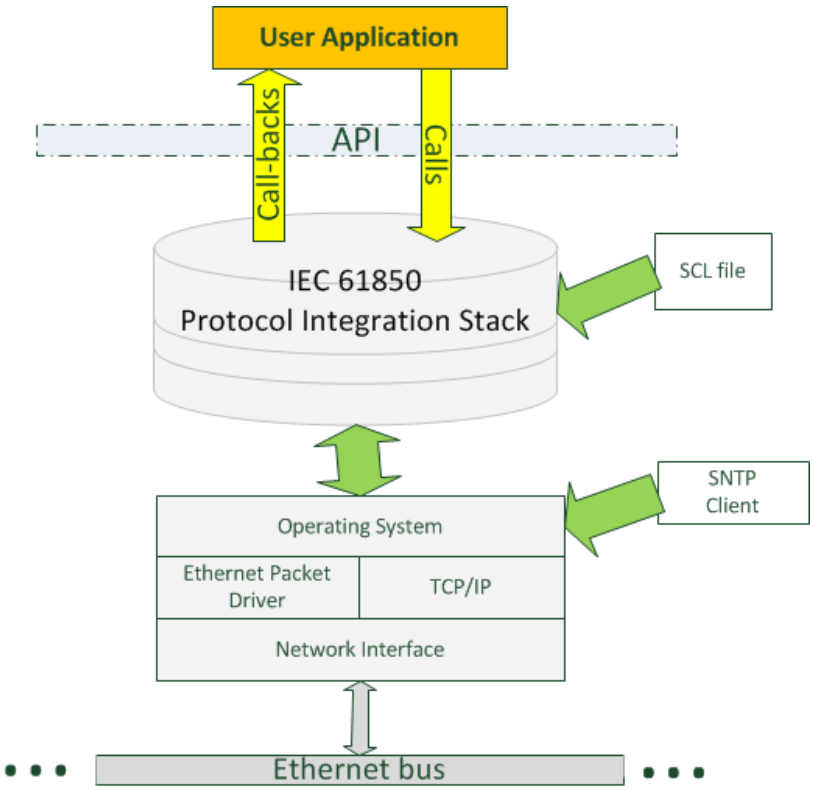


Figure 4.5. Context diagram of the system containing IEC 61850 stack provided by the SystemCORP Pty Ltd. [47].

It runs on the top of the operating system – in this case @CHIP-RTOS. Each data object provided by an IED is described using SCL file, which has to be loaded by the stack using integrated XML Parser. User application communicates with the IEC61850 PIS using "Calls" and "Call-backs" mechanisms provided by the API. SNTP protocol is used to provide time synchronization between IEDs and it runs independently as a operating system task. User application contains objects which (through call-back functions) are mapped into IEC 61850 Data Attributes (DA) described by the SCL file and provided by the IEC 61850 PIS. The API is divided into two categories:

- Client/Server Management,
- Data Attributes Access.

The following list presents the Client/Server Management functions:

- **IEC61850_Create()** - returns a client or server object. Input parameters are used to specify the object type, and contain pointers to the callback functions.
- **IEC61850_LoadSCLFile()** - reads SCL file and configures the client or server.
- **IEC61850_Start()** – starts the server or client.
- **IEC61850_Free()** – deletes a client or server object.

Whereas the Data Attributes can be accessed by these functions:

- **IEC61850_Read()** – reads the value of a data attribute.
- **IEC61850_Write()** – writes the value to a data attribute.
- **IEC61850_Update()** – updates the value of a specified data attribute.

The next subsections describe the process of the ZigBee-to-IEC61850 bridge design.

4.4. Hardware design.

The first step in building the ZigBee-to-IEC61850 bridge was to connect SC143 Embedded Web Controller with the CC2530-ZNP. Figure 4.6 shows which external ports of the SC143 have been used in the project. CC2530-ZNP is connected to the SC143 via SPI interface with additional signals described in the subsection 3.4.3.3. In addition to integration with the IEC61850 network, the ZigBee Sensor Monitor software provided by Texas Instruments is used to view nodes in the bridge's ZigBee PAN. ETH0 Ethernet port of the DK61 development board is used to connect the prototype of the bridge to the IEC 61850 network.

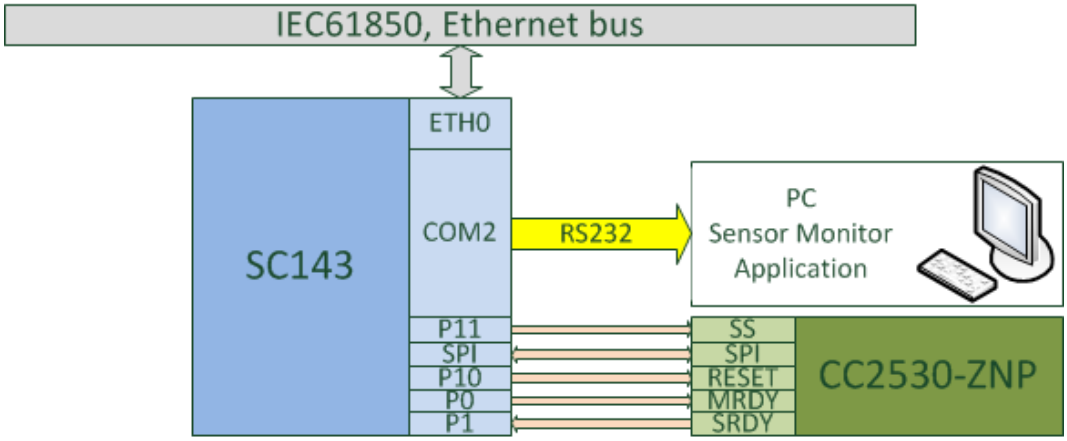


Figure 4.6. Hardware interfaces between SC143, CC2530-ZNP, Personal Computer and Ethernet bus.

4.5. Software architecture.

The ZigBee-to-IEC61850 bridge software uses Texas Instruments ZAP framework described in subsection 3.5.2.2. It had to integrated with the the SC143 software.

4.5.1. Z-Stack HAL port for SC143.

Thanks to a layer-based approach of the Z-Stack software and ZAP framework, to integrate the ZigBee module with the system, only the so called port has been written. The port contains definitions of the functions used to access the hardware CC2530-ZNP interface, and other peripherals used by TI's OSAL system. The port uses high-level software API for

the SC143 hardware, that is provided by IPC@CHIP-RTOS.

4.5.1.1. SPI interface for communication with the CC2530-ZNP.

To access SPI interface the following definitions are made:

```
static uint8 inByte;
#define HAL_SPI_SS_ON()             hal_write_pio(11,0);
#define HAL_SPI_SS_OFF()            hal_write_pio(11,1);
#define HAL_SPI_WRITE_BYTE(X)       spi_read_write_hw(
                                          (void far*) &(inByte),&(X), 1);
#define HAL_SPI_READ_BYTE()          inByte
#define HAL_SPI_WAIT_DONE()
#define HAL_SPI_INIT()              spi_init_hw(SPI_MODE0,50)
```

The hal_write_pio(11,0) function sets logical “0” on the pin P11. SPI API function spi_read_write_hw() blocks the calling task until specified number of bytes are sent via SPI. Because it is a full-duplex transmission, and after each transfer the output register of the SPI – master contains the byte received from the SPI – slave, this single function is used to read and write data. HAL definition used by the upper-layers of Z-Stack sends only one byte and after each operation the received value is saved the inByte variable. The frequency of the SPI master clock is specified by a CPU frequency divider passed as the second argument of the spi_init_hw() function. The followng formula is used to determine the clock speed:

$$SPI_{freq} = \frac{CPU_{freq}}{divider \cdot 2 + 2} \quad .$$

4.5.1.2. Setting serial port for communication with Sensor Monitor software.

To communicate with the PC with running Sensor Monitor software, the Fossil API has been used. It includes buffers for data pending transmission and for received bytes. Their size can be changed using CHIP.INI file. Internally Fossil performs serial port transmission using DMA peripheral. The API is very simple and the following functions are used in the project of the bridge:

- fossil_init() - initializes the Fossil driver,
- fossil_setbaud() - sets serial port parameters: baudrate, parity, word length and stop bits,
- fossil_getbyte() - reads the byte from the input buffer,
- fossil_status_request() - returns the status of the serial port.

4.5.1.3. Configuration of @CHIP-RTOS timers to run TI's OSAL.

To run TI's OSAL, the system timer had to be ported. System clock is initialized by the following procedure, which uses IPC@CHIP-RTOS Hardware API:

```
void InitClock(void)
{
   /* Configure TimerA as 1-KHz HAL Board timer to drive OSAL
      timers and  block waiting/sleeping. */

  hal_install_isr(8,1,tim0_isr_func);
  hal_init_timer(0,0x03,0x3000);
  hal_start_timer(0);   //start timer0
}
```

The hal_install_isr() is used to set external and internal interrupts. It contains three arguments:

- unsigned short irq – the number specifying which interrupt is selected,
- unsigned short count – the number of generated interrupts before executing the isr_handler,
- InterruptHandler isr_handler – the pointer to the function which is executed after interrupt.

The hal_init_timer() sets the timer, timer mode, and a clock divider. To start the Timer0, the hal_start_timer() function with argument "0" is executed.

4.5.2. Application description.

Figure 4.7 presents architecture of a ZigBee-to-IEC61850 bridge software. The following subsections contain description of each software component.

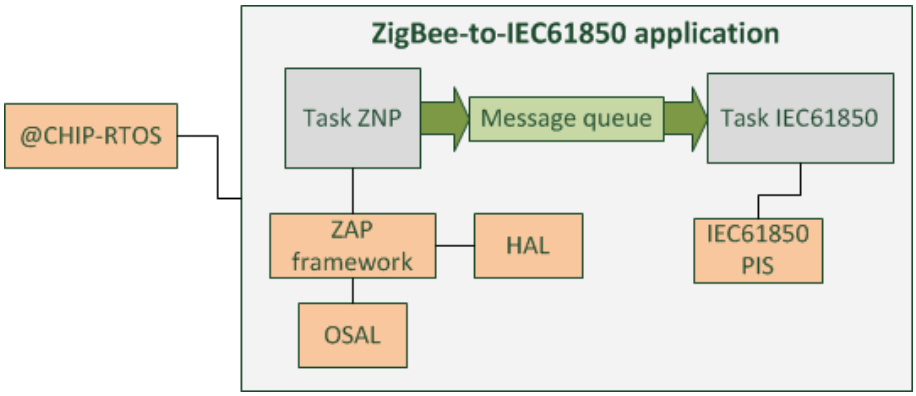


Figure 4.7. Software architecture of a ZigBee-to-IEC61850 bridge.

4.5.2.1. Interactions between RTOS tasks.

The implementation of main() function is as follows:

```
int main(int argc, char *argv[])
{
	RTX_Create_Msg(&temperature_Msg);

	RTX_Create_Task(&task_iec61850_Id, &task_iec61850_DefBlock);
	RTX_Create_Task(&task_znp_Id, &task_znp_DefBlock);
	while(1) {
		RTX_Sleep_Time(1000*60);
	}
	return 0;
}
```

The first step is to create a queue to communicate between the ZigBee and IEC61850 tasks, that are started by the next two RTX_Create_Task() functions. The rest of application is carried out by these tasks, and the main task enters an infinite loop.

Figure 4.8 shows how data from a ZigBee node is received and saved in a local database. ZigBee nodes periodically send temperature values, which are handled by the RTOS task responsible for ZigBee communication.

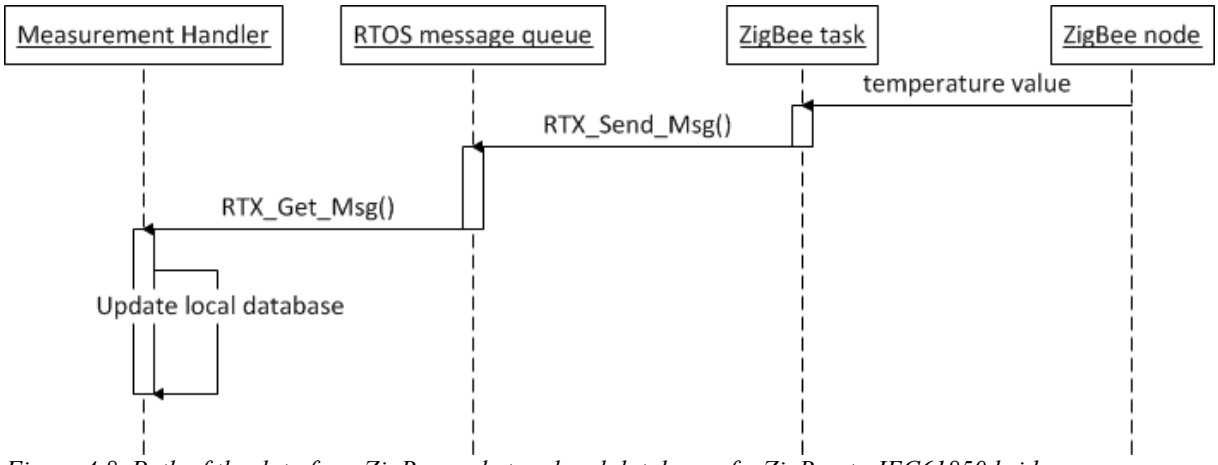


Figure 4.8. Path of the data from ZigBee node to a local database of a ZigBee-to-IEC61850 bridge.

Next, the received temperature value and identifier of the ZigBee node are sent to the RTOS message queue. Finally, running in an infinite loop Measurement Handler function in the IEC61850 task, checks whether there is a pending message, and receives it. Then the fields of the local database corresponding to the ZigBee node are updated.

4.5.2.2. Description of the IEC61850 RTOS task.

After calling IEC61850 task creation API, the functions illustrated in Figure 4.9 are executed.

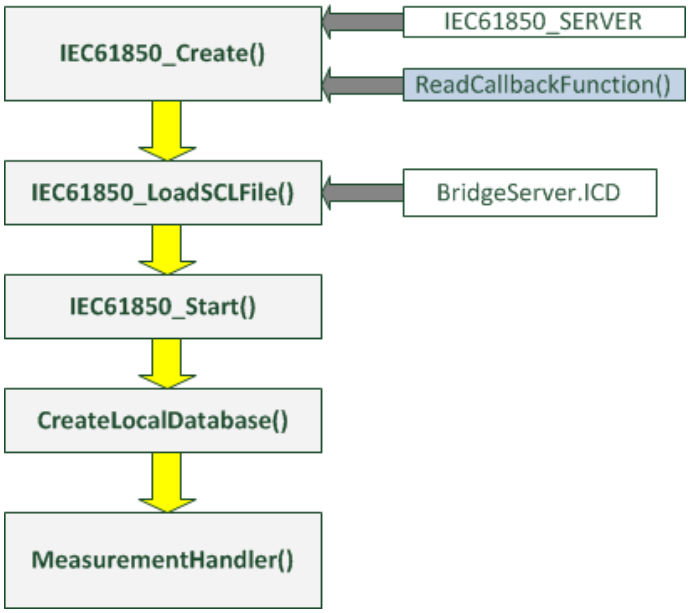


Figure 4.9. Function calls in the IEC61850 task.

4.5.2.2.1. IEC61850 object creation.

At first server object is created by IEC61850_Create() function. Other devices connected to a IEC61850 network can only read values that are made available by the prototype of the bridge. This is why only the ReadCallbackFunction() is defined. It describes how do handle client read calls. Measurements stored in the local database are returned as the IEC61850_ObjectData type object, and then are sent to the client by IEC 61850 protocol stack.

4.5.2.2.2. Loading SCL file.

When the server object is created, it is configured by the IEC61850_LoadSCLFile() function which uses BridgeServer.ICD file contained in the Appendix E. IEC 61850 PIS provided by SystemCORP has a built-in SCL parser, so plain text file can be directly used. The file has been created using the SystemCorp ICD designer software. It describes a logical device called LD_ZigBee_Node, which has got two logical nodes: LLN0 and LN_STMP1. As is written in the reference [23], the STMP Logical Node shall be used to represent various devices that supervise the temperatures of major plant objects. If more than one temperature sensor is connected to the Logical Device, each of them shall be represented by another LN STMP.

LN STMP appears to be the most suitable for the sample application, which stores temperature measurements from ZigBee nodes in the PAN network. LN STMP contains "Tmp" Data Object (DO) which belongs to the MV Common data class. Common Data Classes are described in the reference [22]. MV is the abbreviation for "Measured Value". The "mag" data attribute (DA) within the MV class stores analogue value, in this case temperature, whereas the "t" DA is the time stamp of the measurement. LLN0 – Logical Node zero is used to address common issues for LD such as operation time, local control behavior or reference to a higher level logical device [23]. In the described SCL file there is also a set of network-related addresses of a ZigBee-to-IEC61850 bridge: IP, subnet mask and default gateway.

4.5.2.2.3. Stating IEC61850 server.

If there are no errors during loading of the SCL file, the IEC61850_Start() function is executed, and then server starts operation. If there is a need to stop it, the API provides the IEC61850_Stop() function.

4.5.2.2.4. Local database creation.

Afterwards a local database is created. It is a two-dimensional array of the tBridgeObject type:

```
tBridgeObject atObj[OBJECT_TYPES][OBJECTS];
```

, where:

- OBJECT_TYPES - is the number of object types used in the application. Currently there is only one object type used: temperature sensor.
- OBJECTS - specifies the maximum number of ZigBee nodes which are in the PAN network.

The tBridgeObject structure is defined as follows:

```
typedef struct tag_BridgeObject
{
      unsigned char           ucObjectNo;       /* Object Number */
      unsigned char           ucObjectType;     /* Object Type */
      unsigned long int ucObjectValue;          /* Object Value */
      unsigned short int      usiObjectQuality; /* Object Quality */
      tNTPTimeStamp     tObjectTime;            /* Object Time */
}tBridgeObject;
```

These fields correspond to the data attributes specified in the BridgeServer.ICD file. In the

CreateLocalDatabase() function the values of the atObj[][] table are initialized. If the IED client wants to read data, the ReadCallbackFunction() is executed and by using the ID of the IEC61850 object, the requested values are returned from the local-data base and handled by the IEC 61850 stack.

4.5.2.2.5. Updating IEC61850 local database objects.

Appendix F presents the implementation of the function used to update the local database. It operates in an infinite loop. If the new message with temperature value is available, at first, the index of the ZigBee node represented internally as a IEC61850 logical device, is searched and then the corresponding object is updated by calling IEC61850_Update() function. The updated parameters are:

- temperature value,
- quality of the measurement,
- time stamp of the measurement.

The fields of each objects are described in the BridgeServer.ICD file.

4.6. Demonstration of the ZigBee-to-IEC61850 bridge.

Figure 4.10 presents a picture of a prototype ZigBee-to-IEC61850 bridge. The DK61 board is connected to a ZigBee module by 20 cm cables, but even with full SPI speed supported by a CC2530-ZNP (4 MHz) the transmission is reliable in a laboratory environment. In the same way the USART-to-USB converter used for TI's Sensor Monitor application is connected.

During IPC@CHIP-RTOS start-up the ZigBee-to-IEC61850 bridge application software is executed. When a ZigBee node appears in the ZigBee network and it sends temperature value, the algorithm described in section 4.5.2 updates IEDs local database. Other IEDs can then access stored temperature values using IEC 61850 protocol.

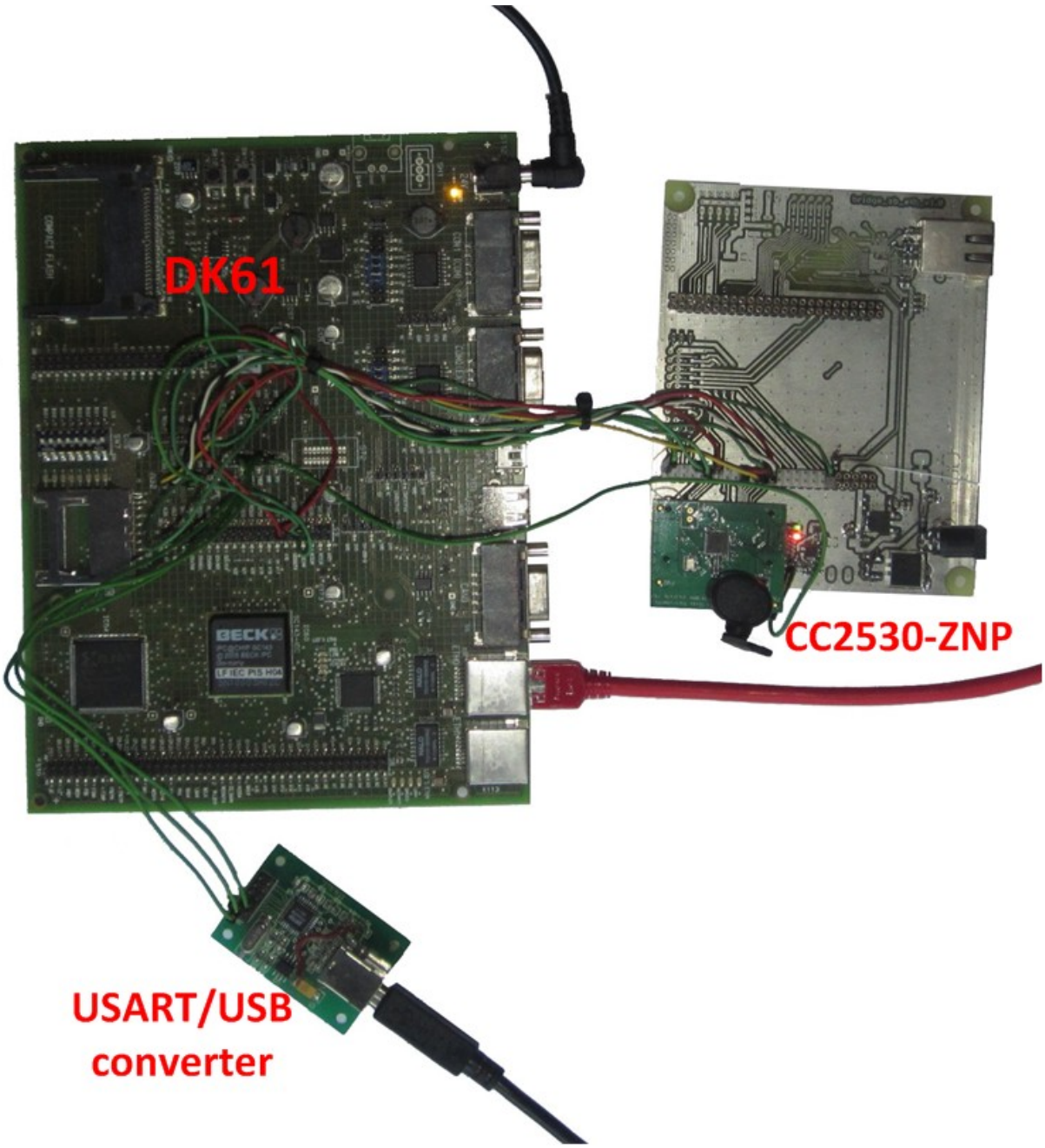


Figure 4.10. The prototype of the ZigBee-to-IEC61850 bridge.

5. CONCLUSION.

5.1. Summary of results.

The master thesis describes two versions of device which can be used to merge Ethernet-based and ZigBee networks.

First of them, the ZigBee-to-Ethernet bridge, uses TCP protocol to provide Ethernet-based network interface to ZigBee PAN network, handling data exchange between ZigBee nodes and network devices. The design process included both hardware and software. Schematic and PCB were designed, manufactured and the complete device was assembled and tested. This device is composed of STM32F4 microcontroller with ARM Cortex-M4 core and CC2530-ZNP ZigBee module. The software uses multiple RTOS tasks to handle ZigBee and TCP communication. The HTTP server on the bridge is also implemented, allowing the user to change network and device specific settings as well as to have an access to the ZigBee nodes via a web browser. Additional Java web application was written to provide MySQL database connectivity, handling and visualizing data from many PAN networks.

The second version of the device, the ZigBee-to-IEC61850 bridge, is intended to be used as a part of a substation automation system. It acts as an intelligent electrical device (IED) called "WSN network", and consists of logical devices that represent remote ZigBee nodes operating in an associated PAN network. The prototype device was built using DK61 evaluation board containing SC143 Embedded Web Controller and CC2530-ZNP ZigBee module. A demonstration application has been prepared, in which remote ZigBee nodes send temperature values to the bridge. These values are stored in a local database, and are made available through the IEC61850 to other devices in the network. The demonstration software integrates multiple libraries and software components supplied by hardware vendors.

The ZigBee-to-Ethernet bridge may be used wherever it is needed to have on-line access to ZigBee nodes, for example in a home/building automation system but also in factory environments. A special case of such environment, considered in this thesis is substation automation in which not TCP but IEC61850 protocol is often used. This is why a second version of the bridge, connecting ZigBee network and substation automation devices (IEDs) was also introduced.

The substation automation is an important issue related to the energy grid of the future – so called "smart grid". There is an ongoing effort to integrate wireless sensor networks in substation automation. This work is in the scope of the KIC-ActiveSubStations project, conducted as a joint research and development project between several European universities and industrial partners, under European Institute of Innovation & Technology (EiT)

sponsorship. Both versions of the presented ZigBee to cable network bridge can be used in such applications, providing necessary communication services.

5.2. Further studies and work.

A ZigBee-to-Ethernet bridge can be improved as follows:

- The system can be expanded with advanced functionality matched to the ZigBee profiles such as: Home Automation, Building Automation, Smart Energy etc. This includes changes in the firmware and Java web application.
- The web server on the bridge should allow not only to view data from the nodes, but also to control them.
- The PCB can be made smaller.

A ZigBee-to-IEC61850 bridge is also only a prototype and further studies and work can be carried out:

- A ZigBee profile suited to the real needs of a substation automation system should be proposed. This should begin with studies of the non-real-time applications in which wireless sensor networks can be beneficial.
- The software and the SCL files for the bridge may be expanded to allow configuration of the ZigBee network parameters.
- The complete device should be tested in a real substation.

Some of these issues are complicated and require a coordinated work of energy industry and wireless network (especially ZigBee) specialists. The key aspect is to recognize areas in which devices with ZigBee connectivity can solve actual problems, improve reliability or lower the cost of the system in a substation.

6. REFERENCES.

6.1. Bibliography.

[1] CISCO, wiki: I*nternetworking Bacics*: http://docwiki.cisco.com/wiki/Internetworking_Basics#OSI_Model_Physical_Layer.

[2] CISCO: *TCP/IP Overview, Document ID: 13769.*

[3] Cisco: *The TCP IP Model of Networking: sisco ccna networking fundamentals chapter 2*: http://www.youtube.com/watch?v=tCRBa3fTR3A.

[4] Radio-Electronics.com: I*EEE 802.15.4 Technology & Standard*: http://www.radio-electronics.com/info/wireless/ieee-802-15-4/wireless-standard-technology.php, www.radio-electronics.com.

[5] IEEE Std 802.15.4™-2003: Part 15.4: *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 1 October 2003.

[6] Gary Legg, *ZigBee: Wireless Technology for Low-Power Sensor Networks,* http://eetimes.com/design/communications-design/4017853/ZigBee-Wireless-Technology-for-Low-Power-Sensor-Networks.

[7] Jon T. Adams, *An Introduction to IEEE STD 802.15.4*, Freescale Semiconductor, Inc. , conference publication.

[8] Daintree Networks Inc, *Getting Started with ZigBee and IEEE 802.15.4.*

[9] ZigBee Alliance: ZigBee Specification, ZigBee document 053474r13, 1 December 2006.

[10] ZigBee Alliance: ZigBee Cluster Library Specification, ZigBee document 075123r02ZB, 29 May 2008.

[11] Jennic, ZigBee tutorial:

http://www.jennic.com/elearning/zigbee/files/html/module5/module5-1.htm.

[12] Wikipedia: *Data integrity*: http://en.wikipedia.org/wiki/Data_integrity.

[13] Cisco: *Ethernet Technologies*: http://docwiki.cisco.com/wiki/Ethernet_Technologies.

[14] YouTube, *Smart Grid Presentation Part 1*: http://www.youtube.com/watch?v=eOM4HyUcDoA&feature=related.

[15] KIC-InnoEnergy web page: http://www.kic-innoenergy.com/innovation-projects/active-sub-stations.html.

[16] Jessica Stromback, Christophe Dromacque, Mazin H. Yassin, VaasaETT: *The potential of smart meter enabled programs to increase energy and systems efficiency: a mass pilot comparison,* Global Energy Think Tank.

[17] Patrycja Batóg: *Indie i Chiny wyznaczają perspektywy światowego zapotrzebowania na energię*, www.energetyka.wnp.pl, July 20, 2011.

[18] IEC, IEC61850-6: Substation automation system configuration description language.

[19] IEC, IEC61850-5: Communication requirements for function and device models.

[20] IEC, IEC61850-7-1: Basic communication structure for substation and feeder equipment – Principles and models.

[21] IEC, IEC61850-7-2: Basic communication structure for substation and feeder equipment – Abstract communication service interface (ACSI).

[22] IEC, IEC61850-7-3: Basic communication structure for substation and feeder equipment – Common data classes.

[23] IEC, IEC61850-7-4: Basic communication structure for substation and feeder equipment – Compatible logical node classes and data classes.

[24] Yang Liu, Qiang Guan, Seung-Soo Han, Myeon-Song Choi, Seung-Jae Lee: *Research on Optimization of Process Bus in IEC 61850-Based Substation Communication Network*, The International Conference on Electrical Engineering 2009.

[25] R.P. Gupta, Member, IEEE: *Substation Automation Using IEC61850 Standard*, Fifteenth National Power Systems Conference (NPSC), IIT Bombay, December 2008.

[26] Klaus-Peter Brand, Wolfgang Wimmer: *The concept of IEC 61850. A new approach for communication in substation automation and beyond*, ABB review: Special Report IEC 61850.

[27] http://www.us-cert.gov/control_systems/practices/documents/Securing%20ZigBee %20Wireless%20Networks%20in%20Process%20Control%20System%20Environments.pdf.

[28] STMicroelectronics, data sheet *DM00035129, STM32F415xx, STM32F417xx.*

[29]-STMicroelectronics reference manual RM0090, *STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs.*

[30] Texas Instruments, ZigBee-PRO network processor description: *CC2530-ZNP, ZigBee PRO Network Processor: accelerate your ZigBee Development.*

[31] Texas Instruments, data sheet, SWRS081B *CC2530F32, CC2530F64,CC2530F128, CC2530F256. A True System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications.*

[32] Media Independent Interface (MII) description: http://www.hardwarebook.info/Media_Independent_Interface_%28MII%29.

[33] National Semiconductor, data sheet: *DP83848 PHYTER – Commercial Temperature Single Port 10/100 Mb/s Ethernet Physical Layer Transceiver.*

[34] Pulse a technitrol company, data sheet: *PulseJack 1x1 Tab-DOWN RJ45.*

[35] Texas Instruments, CC2530EM Reference Design: http://www.ti.com/tool/cc2530em.

[36] Atollic TrueSTUDIO website – http://www.atollic.com/.

[37] Article about LwIP: http://en.wikipedia.org/wiki/LwIP.

[38] LwIP Wiki – http://lwip.wikia.com/wiki/LwIP_Wiki.

[39] Texas Instruments, document slyb134c.pdf: *ZigBee Wireless Networking Overview.*

[40] FreeRTOS Overview: http://www.freertos.org/FreeRTOS_Features.html.

[41] Oracle, *Java Server Faces Technology –* http://docs.oracle.com/javaee/5/tutorial/doc/bnaph.html.

[42] Hibernate web page: http://www.hibernate.org/.

[43] Wikipedia: *Hibernate (Java)*, http://en.wikipedia.org/wiki/Hibernate_%28Java%29.

[44] NetBeans tutorial: *Using Hibernate in a Web Application*: http://netbeans.org/kb/docs/web/hibernate-webapp.html.

[45] Beck IPC GmbH, *Getting Stared, IPC@CHIP Embedded Web Controller Family.*

[47] SystemCORP Pty Ltd. *IEC 61850 Protocol API User Manual: Protocol Integration Stack.*

[46] Beck IPC GmbH, *IPC@CHIP® RTOS Documentation [Build 07.11.2011]*: http://www.beck-ipc.com/files/api/scxxx/index.htm.

[47] Zin Kyaw, System Application Engineer, Texas Instuments: *Creating a ZigBee SmartEnergy Device with the MSP430F54xx and the CC2530-ZNP (ZigBee Pro Network Processor)*

6.2. APPENDIX A: IEEE 802.15.4 MAC frame formats.

6.2.1. Beacon frame format.

Figure 6.1 presents beacon frame format. Superframe specification field in MAC payload includes subfields which are required in beacon-enabled mode of a transmission. Beacon order (BO) subfield specifies the transmission interval of the beacon, and is calculated from the following equation:

$$BI = aBaseSuperframeDuration \cdot 2^{BO} \quad ,$$

where:

BI – Beacon interval,

BO – Beacon Order ($0 \leq BO \leq 14$).

Superframe order specifies the length of time when superframe is active, and is calculated from:

$$SD = aBaseSuperframeDuration \cdot 2^{SO} \quad ,$$

where:

SD – Superframe Duration,

SO – Superframe Order ($0 \leq SO \leq BO \leq 14$).

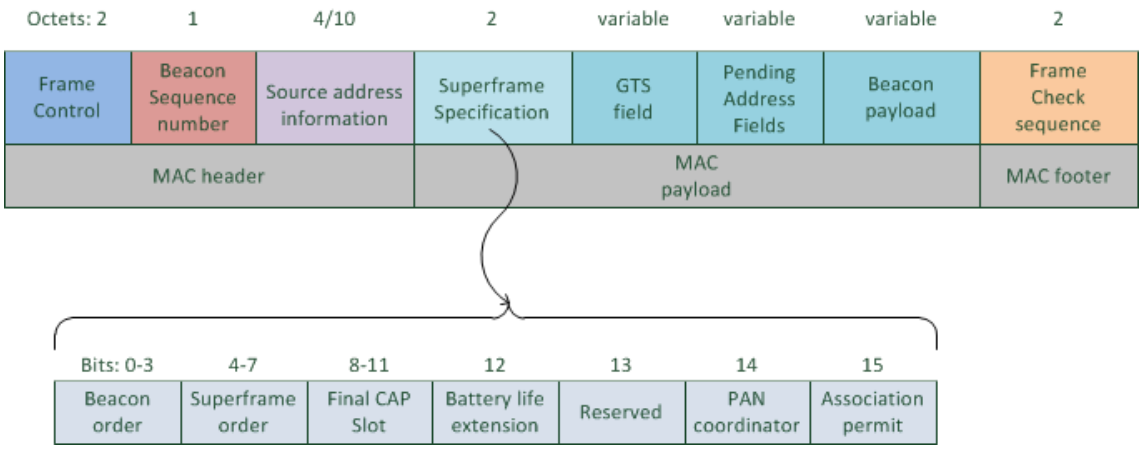


Figure 6.1. Beacon frame format [5].

Final CAP Slot says which is the last superframe slot utilized by the Contention Active Period. Finally the PAN coordinator bit is set to 1 if beacon frame is being transmitted by a PAN coordinator.

Further beacon MAC payload field name is GTS, which has GTS the specification subfield, the direction subfield and the list subfield. It includes settings of Guaranted Time Slots, and addresses of the devices which uses this mechanism.

Pending Address Fields include addresses of devices, to which coordinator has pending messages.

Beacon payload contains space for information which may use higher layer protocols to implement other functions into Beacon frame.

6.2.2. Command frame format.

The second type of frame is the command frame (Figure 6.2). Command frames are used by a coordinator and network devices to react to events, manage and create a network.

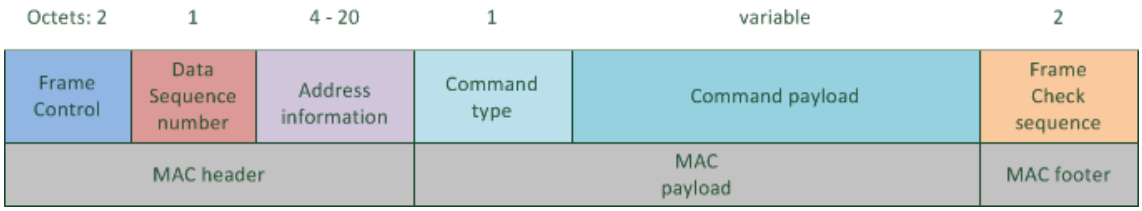


Figure 6.2. MAC command frame format [5].

There are following types of command defined in the IEEE 802.15.4 standard:

- association request,
- association response,
- disassociation notification,
- data request,
- PAN ID conflict notification,
- Orphan Notification,
- Beacon request,
- Coordinator realignment,
- GTS request.

6.2.3. Data frame format.

A data frame contains the data which next higher layer has requested the MAC sublayer to transmit [5], and is presented in figure 6.3

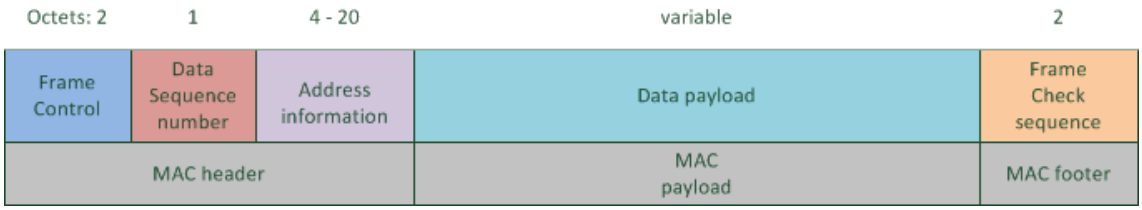


Figure 6.3. MAC data frame format [5].

6.2.4. Acknowledge frame format.

The last type of frame is the acknowledgment frame illustrated in figure 6.4, and it's MAC Protocol Data Unit consist of only 5 octets. The frame type subfield in the frame control field contains the value indicating type of frame which is being acknowledged (Beacon frame, Data frame, Acknowledgment frame, MAC command frame). The data sequence number is used to inform which frame is acknowledged.

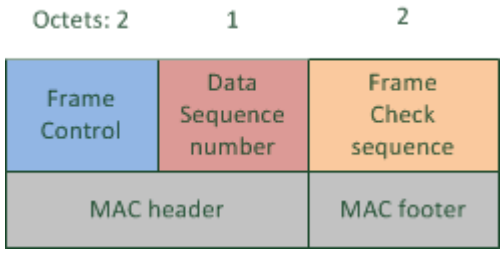


Figure 6.4. MAC acknowledgment frame format [5].

6.3. APPENDIX B: HTML files for HTTP server implemented in a ZigBee-to-Ethernet bridge.

6.3.1. main_page.html

“main_page.html” is the HTML file served by the bridge. Using web browser user can change setting of the bridge and see measurements of sensors within the PAN network.

```
<html>
<head>
</head>
<BODY bgcolor="#FFFFFF" text="#2477E6">
<div id="container" >
 <div id="header" style="background-color:#FFFFFF;">
  <h1 style="margin-bottom:0;">
   <img src="bridge_logo.jpg" alt="bridge_logo" />
  </h1>
 </div>
 <div id="menu" style="background-color:#FFFFFF;float:left;">
  <table border="1">
   <tr>
    <td>ZB IEEE address</td>
    <td> 0x #bridge IEEE address# </td>
   </tr>
   <tr>
    <td>ZB short address</td>
          <td> 0x #bridge short network address# </td>
   </tr>
   <tr>
    <td>Current channel:</td>
    <td> #current ZigBee channel# </td>
   </tr>
   <tr>
```

```
 <td>network <br> PAN id extended</td>
 <td> 0x #extended PAN network ID#</td>
</tr>
<form name="input" action="save_settings" method="get">
 <tr>
  <td>IP address</td>
  <td><input type="text" name="ip_addr" value="#bridge IP address#"/></td>
 </tr>
 <tr>
  <td>Subnet mask</td>
  <td><input type="text" name="subnet_mask" value="#bridge subnet mask#"/></td>
 </tr>
 <tr>
  <td>Default gateway</td>
  <td><input type="text" name="def_gate" value="#bridge default gateway#"/></td>
 </tr>
 <tr>
  <td>User description</td>
  <td><input type="text" name="user_desc" value="#user description#" /></td>
 </tr>
 <tr>
  <td>PAN ID</td>
  <td><input type="text" name="pan_id" value="#bridge PAN ID#" /></td>
 </tr>
 <tr>
  <td>Select channels <br> in which <br> bridge operates</td>
        <td>
   <input type="checkbox" name="chlst" value="c" checked /> channel 11 <br />
         <input type="checkbox" name="chlst" value="c" /> channel 12 <br />
   <input type="checkbox" name="chlst" value="c" /> channel 13 <br />
   <input type="checkbox" name="chlst" value="c" /> channel 14 <br />
   <input type="checkbox" name="chlst" value="c" /> channel 15 <br />
   <input type="checkbox" name="chlst" value="c" /> channel 16 <br />
   <input type="checkbox" name="chlst" value="c" /> channel 17 <br />
   <input type="checkbox" name="chlst" value="c" /> channel 18 <br />
```

```
    <input type="checkbox" name="chlst" value="c" /> channel 19 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 20 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 21 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 22 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 23 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 24 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 25 <br />
    <input type="checkbox" name="chlst" value="c" /> channel 26 <br />
   </td>
  </tr>
  <tr>
   <td><input type="submit" value="save" /></td>
  </form>
   <form name="input" action="reset_znp" method="get">
    <td><input type="submit" value="reset ZNP" /></td>
  </tr>
        </form>
 </table>
</div>
<div id="measurements" style="background-color:#EEEEEE;float:left;">
 <iframe src="measurements.htm" width="800" height="600"></iframe>
</div>
<div id="footer" style="background-color:
#FFA500;clear:both;text-align:center;">
  Copyright Dominik Nowak
</div>
</div>
</BODY>
</html>
```

6.3.2. measurements.htm.

“measurements.htm” is displayed in the “iframe” of the main page. It displays in tabular form data from all nodes within PAN network in which the bridge is a coordinator.

```
<html>
<body
onLoad=\"window.setTimeout(&quot;location.href='/measurements.htm'&quot;,1000)\">
<h1 align=\"center\"> Data from sensors within the PAN </h1>
<table border=\"1\" align=\"center\" >
 <tr>
  <th><h2>network address</h2></th>
  <th><h2>temperature</h2></th>
  <th><h2>timestamp</h2></th>
 <tr>
  <td>0x#addr1"</td>
  <td>XXX *C</td>
  <td> YYY </td>
 </tr>
 <tr>
  <td>0x#addr2"</td>
  <td>ZZZ *C</td>
  <td> YYY </td>
 </tr>
 <tr>
  <td>0x#addr3"</td>
  <td>XXX *C</td>
  <td> YYY </td>
 </tr>
</table>
</body>
</html>
```

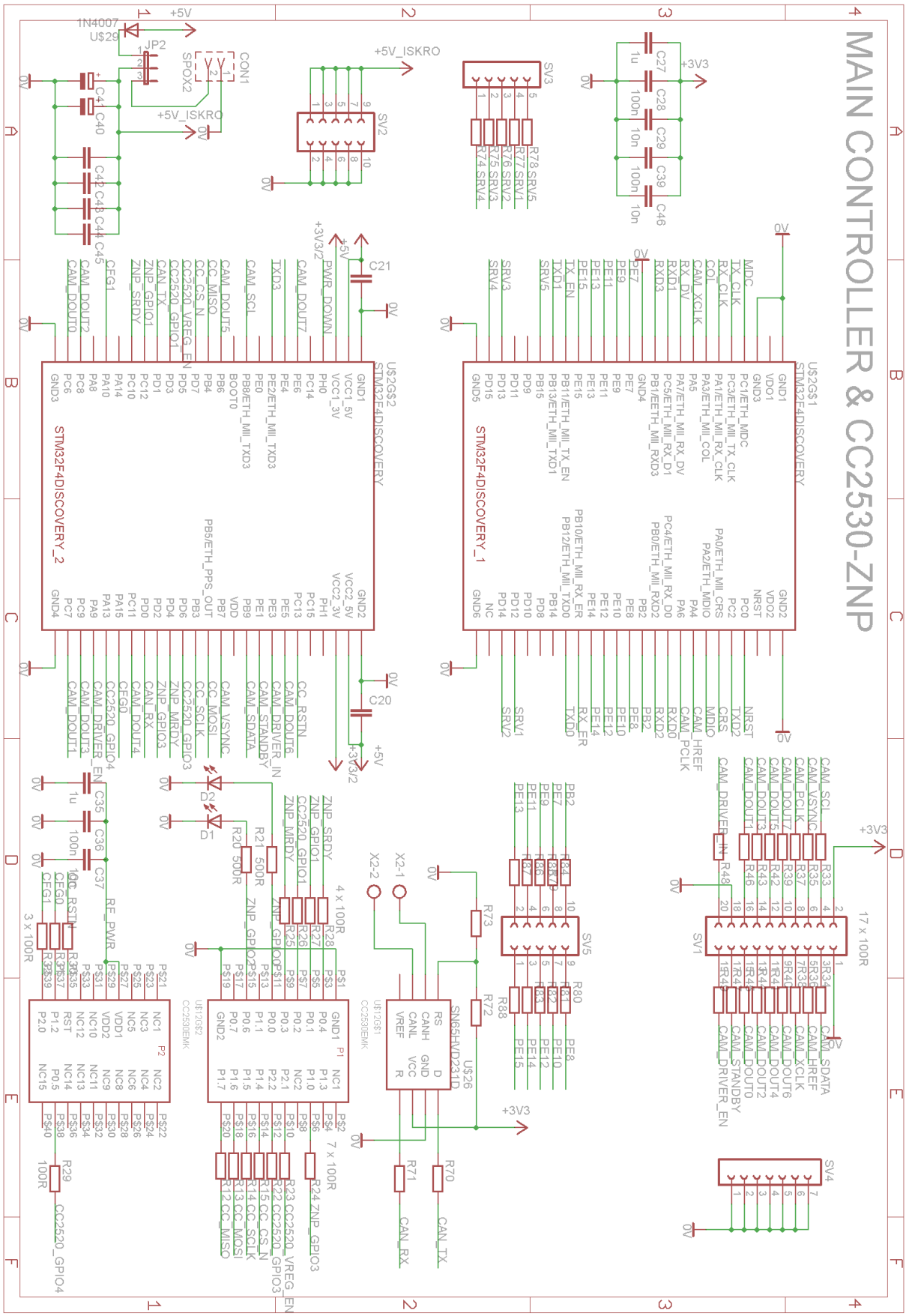
6.4. APPENDIX C: WSN Manager Java Web Application – HTML file of the main web page.

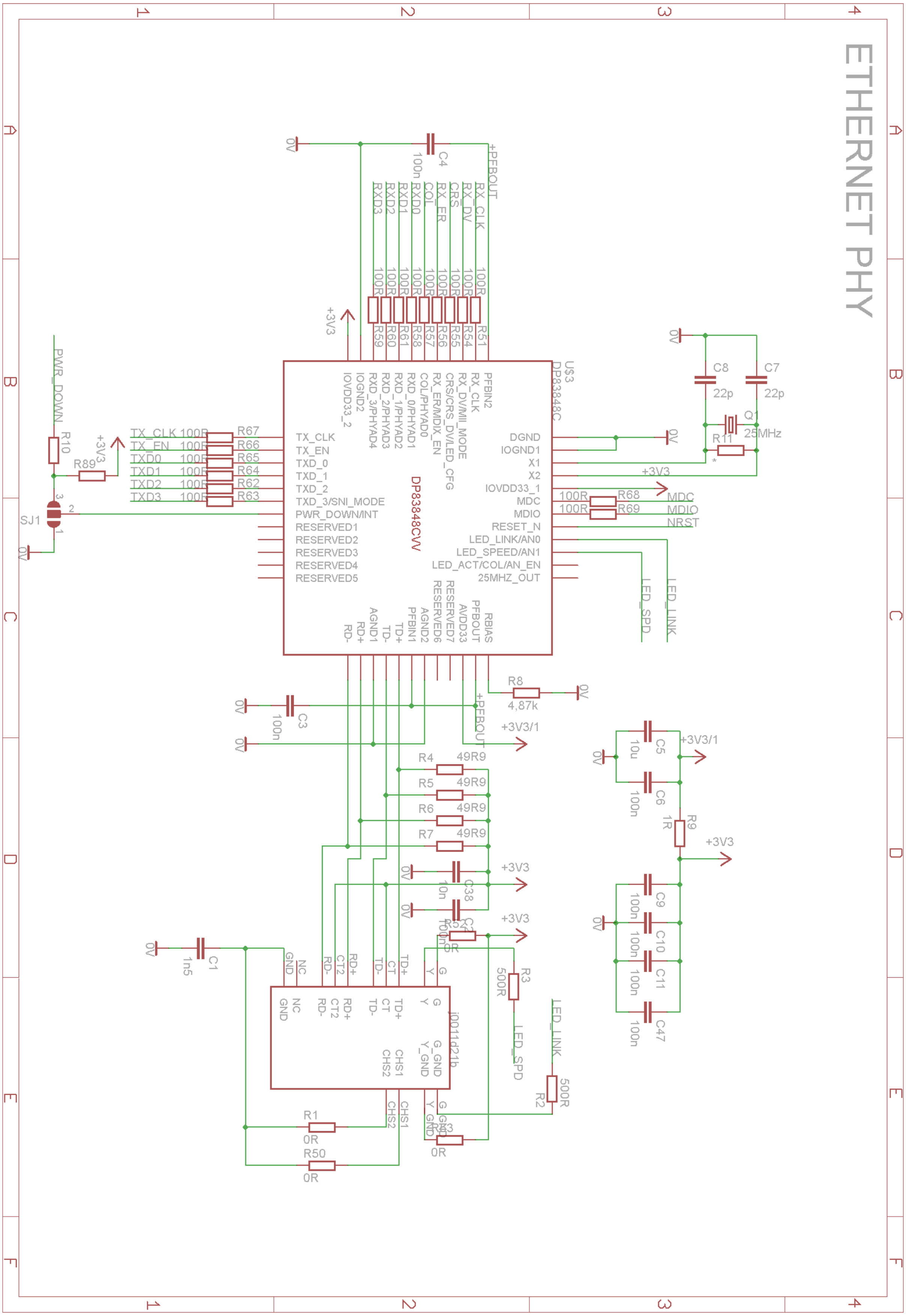
“WSN Manager v1” XHTML page .

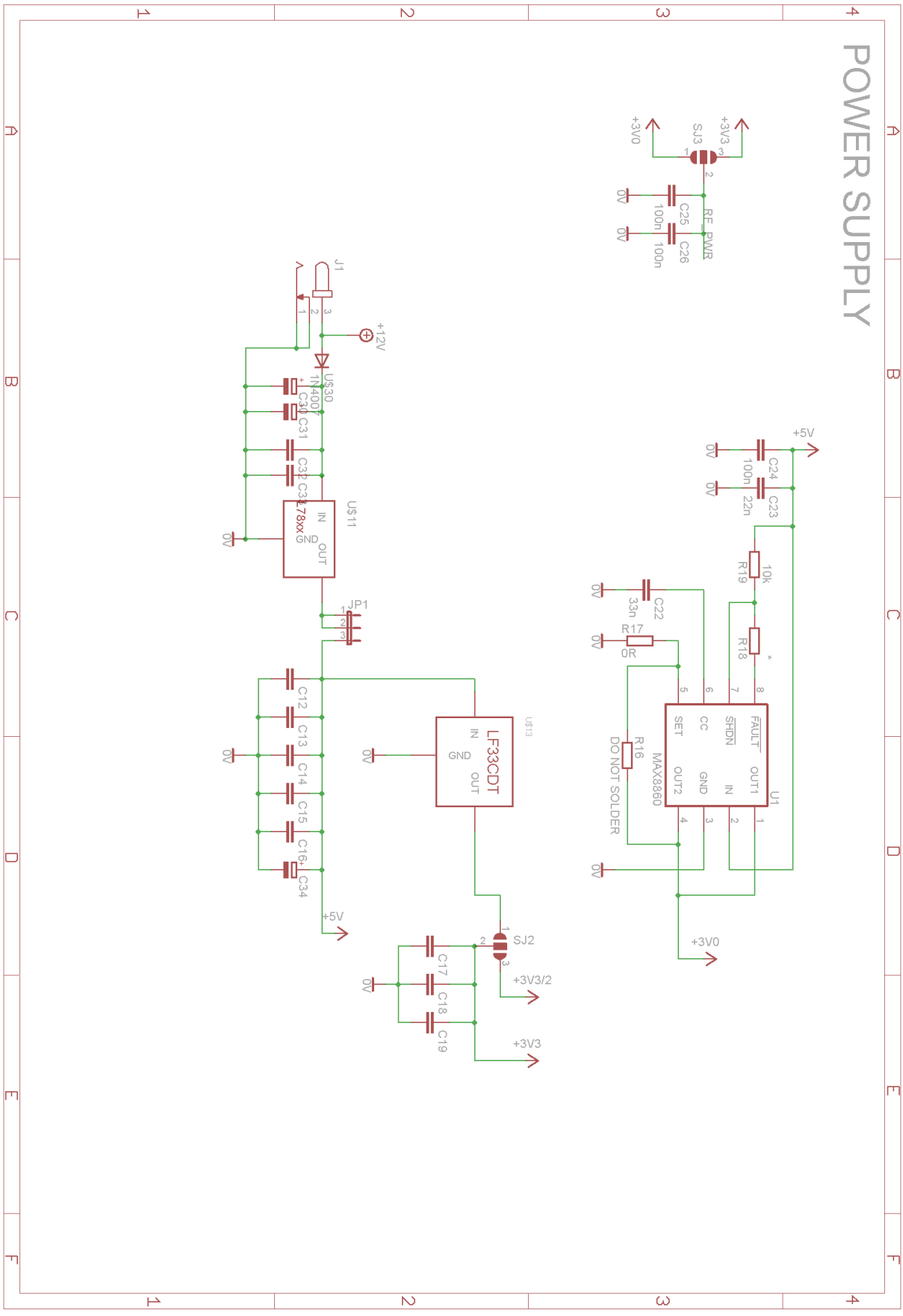
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">

<h:head>
  <title>WSN Manager v1</title>
</h:head>

<img src="logo1.png" alt="wsn_manager" />

<h:body>
 <h:form>
   <p:megaMenu>
     <p:submenu label="Add bridge" icon="ui-icon-check">
       <p:column>
         <p:panel header="Input bridge parameters:">
           <h:panelGrid columns="2" cellpadding="2">
             <h:outputText value="Bridge ID " />
             <p:inputText id="input1" value="#{bridgeZbEth.bridgeID}" />

             <h:outputText value="IEEE address " />
             <p:inputText id="input2" value="#{bridgeZbEth.IEEEaddr}" />

             <h:outputText value="description " />
             <p:inputText id="input3"  value="#{bridgeZbEth.description}" />
```

```
            <p:commandButton  value="Reset" type="reset" />
            <p:commandButton id="cmd1" value="Save bridge"
           action="#{bridgeZbEth.saveBridgeInDB()}" />

          </h:panelGrid>
        </p:panel>
      </p:column>
    </p:submenu>
    <p:submenu label="Connect to the bridge" icon="ui-icon-check">
      <p:column>
        <p:panel header="Input bridge network settings:">
          <h:panelGrid columns="2" cellpadding="2">
            <h:outputText value="Bridge IP " />
            <p:inputText id="input4" value="#{bridgeZbEth.bridge_IP}" />

            <h:outputText value="port " />
            <p:inputText id="input5" value="#{bridgeZbEth.bridge_port}" />

            <p:commandButton  value="Reset" type="reset" />
            <p:commandButton id="cmd2" value="Start connection"
           action="#{bridgeZbEth.startBridge()}" />

          </h:panelGrid>
        </p:panel>
      </p:column>
    </p:submenu>
  </p:megaMenu>
  <p:commandButton id="cmd3" value="update plot" action="#{bridgeZbEth.updatePlot()}"
          update="linear" />
  <p:lineChart id="linear" value="#{bridgeZbEth.linearModel}" legendPosition="e"
     title="data from sensors" minY="0" maxY="30" style="height:500px"/>
 </h:form>
</h:body>
</html>
```

6.5. APPENDIX D: Schematic of a ZigBee-to-Ethernet bridge.




ETHERNET PHY
DP83848CVV
U$3
DP83848C
TX_CLK
TX_EN
TXD_0
TXD_1
TXD_2
TXD_3/SNI_MODE
PWR_DOWN/INT
RESERVED1
RESERVED2
RESERVED3
RESERVED4
RESERVED5
DGND
IOGND1
X1
X2
IOVDD33_1
MDC
MDIO
RESET_N
LED_LINK/AN0
LED_SPEED/AN1
LED_ACT/COL/AN_EN
25MHZ_OUT
PFBIN2
RX_CLK
RX_DV/MII_MODE
CRS/CRS_DV/LED_CFG
RX_ER/MDIX_EN
COL/PHYAD0
RXD_0/PHYAD1
RXD_1/PHYAD2
RXD_2/PHYAD3
RXD_3/PHYAD4
IOGND2
IOVDD33_2
RBIAS
PFBOUT
AVDD33
RESERVED7
RESERVED6
AGND2
PFBIN1
TD+
TD-
AGND1
RD+
RD-
+PFBOUT
+3V3
+3V3/1
0V
C4
100n
C3
C5
10u
C6
C7
C8
22p
Q1
25MHz
R11
R8
4,87k
R9
1R
R4
R5
R6
R7
49R9
C38
10n
C9
C10
C11
C47
C1
1n5
R1
R50
R52
R53
0R
R2
R3
500R
R10
R89
SJ1
PWR_DOWN
MDC
MDIO
NRST
LED_SPD
LED_LINK
j0011d21b
GND
NC
RD-
CT2
RD+
TD-
CT
TD+
Y
G
CHS1
CHS2
G_GND
Y_GND
1
2
3
4
A
B
C
D
E
F


POWER SUPPLY
+3V0
SJ3
+3V3
RF_PWR
C25
100n
C26
100n
0V
J1
+12V
U$30
1N4007
C30
C31
C32
C33
U$11
78xx
IN
GND
OUT
JP1
C12
C13
C14
C15
C16
C34
+5V
U$13
LF33CDT
IN
GND
OUT
SJ2
+3V3/2
+3V3
C17
C18
C19
C24
100n
C23
22n
+5V
R19
10k
C22
33n
R17
0R
R18
*
R16
DO NOT SOLDER
U1
MAX8860
SET
CC
SHDN
FAULT
OUT2
GND
IN
OUT1
+3V0

6.6. APPENDIX E: SCL file used in a ZigBee-to-IEC61850 bridge.

```
<?xml version="1.0" encoding="UTF-8"?>
<SCL xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.iec.ch/61850/2003/SCL">
  <Header id="" version="3"/>
  <Communication>
    <SubNetwork name="SubNetworkName">
      <ConnectedAP iedName="DK61" apName="SubstationRing1">
        <Address>
          <P type="OSI-AP-Title">1,1,9999,1</P>
          <P type="OSI-AE-Qualifier">12</P>
          <P type="OSI-PSEL">00000001</P>
          <P type="OSI-SSEL">0001</P>
          <P type="OSI-TSEL">0001</P>
          <P type="IP">192.168.1.19</P>
          <P type="IP-SUBNET">255.255.255.0</P>
          <P type="IP-GATEWAY">192.168.1.1</P>
        </Address>
      </ConnectedAP>
    </SubNetwork>
  </Communication>
  <IED type="RTUType" manufacturer="SystemCORP Pty Ltd" configVersion="1.0" name="DK61">
    <Services/>
    <AccessPoint name="SubstationRing1">
      <Server timeout="30">
        <Authentication/>
        <LDevice inst="LD_ZigBee_Node" desc="">
          <LN0 lnClass="LLN0" inst="" lnType="LLN0_1">
            <DataSet name="Measurement_DataSet">
              <FCDA ldInst="LD_ZigBee_Node" prefix="LN_" lnClass="STMP" lnInst="1" doName="Tmp"
daName="mag.i" fc="MX"/>
            </DataSet>
            <ReportControl rptID="myRepURCB_ID" confRev="0" intgPd="5000"
datSet="Measurement_DataSet" name="UNBUFFERED_RCB" desc="Unbuf RCB">
              <TrgOps dchg="true" qchg="true" dupd="true" period="true"/>
              <OptFields seqNum="true" timeStamp="true" dataSet="true" reasonCode="true" entryID="true"
configRef="true"/>
              <RptEnabled max="6">
                <ClientLN iedName="BridgeClient" ldInst="none" prefix="" lnClass="IHMI" lnInst="1"/>
              </RptEnabled>
            </ReportControl>
          </LN0>
          <LN lnClass="STMP" inst="1" prefix="LN_" lnType="STMP_0">
            <DOI name="Tmp">
              <SDI name="mag">
                <DAI name="i">
                  <Private type="SystemCorp_Generic">
<SystemCorp_Generic:GenericPrivateObject Field1="1" Field2="1" Field3="1" Field4="0" Field5="0"
xmlns:SystemCorp_Generic="http://www.systemcorp.com.au/61850/SCL/Generic"/></Private>
                </DAI>
              </SDI>
              <DAI name="q">
                <Private type="SystemCorp_Generic">
<SystemCorp_Generic:GenericPrivateObject Field1="1" Field2="1" Field3="2" Field4="0" Field5="0"
xmlns:SystemCorp_Generic="http://www.systemcorp.com.au/61850/SCL/Generic"/></Private>
              </DAI>
              <DAI name="t">
                <Private type="SystemCorp_Generic">
<SystemCorp_Generic:GenericPrivateObject Field1="1" Field2="1" Field3="3" Field4="0" Field5="0"
```

```
xmlns:SystemCorp_Generic="http://www.systemcorp.com.au/61850/SCL/Generic"/></Private>
            </DAI>
          </DOI>
        </LN>
      </LDevice>
    </Server>
  </AccessPoint>
</IED>
<DataTypeTemplates>
  <LNodeType lnClass="LLN0" id="LLN0_1">
    <DO name="Mod" type="ENC_1"/>
    <DO name="Beh" type="ENS_0"/>
    <DO name="Health" type="ENS_0"/>
    <DO name="NamPlt" type="LPL_0"/>
  </LNodeType>
  <LNodeType lnClass="STMP" id="STMP_0">
    <DO name="Mod" type="INC_0"/>
    <DO name="Beh" type="INS_0"/>
    <DO name="Health" type="INS_0"/>
    <DO name="NamPlt" type="LPL_0"/>
    <DO name="Tmp" type="MV_0"/>
  </LNodeType>
  <DOType cdc="LPL" id="LPL_0">
    <DA fc="DC" name="vendor" bType="VisString255"/>
    <DA fc="DC" name="swRev" bType="VisString255"/>
    <DA fc="DC" name="d" bType="VisString255"/>
  </DOType>
  <DOType cdc="INS" id="ENS_0" desc="Integer status">
    <DA dchg="true" fc="ST" name="stVal" bType="Enum"/>
    <DA qchg="true" fc="ST" name="q" bType="Quality"/>
    <DA fc="ST" name="t" bType="Timestamp"/>
  </DOType>
  <DOType cdc="ENC" id="ENC_1" desc="Controllable integer status">
    <DA dchg="true" fc="ST" name="stVal" bType="Enum"/>
    <DA qchg="true" fc="ST" name="q" bType="Quality"/>
    <DA fc="ST" name="t" bType="Timestamp"/>
    <DA dchg="true" fc="CF" name="ctlModel" bType="Enum" type="CtlModels"/>
  </DOType>
  <DOType cdc="INC" id="INC_0" desc="Controllable integer status">
    <DA dchg="true" fc="ST" name="stVal" bType="INT32"/>
    <DA qchg="true" fc="ST" name="q" bType="Quality"/>
    <DA fc="ST" name="t" bType="Timestamp"/>
    <DA fc="CF" name="ctlModel" bType="Enum" type="CtlModels"/>
  </DOType>
  <DOType cdc="INS" id="INS_0" desc="Integer status">
    <DA dchg="true" fc="ST" name="stVal" bType="INT32"/>
    <DA qchg="true" fc="ST" name="q" bType="Quality"/>
    <DA fc="ST" name="t" bType="Timestamp"/>
  </DOType>
  <DOType cdc="MV" id="MV_0" desc="Measured value">
    <DA dchg="true" fc="MX" name="mag" bType="Struct" type="AnalogueValue_0"/>
    <DA qchg="true" fc="MX" name="q" bType="Quality"/>
    <DA fc="MX" name="t" bType="Timestamp"/>
  </DOType>
  <DAType id="AnalogueValue_0">
    <BDA name="i" bType="INT32"/>
  </DAType>
  <EnumType id="ctlModel">
    <EnumVal ord="0">status-only</EnumVal>
    <EnumVal ord="1">direct-with-normal-security</EnumVal>
    <EnumVal ord="2">sbo-with-normal-security</EnumVal>
    <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
    <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
```

```
    </EnumType>
    <EnumType id="Mod">
      <EnumVal ord="1">on</EnumVal>
      <EnumVal ord="2">blocked</EnumVal>
      <EnumVal ord="3">test</EnumVal>
      <EnumVal ord="4">test/blocked</EnumVal>
      <EnumVal ord="5">off</EnumVal>
    </EnumType>
    <EnumType id="Health">
      <EnumVal ord="1">Ok</EnumVal>
      <EnumVal ord="2">Warning</EnumVal>
      <EnumVal ord="3">Alarm</EnumVal>
    </EnumType>
    <EnumType id="OrCat">
      <EnumVal ord="0">not-supported</EnumVal>
      <EnumVal ord="1">bay-control</EnumVal>
      <EnumVal ord="2">station-control</EnumVal>
      <EnumVal ord="3">remote-control</EnumVal>
      <EnumVal ord="4">automatic-bay</EnumVal>
      <EnumVal ord="5">automatic-station</EnumVal>
      <EnumVal ord="6">automatic-remote</EnumVal>
      <EnumVal ord="7">maintenance</EnumVal>
      <EnumVal ord="8">process</EnumVal>
    </EnumType>
    <EnumType id="CtlModels">
      <EnumVal ord="0">status-only</EnumVal>
      <EnumVal ord="1">direct-with-normal-security</EnumVal>
      <EnumVal ord="2">sbo-with-normal-security</EnumVal>
      <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
      <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
    </EnumType>
  </DataTypeTemplates>
</SCL>
```

6.7. APPEDNIX F: implementation of the function used to update local database of a ZigBee-to-IEC61850 bridge.

```
void huge MeasurementHandler(void)
{
  static unsigned long int temperatureValue[OBJECTS];          //temperature value
  IEC61850_ObjectData UpdateValue = {0};                       // Value to send on Change
  IEC61850_ObjectID Object = {0};                              // ID of the Object
  unsigned short  int usiQuality = 0;                          // Local Quality
  tNTPTimeStamp tNTPTime = {0};                                // Local Time Stamp

  int zbNodeindex = 1;

  unsigned char data12B[12];

  while(1)          // Indefinite Loop
  {
    if(0 == RTX_Get_Msg(temperature_Msg.msgID,(void far*) data12B)) {
      if(zbNodeindex = verifyMAC(data12B)) {
        temperatureValue[zbNodeindex-1] = (unsigned long int)data12B[8];

        /* Common Field to all Index */
        Object.uiField1 = zbNodeindex; // Object Number - there is currently only one ZigBee node
        Object.uiField2 = MEASUREMENT_INPUT;  // Object Type

        /* Object Value */
        UpdateValue.pvData = &temperatureValue[zbNodeindex-1];
        UpdateValue.ucType = IEC61850_DATATYPE_INT32;
        UpdateValue.uiBitLength = 32;

        /* Object Value Index */
        Object.uiField3 = VALUE_INDEX;

        /* Update Value in the database */
        atObj[MEASURINPUT_INDEX][zbNodeindex-1].ucObjectValue = temperatureValue[zbNodeindex-1];

        /* Send Update for Value */
        IEC61850_Update(myServer, &Object, &UpdateValue);

        usiQuality = 0;

        if(atObj[MEASURINPUT_INDEX][zbNodeindex-1].usiObjectQuality != usiQuality){
          /* Object Quality  */
          UpdateValue.pvData = &usiQuality;
          UpdateValue.ucType = IEC61850_DATATYPE_QUALITY;
          UpdateValue.uiBitLength = IEC61850_QUALITY_BITSIZE;
          Object.uiField3 = QUALITY_INDEX;

          /* Update Quality in the database */
          atObj[MEASURINPUT_INDEX][zbNodeindex-1].usiObjectQuality    = usiQuality;

          /* Send Update for Quality */
          IEC61850_Update(myServer, &Object, &UpdateValue);
        }

        /* Send Time */
        /* Convert to 61850 Time */
        ConvertTo61850Time(&tNTPTime);
        UpdateValue.pvData = &tNTPTime;
        UpdateValue.ucType = IEC61850_DATATYPE_TIMESTAMP;
```

```
            UpdateValue.uiBitLength = IEC61850_TIMESTAMP_BITSIZE;
            Object.uiField3 = TIME_STAMP_INDEX;

            /* Update Time in the database */
            memcpy(&atObj[MEASURINPUT_INDEX][zbNodeindex-1].tObjectTime, &tNTPTime,
                                    sizeof(tNTPTimeStamp));

            /* Send Update for Time Stamp */
            IEC61850_Update(myServer, &Object, &UpdateValue);
          }
        }
        RTX_Sleep_Time(50);
      }
    }
```

6.8. APPENDIX G: Implementation of the list for a ZigBee-to-Ethernet bridge.

In the “HTTP task” the following structure is used to store data from a single node:

```
typedef struct {
      void* next;
      void* before;
      uint8 flag_free;
      uint16 groupId;          /* Message's group ID - 0 if not set */
      uint16 clusterId;        /* Message's cluster ID */
      afAddrType_t srcAddr;    /* Source Address it's an InterPAN message */
      uint16 macDestAddr;      /* MAC header destination short address */
      uint8 endPoint;          /* destination endpoint */
      uint8 wasBroadcast;      /* TRUE if network destination was a
                                  broadcast address */
      uint8 LinkQuality;      /* The link quality of the received data
                              frame */
      uint8 correlation;       /* The raw correlation value of the received
                                  data frame */
      int8  rssi;              /* The received RF power in units dBm */
      uint8 SecurityUse;       /* deprecated */
      uint32 timestamp;        /* receipt timestamp from MAC */
      uint8 nwkSeqNum;         /* network header frame sequence number */
      uint16 temperature;
      uint16 voltage;
}SensorTypedef_t;
```

It is assumed that nodes can be dynamically added and removed from the network, so functions allowing that operations had to be implemented:

```
void sensors_init(void);
SensorTypedef_t* sensors_malloc(void);
void sensors_free(SensorTypedef_t* stp);
void sensors_process_incomming_data(afIncomingMSGPacket_static_t* msg);
```

The sensors_init() function sets flag_free parameter to one for all elements in the table:

```
static SensorTypedef_t sensors[MAX_SENS_NR];
```

All elements in the sensors[] table are now empty boxes, which can be occupied by new nodes dynamically connected to the PAN network.

The function used to allocate memory for new sensor is implemented as follows:

```
SensorTypedef_t* sensors_malloc(void)
{
      uint16_t i;
      for(i=0;i<MAX_SENS_NR;i++) {
            if(sensors[i].flag_free == 1) {
```

```
                    sensors[i].flag_free = 0;
                    sensors[i].next = NULL;
                    sensors[i].before = NULL;
                    sensors_nr ++;
                    return &sensors[i];
                }
        }
        return NULL;
}
```

The following function is used to process in the “HTTP task” messages received from the “ZNP task”:

```
void sensors_process_incomming_data(afIncomingMSGPacket_static_t* msg)
{
    uint16_t i;
    SensorTypedef_t* stp;
    if(sensors_nr==0) {
          root = sensors_malloc();
          update_sensor(root,msg);
          return;
    }
    for(stp = root; stp != NULL; stp = stp->next) {
          if(msg->srcAddr.addr.shortAddr == stp->srcAddr.addr.shortAddr) {
                update_sensor(stp,msg);
                return;
          } else {
                if(stp->next == NULL) {
                      stp->next = sensors_malloc();
                      if(stp->next == NULL) {
                            return;//error - no free memory
                      } else {
                            ((SensorTypedef_t*)(stp->next))->before = stp;
                            update_sensor(stp->next,msg);
                      }
                      return;
                }
          }
    }

    return; //error - no free memory
}
```

When a ZigBee frame from some node arrives to the bridge, at first it is processed by “task_znp”, and then is sent as afIncomingMSGPacket_static_t type element via FreeRTOS queue. If the new message in queue is available, in the “task_http” it is read and forwarded as a parameter to sensor_process_incomming_data() function. Within it, at first it is checked whether there is no sensors saved in the sensors[] table. In this case pointer to first allocated node structure is assigned to the root pointer, which is declared as:

```
SensorTypedef_t* root;
```

It represents a first element of the bidirectional list. The update_sensor() function simply

prescribes elements from the location pointed by the “msg” , to the location pointed by the “root”. If there is at least one sensor saved in the list, the list is reviewed to find given element, which address is the same as address of structure pointed by the “msg”. If element is found its data is updated, otherwise new node is allocated, by the sensor_malloc() function and then is connected to the list.

The last function used to process data from “ZNP task” is as follows:

```
void sensors_free(SensorTypedef_t* stp)
{
	if((stp->before != NULL) && (stp->next != NULL)) {
		((SensorTypedef_t*)(stp->before))->next = stp->next;
		((SensorTypedef_t*)stp->next)->before = stp->before;
	}
	if((stp->before != NULL) && (stp->next == NULL)) {
		((SensorTypedef_t*)(stp->before))->next = NULL;
	}

	if((stp->before == NULL) && (stp->next != NULL)) {
		((SensorTypedef_t*)(stp->next))->before = NULL;
		root = (SensorTypedef_t*)(stp->next);
	}

	if((stp->before == NULL) && (stp->next == NULL)) {
		return;
	}
	stp->flag_free = 1;
	sensors_nr--;
}
```

If some node has to be removed from the network, or timeout occurs the corresponding location storing its data must be freed by a call to sensor_free() function. Because sensor data is stored in the list, not only flag_free has to be set to one, but also the neighboring elements have to be connected.