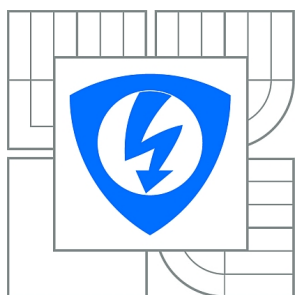


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



**FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLOGIÍ**

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

INTERCONNECTION OF IEEE 802.15.4 AND ETHERNET NETWORKS

PROPOJENÍ SÍTÍ IEEE 802.15.4 A ETHERNET

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. KAREL PAVLATA

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. PETR FIEDLER, Ph.D.

BRNO 2011



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Bc. Karel Pavlata

ID: 83381

Ročník: 2

Akademický rok: 2010/2011

NÁZEV TÉMATU:

Propojení sítí IEEE 802.15.4 a Ethernet

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou bezdrátových sítí IEEE 802.15.4 realizujte zařízení propojujícího bezdrátovou síť standardu IEEE 802.15.4 se sítí Ethernet se zaměřením na sběr dat ze sensorických sítí.

DOPORUČENÁ LITERATURA:

Termín zadání: 7.2.2011

Termín odevzdání: 23.5.2011

Vedoucí práce: doc. Ing. Petr Fiedler, Ph.D.

prof. Ing. Pavel Jura, CSc.

Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRACT

This work is devoted to the problem of interconnection of different network types, specifically IEEE 802.15.4 and Ethernet networks. Motivation for implementing such an interconnection arises from increased use of WSNs (Wireless Sensor Networks) penetrating many of today's segments of human activity. Deployment of WSNs stems out of the need of controlling and/or monitoring of environment this network is attached to. This usually implies the existence of some kind of Gateway nodes capable of relaying of measured data from inside of the WSN to the outside world and/or providing configuration information and control commands to the WSN. A Gateway usually accomplishes this by interconnecting the WSN with other types of networks acting as a border element. There are different types of Gateways with different capabilities regarding to the network operation, all dependent on a particular network in use. On the software part the interconnection may be done from Network up to Application layer of the ISO/OSI model. Hardware interfaces Physical and Data-Link layers and of course has to be capable of running interfacing software (which may be rather complex). So there is always balance between the system complexity and sufficient capabilities.

KEYWORDS

WSN, Gateway, Portux920T, PXB, ZigBit, Linux, IEEE802.15.4, Ethernet, ReST, XMPP

ABSTRAKT

Táto práca sa venuje problému prepojovania rôznych typov sietí, konkrétne sietí typu IEEE 802.15.4 a Ethernetu. Motivácia vychádza zo stále sa rozširujúceho využitia bezdrátových senzorických sietí, potreby zberu dát z nich a ich integrácie. To vyžaduje aby sieť obsahovala prvky schopné preniesť dáta z bezdrátovej siete do okolitého sveta a prípadne poskytnúť konfiguračné a riadiace informácie do vnútra siete. Z hľadiska protokolov a programového vybavenia sa prepojenie uskutočňuje na rôznej úrovni, od sieťovej až po aplikačnú vrstvu komunikačného modelu ISO/OSI, s podporou hardvéru na fyzickej a linkovej vrstve.

KLÍČOVÁ SLOVA

WSN, Gateway, Portux920T, PXB, ZigBit, Linux, IEEE802.15.4, Ethernet, ReST, XMPP

PAVLATA, Karel *Interconnection of IEEE 802.15.4 and Ethernet Networks*: master's thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation, 2011. 60 p. Supervised by doc. Ing. Petr Fiedler, Ph.D.

DECLARATION

I declare that I have elaborated my master's thesis on the theme of "Interconnection of IEEE 802.15.4 and Ethernet Networks" independently, under the supervision of the master's thesis supervisor and with the use of technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, concerning the creation of this master's thesis, master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal copyright and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Vol., including the possible consequences of criminal law resulted from Regulation § 152 of Criminal Act No 140/1961 Vol.

Brno

.....

(author's signature)

Experientia docet
Experience is the best teacher

CONTENTS

Introduction	10
1 Network Architecture	11
2 The Description of Networks	13
2.1 Short description of IEEE 802.15.4 (LR-WPAN)	13
2.1.1 The physical layer [1, 2]	13
2.1.2 The MAC layer [1, 2]	14
2.1.3 Higher Layers	16
2.2 Short description of IEEE 802.3 (Ethernet)	18
3 HW design and parts description	19
3.1 WSN node top level design	19
3.1.1 Sensing subsystem	19
3.1.2 Computing and Communication subsystems	20
3.1.3 Power subsystem	20
3.2 Gateway top level design	20
3.3 Portux920T	21
3.4 ZigBit™	24
4 Detailed design description	27
4.1 Trilobite	27
4.2 WSN PIR node	28
4.3 PXB gateway	29
5 Software implementation	31
5.1 GNU/Linux	32
5.1.1 Overview of TTY	32
5.1.2 Low level USART driver modifications	32
5.1.3 Line discipline implementation	33
5.2 BitCloud	35
5.3 JamVM	35
5.3.1 JNI	35
5.4 Higher layer protocols and frameworks	37
5.4.1 ReST [17]	37
5.4.2 XMPP [19]	38
5.5 Demo application	39

6 Conclusion	41
Bibliography	42
List of symbols, physical constants and abbreviations	44
List of appendices	47
A PCBs and schematics	48
A.1 WSN Gateway	48
A.2 WSN node	51
A.3 RF headers (Trilobite)	53
B partlists	56
B.1 WSN Gateway	56
B.2 WSN Node	57
C Libraries and Frameworks	59
D Content of CD	60

LIST OF FIGURES

1.1	Proposed WSN integration architecture	11
2.1	IEEE 802.15.4 topologies [2]	14
2.2	Schematic view of the beacon frame and the PHY packet [2]	15
2.3	Schematic view of the data frame and the PHY packet [2]	15
2.4	Schematic view of the acknowledgment frame and the PHY packet [2]	15
2.5	Schematic view of the MAC command frame and the PHY packet [2]	16
2.6	Schematic view of the ZigBee stack	17
2.7	Ethernet frame [11]	18
3.1	Top level diagram of WSN node, adapted from [16]	19
3.2	Top level diagram of Gateway	21
3.3	Portux920T [12]	22
3.4	Portux920T diagram [12]	23
3.5	ZigBit block diagrams [15]	25
4.1	ZigBit 2.4 GHz Wireless Amplified Module	27
4.2	ZigBit 2.4 GHz Dual Chip Antenna	27
4.3	WSN PIR node	28
4.4	WSN GW	30
5.1	Software architecture of the gateway	31
5.2	TTY core overview [22]	33
5.3	TTY core layers, adapted from [23]	34
A.1	WSN Gateway parts placement	48
A.2	WSN Gateway top side	49
A.3	WSN Gateway bottom side (mirrored)	49
A.4	WSN Gateway schema	50
A.5	WSN PIR node parts placement	51
A.6	WSN PIR node top side	51
A.7	WSN PIR node bottom side (mirrored)	51
A.8	WSN PIR node schema	52
A.9	Trilobite A2 schematic	53
A.10	Trilobite A2 components placement	53
A.11	Trilobite A2 top PCB layer	54
A.12	Trilobite A2 bottom PCB layer	54
A.13	Trilobite AMP schematic	54
A.14	Trilobite AMP components placement	55
A.15	Trilobite AMP top PCB layer	55
A.16	Trilobite AMP bottom PCB layer	55

LIST OF TABLES

3.1	ZigBit modules overview [15]	24
3.2	ZigBit specifications [15]	26
4.1	Parameters of used sensors	28
4.2	PXB GPIO assignment	30

LISTINGS

5.1	termios structure	33
5.2	N_ZBT tty_ldisc structure	34
5.3	JNI interface	36
5.4	JNI library example	36
5.5	Data parsing function	39
5.6	Data in the JSON format	39
5.7	USART transmission handling function	40
B.1	Partlist for the Gateway	56
B.2	Partlist for the WSN node	57

INTRODUCTION

Wireless Sensor Networks (WSN) use networked, resource constrained embedded devices to interact with its environment. Although there is possibility of self-contained deployment, usually there is a need for input (commands) and/or output (data) interactions with WSN depending on particular application scenario. Examples are Data Collection and Actuator Control services and also Service and Network Discovery. These interactions can be divided into three large classes: random access interactions (request-response model), continuous monitoring (periodic stream of data) and event-based interactions (sporadic events) which in turn can be further divided to human-machine and machine-to-machine interactions (client can be either human or computer). Interactions should be managed in a consistent way regardless of the type of a client. Although the lack of open and simple standards in this area makes it difficult, there is a proliferation of the use of technologies based on ReST¹ architectural design, which is a style of software architecture for distributed hypermedia systems (such as WWW)[20].

An important aspect of the WSN is incorporation of techniques contributing to a decrease of power consumption and thus an increase in the life span of the network (devices are usually battery powered). Using techniques like low duty cycle operation, impose significant delays to the communication that must be dealt with, in order to ensure smooth operation of the network from client's point of view. Clients have to be abstracted from peculiarities of particular WSN architecture in use. This, and the facts that many WSN platforms use devices too constrained in resources, implies that gateways are a vital part of WSN deployments. IP-based and HTTP-enabled devices are being developed and emerging, in particular 6LowPAN² is gaining increasing popularity. However, it is not always feasible to use this technology due to restrictions in resources. Nevertheless, these kinds of devices can be supported transparently as there is also the need for network interfacing at the lower layers.

In first part of this thesis the general concept of a WSN network integration is described, followed by the brief description of networks.

In the next part, the general design of hardware and it's parts is described followed by a more detailed description.

The proposed software stack and implementation is described in the last chapter together with the implemented example application.

¹ReST - Representational State Transfer

²6LowPAN - IPv6 over Low power Wireless Personal Area Networks

1 NETWORK ARCHITECTURE

For successful integration of WSN into the Internet and the Web we need to take into account characteristics of different interactions with WSN and model them, preferably using some standard protocol. This allows us to hide peculiarities of particular (perhaps proprietary) WSN protocol and export generic interface to users, thus shield them to the maximum possible extent from the underlying WSN technology which might change or could be even heterogeneous consisting of different WSN network segments.

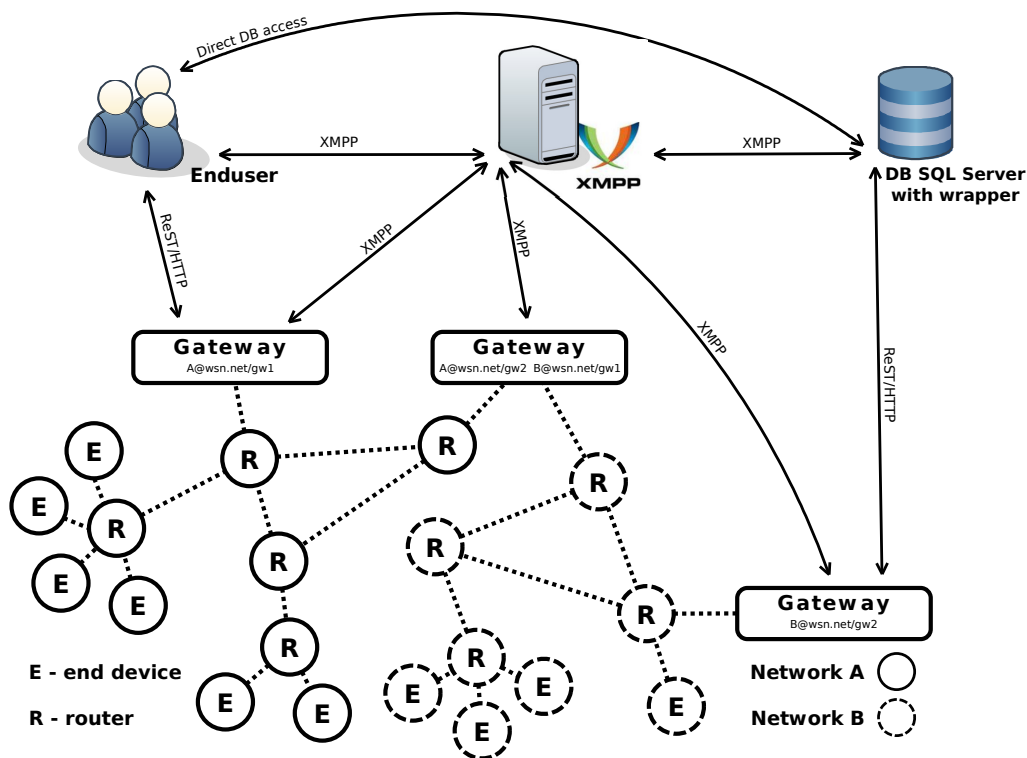


Fig. 1.1: Proposed WSN integration architecture

The underlying WSN deployment may consist of several distinct technologies be it at the level of protocol or even different physical layers as demonstrated in Fig. 1.1. This forms separate WSN layer connected to the gateways, which possess internal representation of devices forming the particular network. It exports interface to the users consistently regardless of a WSN in use or type of a client. In figure there are depicted two types of clients representing humans and machines. One, being true human client, accessing the data directly using for instance web browser, the other one, being database agent, collecting selected data. The important thing is that they share common interface. The gateway maps resources of the WSN in a ReSTful way, independently of the underlying

WSN, using URIs¹ and stateless HTTP requests. All four CRUD (Create/Read/Update/Delete) operations are carried out using methods of the HTTP protocol: GET, POST, PUT and DELETE. As the interaction is stateless, where state is actually part of the resource's URI itself, there is no need for per-client state information, and the server is much less complex. The example of resource identification is: `http://gateway/network/node23/resources/temperature`. The gateway also publishes list of connected nodes forming the network and it's available resources, so user can navigate to desired information. Interaction with different types of clients can be handled transparently using the content negotiation mechanism of HTTP where human clients might request a different type of answer (HTML) than machines (JSON², XML) based on MIME types. Caching of HTTP requests may also be employed. This preserves the bandwidth and may eliminate duplicated request to the WSN, thus conserving energy of battery powered nodes. Each particular resource (e.g. temperature sensor value) is decoupled with a time-frame of validity and represented within the gateway dependent on the underlying WSN. The node might be sleeping for extended period of time and not be accessible. With this representation, gateway can answer user's requests on behalf of that node with the latest valid value supplying also it's expiration. HTTP caching mechanism can use this value to cache responses in the network so other users get their responses quickly. This is particularly suited for the request-response class of interactions, where users occasionally request some data.

For continuous monitoring (not to mention sporadic events) constant polling for new data is not suitable as it poses high load on the network. To alleviate this problem, and because of the inherent asynchrony of WSN from the client's request, the push-based approach using publish-subscribe model can be used. In contrast to the pull-based approach mentioned above, push technology utilizes server initiated communication. This way users register for particular events they are interested in and they are notified whenever such an event occurs. XMPP (Extensible Messaging and Presence Protocol) is well established technology employing decentralized client-server architecture and long-lived TCP connections to deliver messages. Especially it's Extension Protocols *XEP-0060: Publish-Subscribe* and *XEP-0163: Personal Eventing Protocol* are useful for building highly distributed event-driven applications. Interactions of individual entities are shown in Fig. 1.1. In terms of XMPP, the whole WSN appear as user with one unique JID³, whereas individual gateways represent resources. In this way user gets notified regardless of which gateway receives actual data form the WSNs.

¹URI - Uniform Resource Identifier

²JSON - Java Script Object Notation

³JID - Jabber ID

2 THE DESCRIPTION OF NETWORKS

2.1 Short description of IEEE 802.15.4 (LR-WPAN)

IEEE standard 802.15.4 specifies the physical layer and part of the data-link layer (MAC - Media Access Control) of the ISO/OSI model for low-cost, low-speed, minimal-power Low-rate Wireless Personal Area Networks (LR-WPANs). Possible network topologies are shown in Fig. 2.1 which include star topology and peer-to-peer topology. Additional higher layers protocols can add support for routing and multi-hop communications in the form of mesh or (cluster-)tree topologies. Devices either use 64 bit long IEEE address or short 16 bit address assigned during association process. Network can operate either in Beaconing mode with slot reservation or Non-Beaconing (unslotted) mode. It operates in one of the following frequency bands using various modulation and spreading techniques:

- 868.0-868.6 MHz: Europe, allows one communication channel (2003, 2006)
- 902-928 MHz: North America, up to ten channels (2003), extended to thirty (2006)
- 2400-2483.5 MHz: worldwide use, up to sixteen channels (2003, 2006)
- There are several amendments specifying new frequency bands and modulation techniques such as UWB. See [3, 4, 5]

The standard defines four frame structures (beacon frame Fig. 2.2, data frame Fig. 2.3, acknowledgement frame Fig. 2.4 and MAC command frame Fig. 2.5) and specifies following items in detail:

- Device type (PAN Coordinator/FFD¹/RFD²)
- Frame structure
- Superframe structure
- Data transfer model (to/from coordinator, peer-to-peer)
- Robustness
- Energy saving considerations
- Security

2.1.1 The physical layer [1, 2]

The physical layer defines set of constants such as maximum PSDU³ packet size (127) and provides data transmission service and interface to the physical channel. It defines two SAPs⁴ (PD-SAP, PLME-SAP) and provides following services to the

¹FFD - Full-Function Device

²RFD - Reduced-Function Device

³PSDU - Phy Service Data Unit

⁴SAP - Service Access Point

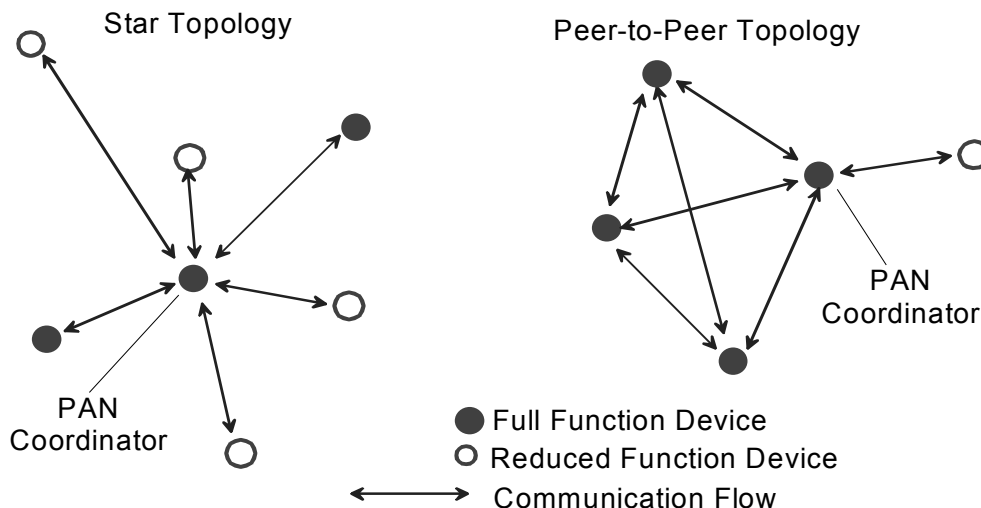


Fig. 2.1: IEEE 802.15.4 topologies [2]

above layer using those SAPs:

- Management of the physical RF transceiver
- Energy Detection (ED)
- Link Quality Indication (LQI)
- Clear Channel Assesment (CCA)
- Channel Frequency Selection
- Data transmission and reception

2.1.2 The MAC layer [1, 2]

The medium access control (MAC) allows the transmission of MAC frames through the use of the physical channel. Besides the data service, it offers a management interface and itself manages access to the physical channel and network beaconing. It also controls frame validation, guarantees time slots and handles node associations.

Overview of MAC layer responsibilities:

- Beacon generation
- Beacon synchronization
- PAN association and disassociation support
- Utilization of the CSMA/CA channel access mechanism
- Control and maintenance of GTS ⁵
- Provision of reliable connection by the means of retransmission, ACK ⁶ and CRC ⁷

⁵GTS - Guaranteed Time Slot

⁶ACK - Acknowledgement

⁷CRC - Cyclic Redundancy Check

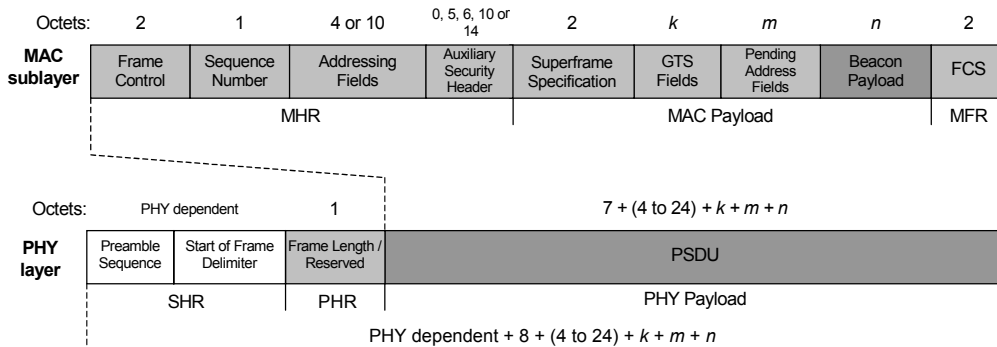


Fig. 2.2: Schematic view of the beacon frame and the PHY packet [2]

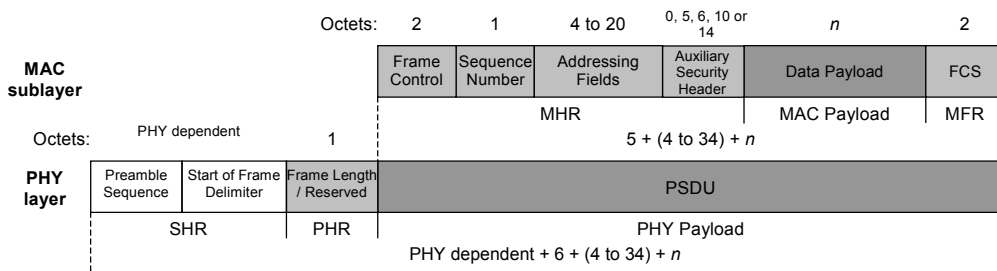


Fig. 2.3: Schematic view of the data frame and the PHY packet [2]

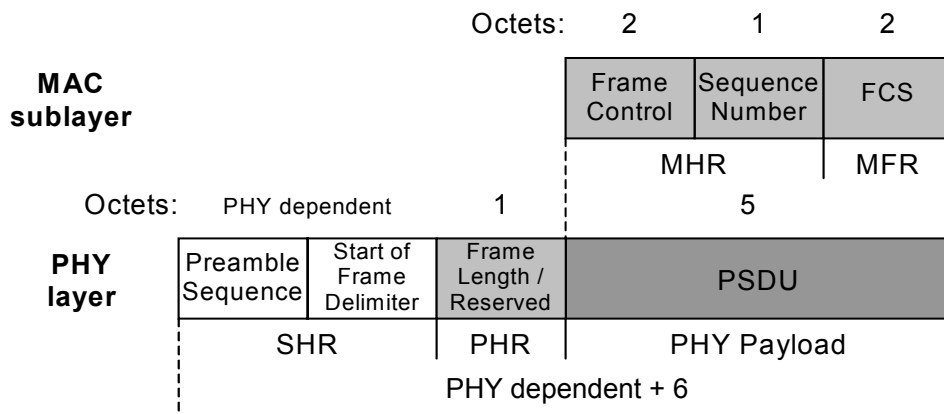


Fig. 2.4: Schematic view of the acknowledgment frame and the PHY packet [2]

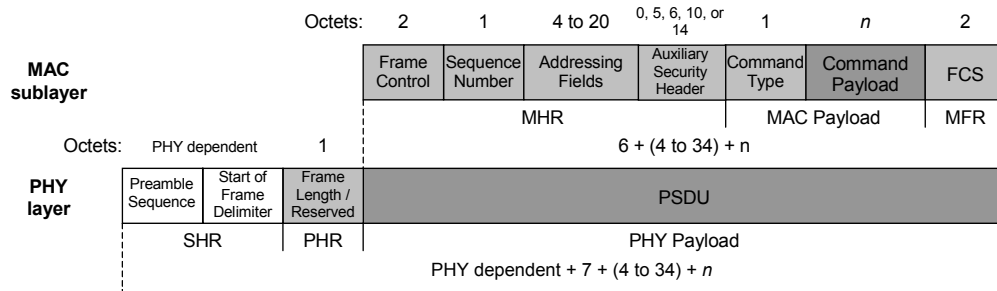


Fig. 2.5: Schematic view of the MAC command frame and the PHY packet [2]

More information can be found in [1, 2, 3, 4, 5]

2.1.3 Higher Layers

The standard does not define higher layers (the network layer and above) but instead rely on other standards and specifications which build upon it, such as ZigBee and 6LoWPAN. Short description of the aforementioned follows.

ZigBee [6]

ZigBee is proprietary standard for low-cost, low-power wireless mesh networking and is maintained by the ZigBee Alliance. It builds upon IEEE802.14.4 LR-WPAN standard and defines the network layer, security provider and application profiles for following uses:

- Home Automation
- Home Entertainment and Control
- Automated Meter Reading
- Asset Tracking
- Building Automation
- Industrial Control
- Personal, Home and Hospital Care
- Toys
- RF4CE (Radio Frequency for Consumer Electronics)

Schematic view of the ZigBee stack is shown in Fig. 2.6. ZigBee defines three types of devices:

- ZigBee coordinator (ZC)
- ZigBee Router (ZR)
- ZigBee End Device (ZED)

Capabilities of these devices more-less correspond to those defined by IEEE802.15.4. ZigBee uses AODV (Ad-hoc On-demand Distance Vector) routing protocol to automatically construct path between nodes. Current ZigBee profiles should support non-beaconed as well as beaconed modes of operation. More on this topic can be found in [6, 7, 8].

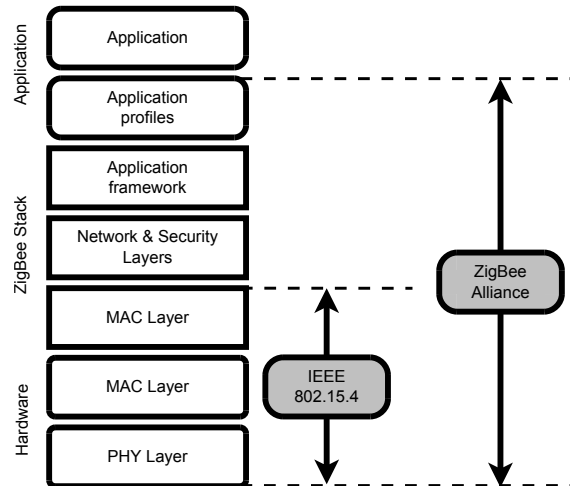


Fig. 2.6: Schematic view of the ZigBee stack

6LoWPAN [9][10]

6LoWPAN acronym means IPv6 over Low power Wireless Personal Area Networks and it is the name of a working group in the IETF (Internet Engineering Task Force). The group has defined the adaptation layer (consisting of encapsulation and header compression mechanisms) that allows IPv6 packets to be sent over IEEE 802.15.4 based networks.

Through deep understanding of the interaction between IEEE 802.15.4, IPv6 and UDP, 6LoWPAN removes fields which are redundant among those headers, thereby reducing the size of the packets being transmitted over the air. 6LoWPAN removes a number of fields in the IPv6 and UDP headers because they take well-known values, or because they can be inferred from fields in the IEEE802.15.4 header.

2.2 Short description of IEEE 802.3 (Ethernet)

Ethernet is a large family of frame-based computer networking technologies designated for LANs. It defines varieties of Physical layers and MAC and LLC sub-layers of the Data Link layer of the ISO/OSI model. It comes in various speed and physical media types ranging from 10 Mbit/s with distance limit of 100 meters over the twisted pair up to 100 Gbit/s with distance limit of 100 km over the optical fiber. New specifications with higher speeds are under development. The most common of the Ethernet types (this is particularly true for embedded systems) is 100 Mbit/s, so called Fast Ethernet, particularly it's 100BASE-TX variant which runs over two twisted wire-pairs inside a category 5 or above UTP cable. Each network segment can have a maximum distance of 100 meters. In it's typical configuration it provides full-duplex operation with throughput of 100Mbit/s in both directions. It is reverse compatible with newer Gigabit Ethernet through procedure called autonegotiation, where both transmitters agree on common capabilities. The Fig. 2.7 shows the Ethernet frame structure. Particular fields have following lengths:

- Preamble: 7 octets of 10101010
- SFD: 1 octet of 10101011
- Destination: 6 octets
- Source: 6 octets
- EtherType: 2 octets
- Payload: 46-1500 octets
- FCS (32-bit CRC): 4 octets

Ethernet is the de facto standard of the Internet connection, with almost all of the Internet end users connected using this technology. The dominant protocol suite used with Ethernet (and Internet in general) is TCP/IP, residing on the Network and Transport layers of the ISO/OSI model and containing such protocols as IP, TCP and UDP. In mature systems reception and transmission is commonly handled by the Operating system so the user utilize the application interface the particular system exports to access the network. This is commonly done through *sockets*. More about Ethernet can be found in [11].

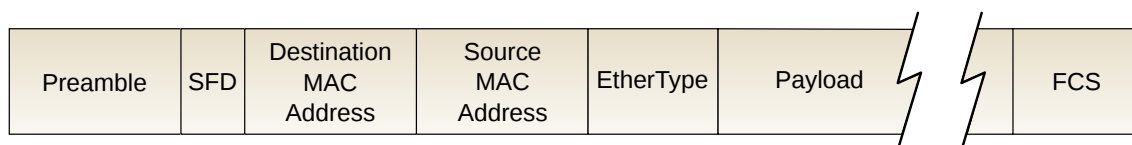


Fig. 2.7: Ethernet frame [11]

3 HW DESIGN AND PARTS DESCRIPTION

Within scope of this thesis common platform for WSN was developed together with hardware of WSN nodes and the Gateway. Top level designs are described next, followed by part description. Detailed implementation is described in the next chapter.

3.1 WSN node top level design

WSN node top level designed can be divided into following parts:

- Sensing subsystem
- Computing subsystem
- Communication subsystem
- Power subsystem

The diagram is shown in Fig. 3.1

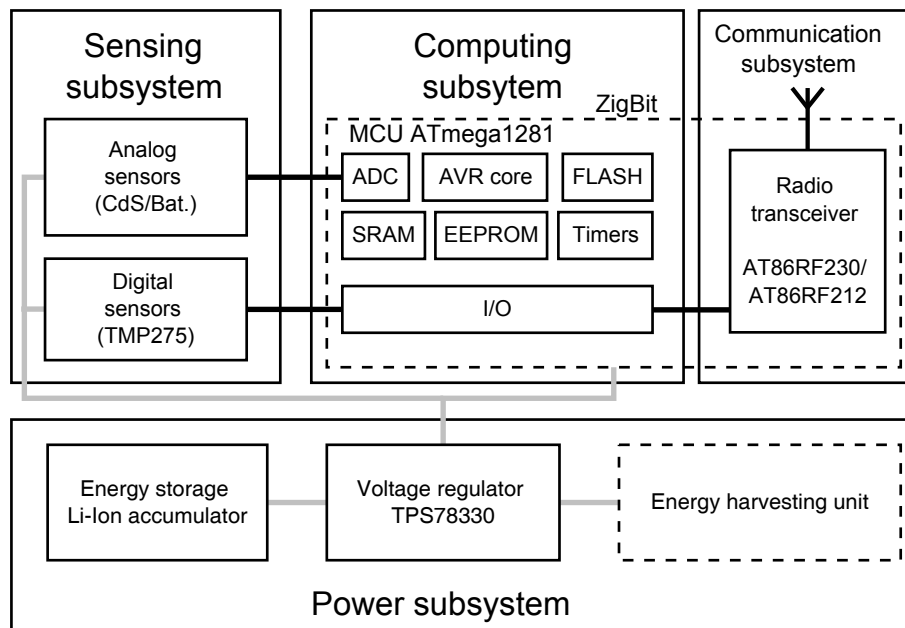


Fig. 3.1: Top level diagram of WSN node, adapted from [16]

3.1.1 Sensing subsystem

The sensing subsystem of this particular node consists of analog sensors such as CdS¹ light intensity sensor and battery voltage level connected to the ADC and digital sensors such as TMP275 temperature sensor connected using TWI interface and PIR connected using GPIO. CdS and PIR sensors are part of MS-360LP motion

¹CdS - Cadmium Sulfide cell

detection sensor. Sensors were selected with capabilities of low supply voltage and low power operation in mind.

3.1.2 Computing and Communication subsystems

The computing subsystem together with the communication subsystem are realized using ZigBit modules described in sections 3.4. Use of this modules promotes modularity of the design and allows for quick change of the frequency band.

3.1.3 Power subsystem

The power subsystem comprises Lithium-Ion chemistry accumulator and TPS78330 LDO linear voltage regulator, which is characterized by very low quiescent current. There is possibility of attaching energy harvesting unit which scavenges available energy from it's environment.

3.2 Gateway top level design

Top level diagram is shown in Fig. 3.2. The system basically consists of two parts:

- Portux920T SBC ²
- PXB³ extension board equipped with 2 ZigBitTM modules

Use of SBC capable of running Linux operating system with "standardized" hardware extension interface is favorable from software as well as hardware point of view. It allows to separate both sides and use complex board repeatedly if there is any need for modification. Usage of Linux adds the ease of software developing and debugging as well as flexibility, together with mature TCP/IP stack implementation.

Parts are interconnected using USARTs⁴ available on PXB interface. Usage of synchronous interface adds robustness and reliability to the interconnection and allows for higher communication speed without worrying about baud rate mismatch in case of ZigBit module's oscillator instability. PXB makes use of DIN41612 connectors, so that extension board can be connected to a regular computer using this interface and an auxiliary board for software development purposes. Individual components are described in later sections.

²SBC - Single Board Computer

³PXB - Portux eXtension Board

⁴USART - Universal Synchronous-Asynchronous Receiver/Transmitter

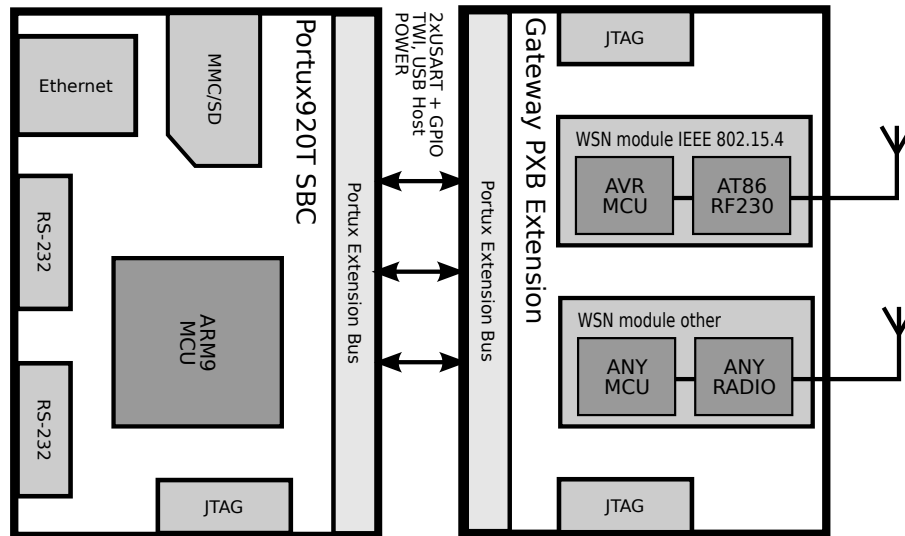


Fig. 3.2: Top level diagram of Gateway

3.3 Portux920T

Portux920T is Single Board Computer equipped with AT91RM9200 ARM920T core CPU from german vendor Taskit. Diagram is shown in Fig. 3.4 and Fig. 3.3 shows actual appearance of Portux920T Eurocard version. Basic features of Portux920T:

- SBC with AT91RM9200 CPU
- half size euroboard
- Linux open source operating system
- flexible Portux Extension Bus provides modularity

Technical details[13]:

CPU

- Atmel® AT91RM9200 with ARM920T core
- ARM9TDMI instruction set
- 200 MIPS at 180 MHz
- 16/16 kB data/instruction cache
- Memory Management Unit (MMU)
- External Bus Interface (EBI)

Memory

- 64 MB SDRAM
- 16 MB Flash
- SD/MMC card slot onboard
- Optional CompactFlash card via Portux Extension Bus

Peripherals (onboard)

- 2 serial interfaces, USART 0 and USART 1

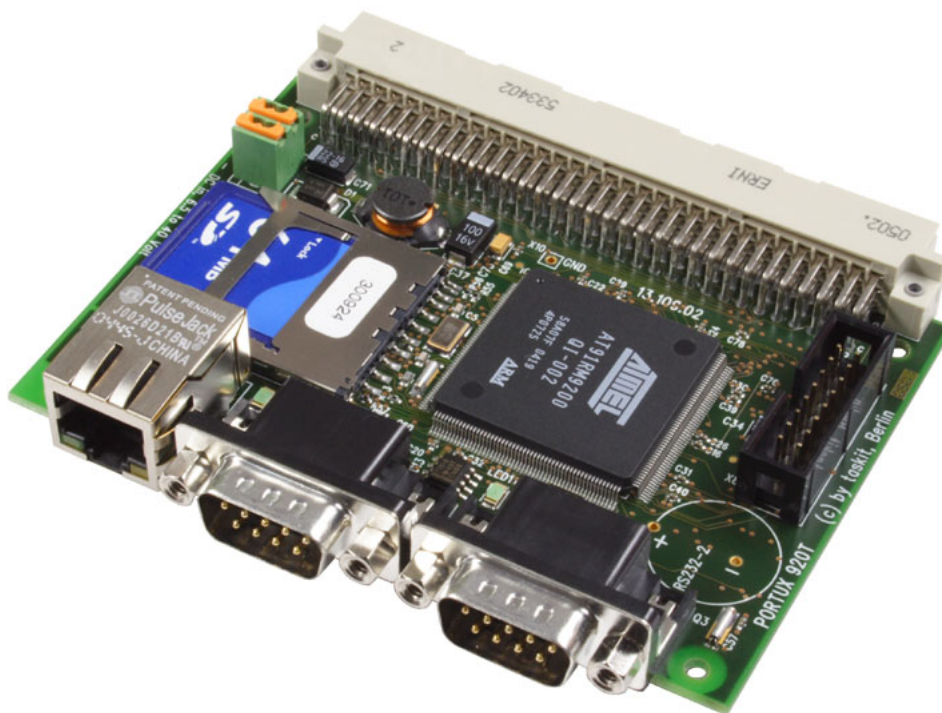


Fig. 3.3: Portux920T [12]

- Debug unit as an alternative to the first serial interface
- MAC 10/100 Mbit/s ethernet
- JTAG
- Portux Extension Bus via 96-pin connector
- Via Portux Extension Bus additionally required modules can be connected

Peripherals (Portux Extension Bus)

- Intergrated components of the microcontroller are accessible via Portux Extension Bus
- 2 serial interfaces, USART 2 and 3
- TWI (Two Wire Interface)
- SPI (Serial Peripheral Interface)
- PIF bus, universal easy programmable 8-bit bus with 64 I/O addresses
- USB host port
- USB client port
- 32 single programmable I/O ports multiplexed with integrated components (chipselect for EBI, USART 2, USART 3, SPI, TWI)

Operating system

- Embedded Linux, kernel version 2.6
- "U-Boot" bootloader and monitor

- Journaling Flash file system (JFFS)
- Compiled toolchain, binaries and source code available

Power management

- Supply voltage: 6.5 - 24 V
- Operating voltage: 3.3 V
- Power consumption: 70 mA at 10 V (normal operation)

Eurocard version

- Equipped with 96-pin connector angled (PXB)
- Equipped with standard connector plugs for USART 0 and USART 1 (DSUB-9) and ethernet (RJ-45)
- Dimensions: 100 mm x 71 mm x 16 mm (half Eurocard)
- Format corresponds to a full Eurocard with an extension board of the same size, enabling installation in standard housings

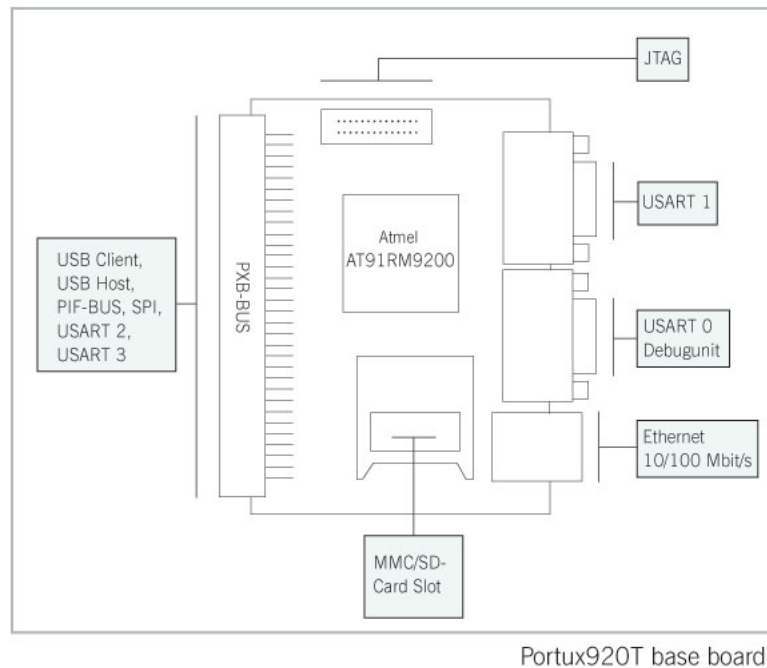
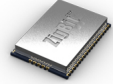



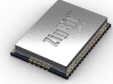


Fig. 3.4: Portux920T diagram [12]

3.4 ZigBit™

ZigBit™ is the name for range of ultra-compact 802.15.4/ZigBee OEM modules from Atmel intended for wireless networking applications. It has integrated ATmega1281, the 8 bit AVR MCU, and RF transceiver and features ease of integration, ultra-low power consumption and superior radio performance. There exist various models, summary of which is listed in Tab. 3.1.

Model	Image
ZigBit 2.4 GHz Wireless Modules	
Balanced Output	
Dual Chip Antenna	
ZigBit 2.4 GHz Wireless Amplified Modules	
Amp UFL-connector	
Amp Un-balanced Output	
ZigBit 700/800/900 MHz Wireless Module	
Balanced Output	

Tab. 3.1: ZigBit modules overview [15]

Figure 3.5 shows block diagram connection for particular ZigBit™ modules. Modules operating on 2.4 GHz make use of the AT86RF230 transceiver and the ones operating on 700/800/900 MHz makes use of the AT86RF212 transceiver.

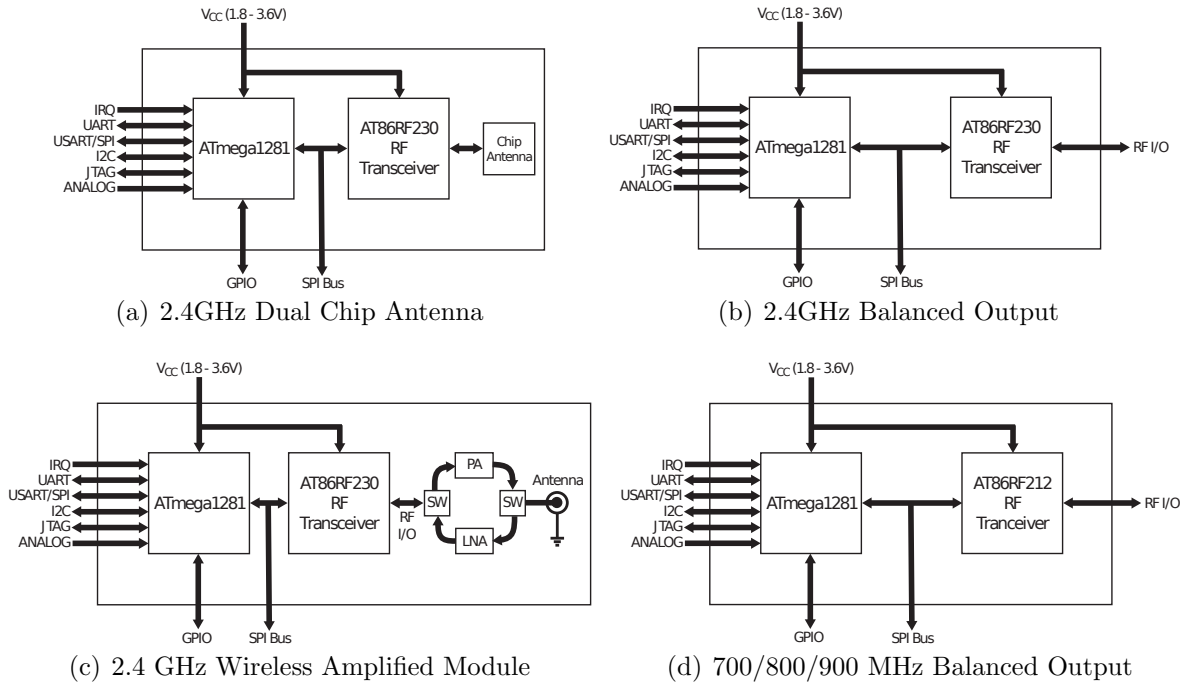


Fig. 3.5: ZigBit block diagrams [15]

Parameters of ZigBit Module Hardware Platforms are shown in Table 3.2.

Supported External Interfaces [15]:

- USART/SPI, I2C, 1-wire
- UART with CTS/RTS control
- JTAG
- 9 spare GPIOs (up to 25 GPIOs total)
- 2 spare IRQ lines
- 4 ADC lines

Parameter	ZigBit900	ZigBit 2.4	ZigBit 2.4 AMP
Frequency band	EU ISM 863 - 870 MHz AM ISM 902 - 928 MHz	2.400 - 2.483 GHz	2.400 - 2.483 GHz
Hardware data encryption	AES 128bit	?	?
Data rate	up to 1 MBit/s	250 kBit/s	250 kBit/s
Max output power	up to +11 dBm	+3 dBm	+20 dBm
Receiver Sensitivity (PER 1%)	up to - 110 dBm	- 101 dBm	- 104 dBm
Supply Voltage (V_{CC})	1.8 V to 3.6 V	1.8 V to 3.6 V	3.0 V to 3.6 V
Current Consumption RX/TX	11 mA / 26 mA	19 mA / 18 mA	23 mA / 50 mA
Current Consumption Power Save	<6 μ A	<6 μ A	<6 μ A
On-Chip Flash Memory Size	128 kBytes	128 kBytes	128 kBytes
On-Chip RAM Size	8 kBytes	8 kBytes	8 kBytes
On-Chip EEPROM Size	4 kBytes	4 kBytes	4 kBytes
Size	18.8 x 13.5 x 2.8 mm	?	?
Weight	1.3 g	?	?
Operating Temperature	-40 — +85 °C	-40 — +85 °C	-40 — +85 °C

Tab. 3.2: ZigBit specifications [15]

4 DETAILED DESIGN DESCRIPTION

4.1 Trilobite

To emphasize modular design and create common RF platform for gateway and sensor nodes, the Trilobite modules have been created. It's basically PCB¹ containing ZigBit™ modules with necessary circuitry and B2B² connector. It allows to freely interchange between different ZigBit™ modules in one design according to application's needs and separates power and sensor part from RF part. In this way it's easy to change frequency bands or event entire wireless technology with rest of the application left intact.

2.4GHz amplified module and 2.4GHz module with dual chip antenna are shown in Fig. 4.1 and Fig. 4.2 respectively.



Fig. 4.1: ZigBit 2.4 GHz Wireless Amplified Module

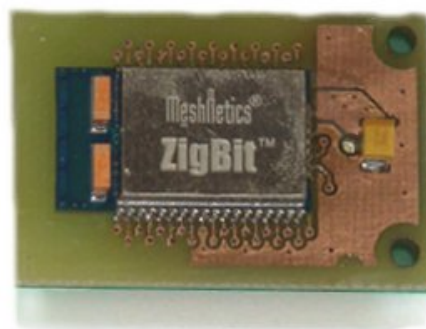


Fig. 4.2: ZigBit 2.4 GHz Dual Chip Antenna

Schematic diagrams and layouts are included in Appendix A.3.

¹PCB - Printed Circuit Board

²B2B - Board-to-Board

4.2 WSN PIR node

PIR node is one of many sensor nodes realized to form WSN within premises of Brno University of Technology. This particular node is equipped with the Passive Infrared sensor to enable motion detection, the CdS light intensity sensor and the digital temperature sensor. The picture of node is shown in Fig. 4.3 and schematic diagrams and layouts are included in Appendix A.2.



Fig. 4.3: WSN PIR node

The node is powered by Lithium-Ion chemistry accumulator with fully-charged voltage above the maximum allowable supply voltage of ZigBit module. To cope with this problem TPS78330 LDO linear voltage regulator with quiescent current of 500 nA was used. It regulates output voltage to the value of 3 V (3.3 V for TPS780330220) during the time when input voltage is above it's regulation point and then scales with decreasing voltage. To prevent excessive discharge, to which Li-Ion chemistry is particularly sensitive, shut-down circuitry is employed.

Sensor parameters are summarized in Tab. 4.1. TMP275 is a digital temperature

Sensor	Phenomenon	Producer	V_{cc} [V]	$I_{standby}$ [μ A]	Interface
TMP275	temperature	TI	2.7 - 5.5	0.1	I^2C
MS-360LP	PIR motion light intensity	IR-TEC	3 - 4	10	binary analog

Tab. 4.1: Parameters of used sensors

sensor with accuracy of ± 0.5 °C over the operating range from -20 °C to +100 °C and resolution of 0.0625 °C. It's wide supply range and low power consumption makes it ideal for battery powered applications. The sensor is connected to the ZigBit module using I^2C interface and it's alarm output is tied to the IRQ pin capable

of waking the MCU from deep sleep mode in case the programmed temperature threshold is exceeded. Since power consumption of the sensor in standby mode is negligible comparing to ZigBit module itself, it is powered directly from main power supply. The measurement itself is done in one-shot mode after which the sensor automatically switch itself to standby mode to save energy. The next measurement cycle is started on demand from the MCU.

IR-TEC's MS-360LP is a low power Passive Infrared motion sensor module with integrated CdS light intensity sensor. The motion detection output is in a form of TTL open collector with externally applied pull-up resistor. It is tied to the pin change IRQ of the ZigBit module to allow wake up from deep sleep when main oscillator of MCU is not running. Light intensity output forms a voltage divider, the output of which is fed to the ADC pin. The divider is switched using TS5A23166 TI's dual SPST analog switch to stop the current flowing through the divider at the time when there is no ongoing measurement and thus reduce the power consumption. The battery voltage measurement is done in similar fashion. Supply voltage for PIR sensor is switchable using TI's TS5A3159 SPDT analog switch. This way, the power consumption can be reduced even further in times when motion detection is not required.

Connector with USART and JTAG interface is available on the side of the PCB for debugging purposes and eventual firmware upgrades.

4.3 PXB gateway

Presented gateway extension board was developed as an universal platform capable of carrying two WSN radio modules which are attached through board-to-board connectors. It's intended to run 900 MHz and 2.4 GHz Trilobite modules in parallel (preferably using modules with external antennas) to interconnect and collect data from both types of networks, but any kind of WSN modules can be attached provided the hardware interfacing is done right. The interconnection of modules is done using synchronous mode of USART interface to increase reliability and communication speed. There are two USART interfaces available at the PXB port, both of which are used. If, for any reason in the future the modules are being replaced with types using other interface, e.g. SPI, Linux either support GPIO based SPI or modification of extension board can be done without necessity to rebuild complex hardware of the SBC.

Gateway extension board is equipped with the same temperature sensor as the aforementioned WSN node to give it some basic sensing capability. It further contain USB host interface allowing to attach various peripherals, such as USB Wi-Fi

cards, bluetooth dongles, etc. Each module has four indication LEDs attached to it and DIP switch for eventual manual configuration. Also JTAG interface and reset circuitry for facilitating of firmware development is present. The power part takes care of powering USB interface. The gateway also features three indication LEDs connected to the GPIO pins of Portux.

CPU	I/O line	PXB	Function	CPU	I/O line	PXB	Function
123	PB28	A13	FI	17	PC0	C14	LED1
125	HDMA	A14	USB HD	18	PC1	C15	LED2
126	HDP A	A15	USB HD	122	PB27	A21	LED3
71	PA25	A17	TWD	86	PB6	C28	RST2(2)
72	PA26	C17	TWCK	87	PB7	C30	RST1(1)
68	PA22	C24	RXD2(2)	47	PA5	A30	TXD3(1)
69	PA23	A25	TXD2(2)	48	PA6	C29	RXD3(1)
70	PA24	C23	SCK2(2)	82	PB2	C25	SCK3(1)

Tab. 4.2: PXB GPIO assignment

The picture of the PXB gateway is shown in Fig. 4.4 and schematic diagrams and layouts are included in Appendix A.1.

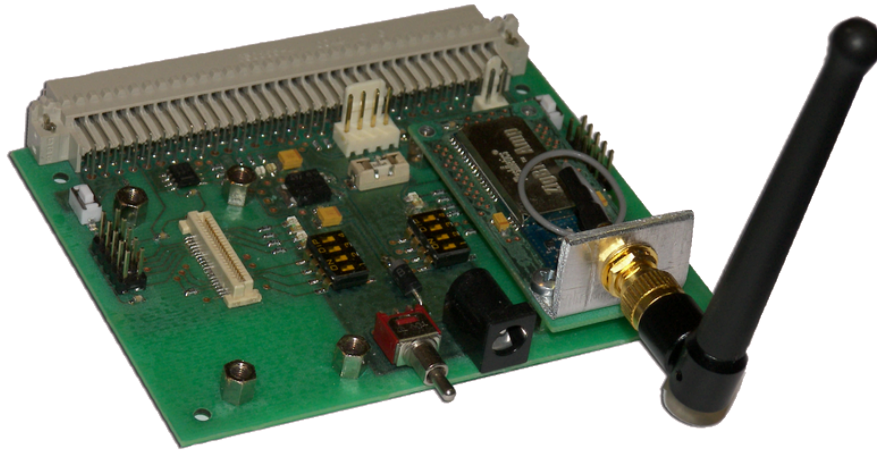


Fig. 4.4: WSN GW

5 SOFTWARE IMPLEMENTATION

Software architecture of the gateway is using layered structure depicted in Fig. 5.1. With the bottom layer representing physical devices and their eventual firmware, GNU/Linux operating system is placed above and enables interaction with these devices on standard and higher level manner. The use of complex operating system such as Linux is justified by the flexibility and the portability of the final solution at relatively small cost in terms of hardware performance. Linux has a large community of users and developers, is entirely open source and scales down well for embedded systems. There are many applications already available for Linux which can be used to ease the development. Java virtual machine resides on top of Linux OS. The same reasoning applies as for use of Linux OS itself. Java is wide spread programming language, particularly suitable for networked and multi-threaded applications, flexible and portable. With plenty of available frameworks and libraries, it allows for rapid development with no need for cross-compilation for particular hardware once JVM¹ is running. Virtual machine used here, JamVM is highly optimized and suitable for embedded systems.

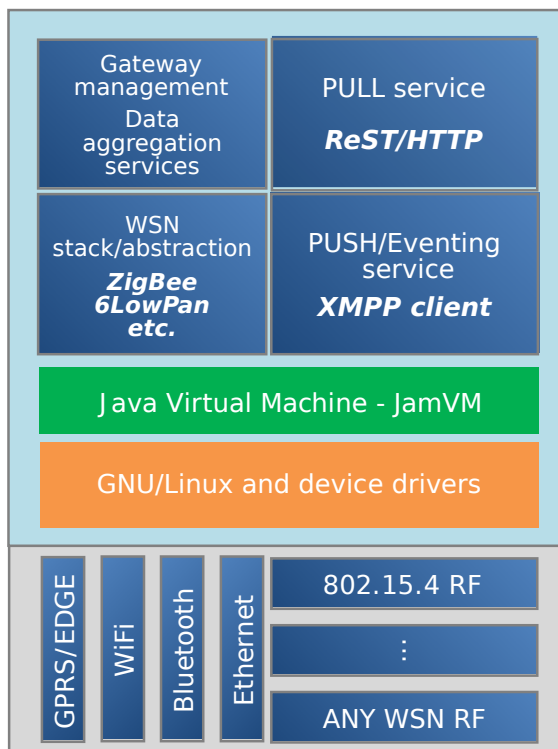


Fig. 5.1: Software architecture of the gateway

The main part of this thesis deals with lower parts of the software stack, en-

¹JVM - Java Virtual Machine

abling easy and smooth integration of wireless modules within Linux and to allow applications (written in Java in particular) to be platform independent. Description of individual layers follows next, ending with overview of frameworks suitable for implementing the network architecture described in Chap. 1.

5.1 GNU/Linux

The Linux kernel version used is `linux-2.6.22` with modifications from taskit GmbH to incorporate specifics of the Portux920T embedded SBC. It has been cross compiled for ARM architecture using GCC toolchain. Portux uses U-Boot bootloader to load the kernel either from on board flash memory or through the network using TFTP² protocol. The bootloader is configured to provide Linux kernel with arguments during boot-time, such as where to find and mount root filesystem, partitioning of on-board flash, size of RAM and which serial port to use as a console device. SD card is used as storage space with ext-2 type filesystem, which provide enough space to store all the necessary applications and libraries and also allows for easy upgrades. Another custom modifications done to the Linux kernel to facilitate further development are explained in subsequent sections.

5.1.1 Overview of TTY

Fig. 5.2 shows layered structure of TTY management. Serial interfaces are part of the TTY from historical reason when remote terminals were used, attached using RS-232 interface.

To allow for greater flexibility and reuse, tty handling is made up of several building blocks. Low level drivers deal with the underlying serial hardware, hiding it's peculiarities on different platforms together with TTY driver. Line disciplines apply policies to the data according to the specific application. TTY core manages all the interconnections and provides core APIs. Flow of data is depicted in Fig. 5.3.

5.1.2 Low level USART driver modifications

Since there is no inherent support for synchronous mode of communication in Linux, there had to be done some modifications to utilize the capability of USART hardware present on Portux to work in synchronous mode. `termios` structure (Lst. 5.1) is used to communicate desired parameters, such as communication speed, parity, etc. to the low level driver. Flag requesting synchronous mode of communication has been added to the `c_cflag` field of structure and `set_termios` function of

²TFTP - Trivial File Transfer Protocol

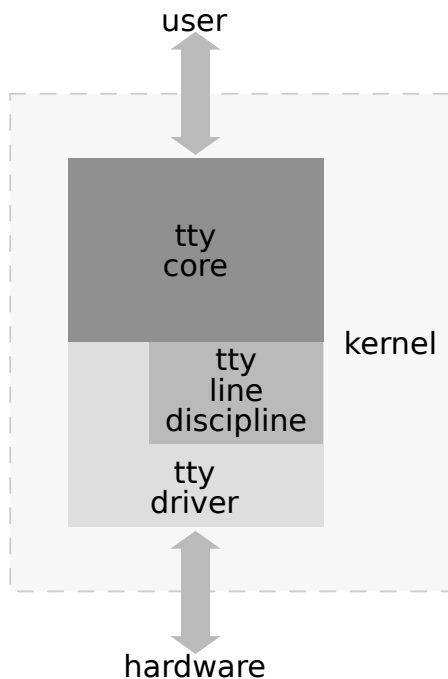


Fig. 5.2: TTY core overview [22]

Atmels's USART low level driver has been modified to incorporate necessary actions in form of register setting etc. In this way we are able to use more robust and reliable USART communication. This setting is automatically applied upon setting our custom line discipline (described in the next section) to the particular serial port. As the termios settings are preserved across openings of the serial port we change the setting back to the standard asynchronous mode after closing the port.

Listing 5.1: termios structure

```

1 struct termios {
2     tcflag_t c_iflag;           /* input mode flags */
3     tcflag_t c_oflag;           /* output mode flags */
4     tcflag_t c_cflag;           /* control mode flags */
5     tcflag_t c_lflag;           /* local mode flags */
6     cc_t c_line;                /* line discipline */
7     cc_t c_cc[NCCS];           /* control characters */
8 };

```

5.1.3 Line discipline implementation

Line disciplines provide an elegant mechanism to use the same serial driver to run different technologies. The low-level physical driver and the tty driver handle the transfer of data to and from the hardware, while line disciplines are responsible for processing the data and transferring it between kernel space and user space [23].

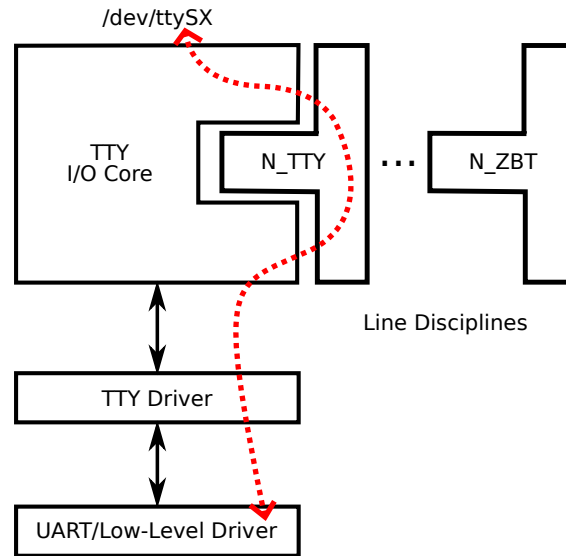


Fig. 5.3: TTY core layers, adapted from [23]

Linux strictly separates between the user space and the kernel space. As we need to interact with the hardware, it is much more comfortable to do so from the kernel side. We can directly access to and configure the GPIO pins needed to interact with the ZigBit modules and also process the data into frames to facilitate programming of higher layers. `tty_ldisc` structure (Lst. 5.2) is used to represent particular line discipline and is registered using `tty_register_ldisc()` function upon loading the kernel module containing the line discipline's functionality.

Listing 5.2: N_ZBT `tty_ldisc` structure

```

1 static struct tty_ldisc n_zbt_ldisc = {
2     .magic      = TTY_LDISC_MAGIC,
3     .name       = "n_zbt",
4     .flags      = 0,
5     /* routines called from above */
6     .open       = n_zbt_open,
7     .close      = n_zbt_close,
8     .read       = n_zbt_read,
9     .write      = n_zbt_write,
10    .ioctl      = n_zbt_ioctl,
11    .poll       = n_zbt_poll,
12    /* routines called from below */
13    .receive_buf = n_zbt_receive_buf,
14    .write_wakeup = n_zbt_write_wakeup,
15    .owner      = THIS_MODULE,
16 };

```

N_ZBT line discipline provides compact interface to control the ZigBit device

and wrap the data to form frames with simple FCS³. The synchronization to frame start and securing the data transmission is done in this way in unlikely case of transmission error. Line discipline collects data that arrived from the serial port and transfers them to the user space in frame by frame fashion, using frame buffers management system. The user space application is protected from data fragmentation and consumes whole packets. The same applies for the transmission. Data are prefixed with a header containing the synchronization pattern and a length and appended with FCS created by XORing all the data.

5.2 BitCloud

BitCloud is a full-featured embedded software stack from Atmel. The stack provides a firmware development platform for reliable, scalable, and secure wireless applications running on Atmel hardware kits such as Zigbit modules. Primary application domains include home automation, commercial building automation, automated meter reading, asset tracking, and industrial automation. BitCloud is fully compliant with ZigBee PRO and ZigBee standards for wireless sensing and control. BitCloud is used for an example implementation of WSN integration, the functionality of the gateway can be easily extended for another networks.

5.3 JamVM

JamVM is a small, open source Java Virtual Machine (JVM) suitable for embedded systems. JamVM is designed to use the GNU Classpath Java class library and support multiple platforms. It features highly optimized interpreter and code-copying JIT (Just In Time) compiler and has support for JNI (Java Native Interface) and Reflection API.

5.3.1 JNI

The Java Native Interface (JNI) is a framework that allows Java code running in a Java Virtual Machine to call native libraries (specific to a hardware platform) written in other languages. In this way, it allows the Java application to interact with a device driver through a device file and isolate this to be the only platform specific part of code. In Lst. 5.3 is listed the exemplary Java part of the JNI implementation of access to ZigBit modules. The native library `libdeviceio.so` is loaded first and then methods can be accessed.

³FCS - Frame Check Sequence

Listing 5.3: JNI interface

```

1 public class DeviceIO {
2     static {
3         System.loadLibrary("deviceio");
4     }
5     public static native int open(String device);
6     public static native int close();
7     public static native int read(byte buf[], int count);
8     public static native int write(byte buf[], int count);
9     public static native int reset();
10 }

```

The listing Lst. 5.4 shows one of the native library functions implemented in C which gets called from the Java application upon calling the open method of the class listed in Lst. 5.4

Listing 5.4: JNI library example

```

1 #define N_ZBT 17
2 static int fd = 0;
3
4 JNIEXPORT jint JNICALL
5 Java_DeviceIO_open (JNIEnv * env, jclass class, jstring devname)
6 {
7     const char *str;
8     int ldisc, err;
9     str = (*env)->GetStringUTFChars(env, devname, NULL);
10    if (NULL == str)
11        return -1;
12    if (fd) {
13        (*env)->ReleaseStringUTFChars(env, devname, str);
14        return -1;
15    }
16    fd = open(str, O_RDWR | O_NOCTTY);
17    ldisc = N_ZBT;
18    err = ioctl(fd, TIOCSETD, &ldisc);
19    if (0 != err || ldisc != N_ZBT) {
20        close(fd);
21        fd = -1;
22    }
23    (*env)->ReleaseStringUTFChars(env, devname, str);
24    return fd;
25 }

```

First few lines of Lst. 5.4 serve the purpose of getting the filename from Java environment to its C representation. On the line 16 the device file is opened and in the next, the line discipline is applied. The calls to open and ioctl functions are platform specific and the library has to be adapted for each specific target platform.

5.4 Higher layer protocols and frameworks

This section provides general overview of protocols and frameworks planned to be used for the final solution of Web integration of the WSN.

5.4.1 ReST [17]

ReST stands for Representational State Transfer. It is an architecture style for designing networked applications and relies on stateless, client-server, cacheable communication protocol such as HTTP. ReSTful applications use HTTP requests to post data (create and/or update), read data (make queries), and delete data. As a programming approach it is a lightweight alternative to Web Services (SOAP) and RPC (Remote Procedure Call). ReST service is:

- Platform-independent
- Language-independent
- Standards-based (runs on top of HTTP)
- Can easily be used in presence of firewalls
- Security and encryption are built on top of HTTP (e.g. HTTPS)

The query is simply encoded inside URL and is sent to the server using simple GET request method of HTTP. Key components forming the ReST architecture:

- *Resources* - identified by logical URLs. Represents both *state* and *functionality*
- *A web of resources* - resources should contain links to other resources
- *Client-server* architecture (application can act as a both)
- *Stateless interaction* - each request contains all the information required to complete it and must not rely on previous interactions
- *Cacheability* - resources should be cacheable whenever possible (with an expiration time). The protocol must allow the server to explicitly specify which resources may be cached and for how long.

Since HTTP is universally used as the ReST protocol, the HTTP cache-control headers are used for this purpose.

Clients must respect the server's cache specification for each resource.

- *Proxy servers* can be used as part of the architecture, to improve performance and scalability.

It is natural to map the resources the WSN exports in a ReSTful way. This allows for simple access to the WSN from the Internet regardless of underlying technology.

Restlet [18]

The Restlet is a lightweight and comprehensive, open source framework for mapping ReST concepts to Java classes. It supports major Internet standards like HTTP,

HTTPS, SMTP, XML, JSON, Atom, WADL and is suitable for both server and client Web applications. It is planned to be used as a basis of PULL service of final solution.

5.4.2 XMPP [19]

The Extensible Messaging and Presence Protocol (XMPP) is an open technology for real-time communication, using the Extensible Markup Language (XML) as the base format for exchanging information. In essence, XMPP provides a way to send small pieces of XML from one entity to another in close to real time. It provides following core services:

- Channel encryption
- Authentication
- Presence
- Contact lists
- One-to-one messaging
- Multi-party messaging
- *Notifications* - XEP-0060 extension, Publish-Subscribe model
- *Service discovery* - XEP-0030 extension

SOX

SOX is the shortcut for the Sensor Over XMPP, the library developed for the *Sensor Andrew* network at the *Carnegie Mellon University*. SOX library is available in C and Java programming language and is based on the XMPP's XEP-0060 extension using the push-based publish-subscribe communication model. It is planned to be implemented as a part of the future development.

5.5 Demo application

Example application consists of the WSN network sending sensor readings at regular intervals and the gateway processing the data and making it accessible on the Web either in HTML or JSON format. The WSN network firmware is implemented using BitCloud ZigBee PRO stack mentioned above. The network coordinator resides at the gateway and starts the network. Sensor nodes join the network and forwards all the data to the coordinator. Upon reception of the packet, coordinator parses it to the JSON representation and sends the data to the gateway using USART interface. Parsing is done using the cJSON library and the example of parsing and sending the data is listed in Lst. 5.5. The notification of nodes joining and leaving the network is done in similar fashion. This way application keeps track of sensor nodes currently present in the network and the Web interface presents current state of the network.

Listing 5.5: Data parsing function

```

1 void boardAbstractionSendData (APS_DataInd_t*ind)
2 {
3     cJSON *root,*fmt; char *out;
4     AppSensorMessage_t *msg = (AppSensorMessage_t*)ind->asdu;
5     root = cJSON_CreateObject();
6     cJSON_AddItemToObject (root, "message", cJSON_CreateString("
       sensor_reading"));
7     cJSON_AddNumberToObject (root, "node_addr", (double) (ind->srcAddress.
       shortAddress));
8     cJSON_AddItemToObject (root, "data", fmt=cJSON_CreateObject());
9     cJSON_AddStringToObject (fmt, "sensor_type", sensorNames[msg->type]);
10    cJSON_AddNumberToObject (fmt, "value", (double) msg->value);
11
12    out = cJSON_Print (root); cJSON_Delete (root);
13    my_WriteUsart (&usartDescriptor, (uint8_t*)out, strlen (out));
14    free (out);
15 }

```

An example of parsed JSON object transmitted to the gateway created from the previous listing is listed in Lst. 5.6.

Listing 5.6: Data in the JSON format

```

1 {
2     "message": "sensor_reading",
3     "data": {
4         "sensor_type": "Temperature",
5         "value": 0
6     },
7     "node_addr": 48059
8 }

```

Function handling the transmission is listed in Lst. 5.7. The data are reassembled and the FCS is checked in the line discipline. The complete packet is then forwarded to the application.

Listing 5.7: USART transmission handling function

```

1 int my_WriteUsart (HAL_UsartDescriptor_t *descriptor, uint8_t *buffer,
   uint16_t length)
2 {
3   uint8_t buf[4], *ptr;
4   uint8_t i, fcs;
5   buf[0] = 0xaa;
6   buf[1] = 0xcc | (uint8_t)((length >> 8) & 0x03);
7   buf[2] = (uint8_t)length;
8   for (i = 0, fcs = 0, ptr = buffer; i < length; i++)
9     fcs ^= *ptr++;
10  buf[3] = fcs;
11  USART_Write(descriptor, (uint8_t*)buf, 3);
12  USART_Write(descriptor, (uint8_t*)buffer, length);
13  USART_Write(descriptor, (uint8_t*)&buf[3], 1);
14  return length;
15 }
```

The application is written in Java, runs in JamVM Java Virtual Machine and consists of two parts. The first part runs in separate thread and opens the device using class listed in Lst. 5.3. It reads all the data frames available and parses it to the JSON object representation form from the serialized (text) format using JSON-java. The JSON objects are then stored using HashMap container. The second part is inherited subclass of the NanoHTTPD.class and implements the actual HTTP server. Upon the GET request the URL is parsed if it contains /nodes, /network, /resources identifiers or identifier of particular resource (e.g. node address) and serves the response accordingly. If the request has .json suffix (e.g. /network.json the JSON object is retrieved from the map and is serialized to its textual representation. The response the contains the object and the content-type HTTP header is set to application/json. If the url does not specify the format the HTML version is build from the data contained in the map and hyper-links to the nodes or resources are built. For the aforementioned requests the application respond with list of currently available nodes, network details such as PAN ID, extended PAN ID and channel on which the network is started or resources available in the network (e.g. available sensor types).

6 CONCLUSION

This work dealt with the problem of interfacing IEEE 802.15.4 and Ethernet networks and tried to do that from as much generic standpoint as possible. The problem of integration of the Wireless Sensor Network is described, and the network architecture that solves this problem by incorporating gateways is proposed.

Later parts described the hardware of a gateway capable of interfacing these networks. The idea behind is its modularity and flexibility, so that the future development and extensibility is not limited by the design. The suggested solution makes use of Linux equipped ARM based SBC, which is powerful enough to run complex applications. Ethernet communication and TCP/IP stack is native to Linux thus software development may concentrate on the original objective of network integration. Furthermore ZigBitTM RF modules capable of IEEE 802.15.4 communication are used mainly because of severe complexity of RF HW design and the requirement of certification of devices working in these frequency bands. Use of OEM modules (with ensured certification) removes this burden. Gateway makes use of up to two of this modules mainly because of the ability to work on both frequency band of IEEE 802.15.4 standard.

The implemented software forms hierarchical architecture, mainly focused on the lower parts comprising firmware for the communication modules and its integration to the Linux operating system. Kernel device driver in form of a module implementing line discipline was developed and the integration with Java platform has been done. It allows the possibility of the use of various frameworks and extensions in the future development.

Sample application forms HTTP server which exports the data from the ZigBee network to the outside world in a simple ReSTful architecture.

Future work is intended in the area of software implementation and large scale Wireless Sensor Network deployment as a proof of concept.

BIBLIOGRAPHY

- [1] *IEEE 802.15.4-2006* [online]. Wikipedia, the free encyclopedia. 04/2010. Available: <http://en.wikipedia.org/wiki/IEEE_802.15.4-2006>. May 17, 2010.
- [2] *IEEE Std 802.15.4TM-2006 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)* [online]. IEEE. NY, USA. 09/2006. 323 p. Available: <<http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>>. ISBN 0-7381-4997-7
- [3] *IEEE Std 802.15.4aTM-2007 Amendment 1: Add Alternate PHYs* [online]. IEEE. NY, USA. 08/2007. 203 p. Available: <<http://standards.ieee.org/getieee802/download/802.15.4a-2007.pdf>>. ISBN 0-7381-5584-5
- [4] *IEEE Std 802.15.4cTM-2009 Amendment 2: Alternative Physical Layer Extension to support one or more of the Chinese 314-316 MHz, 430-434 MHz, and 779-787 MHz bands* [online]. IEEE. NY, USA. 04/2009. 33 p. Available: <<http://standards.ieee.org/getieee802/download/802.15.4c-2009.pdf>>. ISBN 978-0-7381-5913-3
- [5] *IEEE Std 802.15.4dTM-2009 Amendment 3: Alternative Physical Layer Extension to support the Japanese 950 MHz bands* [online]. IEEE. NY, USA. 04/2009. 39 p. Available: <<http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf>>. ISBN 978-0-7381-5915-7
- [6] *ZigBee* [online]. Wikipedia, the free encyclopedia. 05/2010. Available: <<http://en.wikipedia.org/wiki/ZigBee>>. May 18, 2010.
- [7] *ZigBee Alliance* [online]. ZigBee Alliance. 2010. Available: <<http://www.zigbee.org/>>.
- [8] *Welcome to Wireless Sensor Networks Tutorial* [online]. Atmel Corporation. 2009. Available: <<http://meshnetics.com/zigbee-learning/>>.
- [9] *6LoWPAN* [online]. Wikipedia, the free encyclopedia. 04/2010. Available: <<http://en.wikipedia.org/wiki/6LoWPAN>>. May 18, 2010.
- [10] *6LoWPAN* [online]. OpenWSN, Implementing the Internet of Things. 10/2010. Available: <<http://openwsn.berkeley.edu/wiki/OpenLowPan#a6LoWPAN>>. May 16, 2011.

- [11] *IEEE 802.3* [online]. Wikipedia, the free encyclopedia. 05/2010. Available: <http://en.wikipedia.org/wiki/IEEE_802.3>. May 17, 2011.
- [12] *Portux920T : Overview* [online]. taskit GmbH. 2006. Available: <<http://www.taskit.de/en/products/portux/index.htm>>. Dec 18, 2009.
- [13] *Portux920T : Technical details* [online]. taskit GmbH. 2007. Available: <<http://www.taskit.de/en/products/portux/tech.htm>>. Dec 18, 2009.
- [14] *Portux : Technical Reference* [online]. taskit GmbH. Version 1.2. Available: <<http://www.taskit.de/en/support/manuals.htm>>. Dec 18, 2009.
- [15] *ZigBit™ OEM Modules* [online]. 04/2008. Available: <<http://meshnetics.com/>>. Dec 18, 2009.
- [16] M. KOHVAKKA. *Medium Access Control and Hardware Prototype Designs for Low-Energy Wireless Sensor Networks*. TUT, Tampere, 2009. Available: <http://www.tkt.cs.tut.fi/research/daci/pub_open/Kohvakka-Medium_Access_Control_and_Hardware_Prototype_designs_for_Low-Energy_Wireless_Sensor_Networks.pdf>. ISBN 978-952-15-2153-9.
- [17] M. ELKSTEIN. *Learn REST: A Tutorial* [online]. 04/2011. Available: <<http://rest.elkstein.org/>>. May 18, 2011.
- [18] L. RICHARDSON AND S. RUBY. *RESTful Web Services*. O'Reilly Media, Sebastopol, CA, USA. 2007. ISBN-13 978-0-596-52926-0
- [19] P. SAINT-ANDRE, K. SMITH, AND R. TRONÇON. *XMPP: The Definitive Guide*. O'Reilly Media, Sebastopol, CA, USA. 2009. ISBN 978-0-596-52126-4
- [20] A. KAMILARIS. *A lightweight resource-oriented application framework for wireless sensor networks*. Master's thesis, Institute of Pervasive Computing, ETH Zurich, 2009.
- [21] S. WIELAND. *Design and implementation of a gateway for web-based interaction and management of embedded devices*. Master's thesis, Department of Computer Science, ETH Zurich, 2009.
- [22] J. CORBET, A. RUBINI, AND G. KROAH-HARTMAN. *Linux Device Drivers, Third Edition*. O'Reilly Media, Sebastopol, CA, USA. 2005. ISBN 0-596-00590-3
- [23] SREEKRISHNAN VENKATESWARAN. *Essential Linux Device Drivers*. Prentice Hall, Upper Saddle River, New Jersey, USA. 2008. ISBN 978-0-13-239655-4

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

ACK Acknowledgement

ADC Analog-to-Digital Converter

AODV Ad-hoc On-demand Distance Vector

B2B Board-to-Board

CCA Clear Channel Assesment

CFP Contention Free Period

CdS Cadmium Sulfide cell

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CRC Cyclic Redundancy Check

ED Energy Detection

FCS Frame Check Sequence

FFD Full-Function Device

GPIO General Purpose Input/Output

GTS Guaranteed Time Slot

HAL Hardware Abstraction Layer

HTTP Hyper Text Transfer Protocol

HW Hardware

I^2C Inter-Integrated Circuit

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IRQ Interrupt Request

ISM Industrial, Scientific and Medical radio band

JID Jabber ID

- JNI Java Native Interface
- JSON JavaScript Object Notation
- JTAG Digital interface for debugging of embedded devices, also known as IEEE 1149.1 standard interface
- JVM Java Virtual Machine
- LAN Local Area Network
- LDO Low-dropout regulator
- LLC Logical Link Control
- LQI Link Quality Indication
- LR-WPAN Low-rate Wireless Personal Area Network
- MAC Media Access Control
- MCU Microcontroller Unit
- MIME Multipurpose Internet Mail Extensions
- OEM Original Equipment Manufacturer
- PAN Personal Area Network
- PCB Printed Circuit Board
- PD-SAP Phy Data SAP
- PIR Passive Infra-Red
- PLME-SAP Phy Layer Management Entity SAP
- PSDU Phy Service Data Unit
- PXB Portux eXtension Board
- ReST Representational State Transfer
- RF Radio Frequency
- RF4CE Radio Frequency for Consumer Electronics
- RFD Reduced-Function Device

RPC	Remote Procedure Call
RTS/CTS	Request to Send / Clear to Send
SAP	Service Access Point
SBC	Single Board Computer
SOAP	Simple Object Access Protocol
SPI	Serial Peripheral Interface bus
SPDT	Single Pole, Double Throw
SPST	Single Pole, Single Throw
SW	Software
TFTP	Trivial File Transfer Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TWI	Two Wire Interface (I^2C like bus)
UDP	User Datagram Protocol
USART	Universal Synchronous-Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UTP	Unshielded Twisted Pair
UWB	Ultra Wide Band
WADL	Web Application Description Language
WSN	Wireless Sensor Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
ZED	ZigBee End Device
ZC	ZigBee Coordinator
ZR	ZigBee Router

LIST OF APPENDICES

A PCBs and schematics	48
A.1 WSN Gateway	48
A.2 WSN node	51
A.3 RF headers (Trilobite)	53
B partlists	56
B.1 WSN Gateway	56
B.2 WSN Node	57
C Libraries and Frameworks	59
D Content of CD	60

A PCBS AND SCHEMATICS

A.1 WSN Gateway

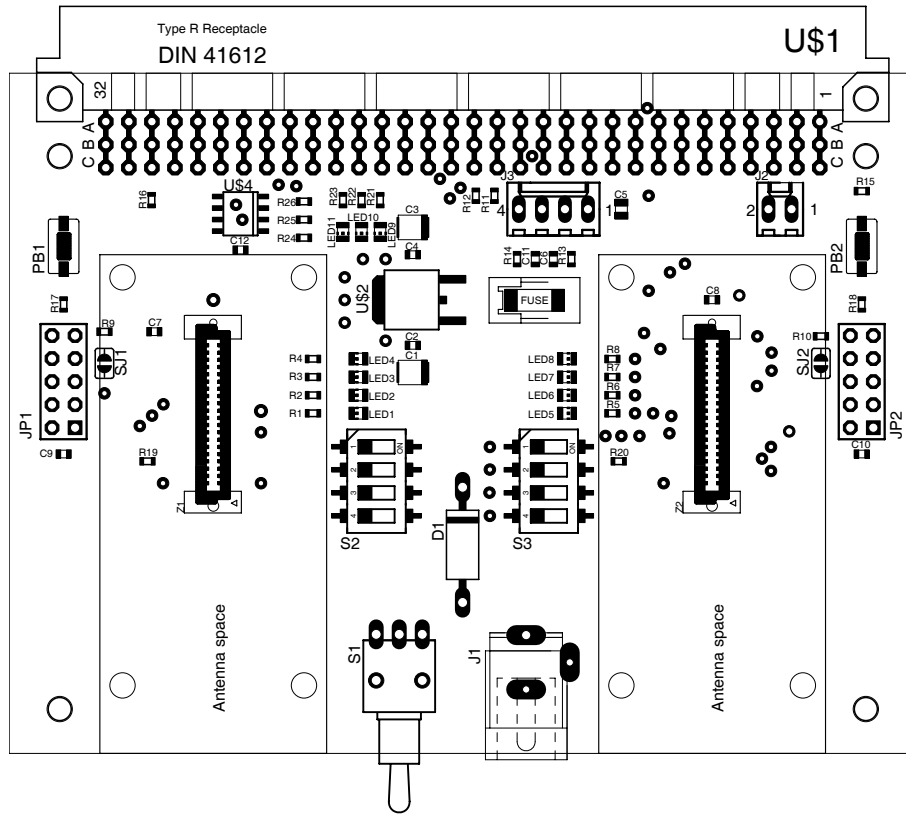


Fig. A.1: WSN Gateway parts placement

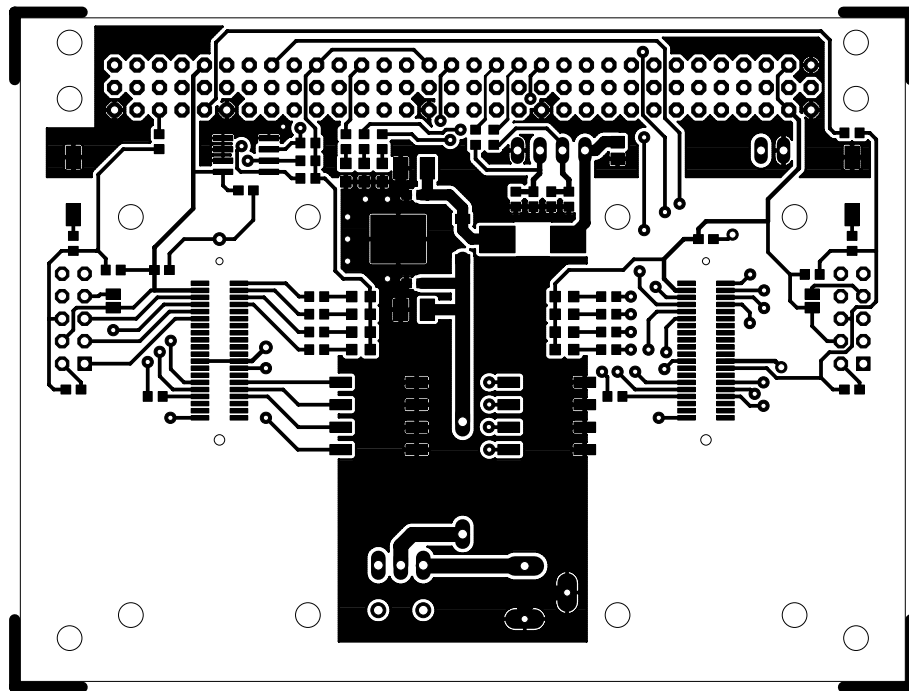


Fig. A.2: WSN Gateway top side

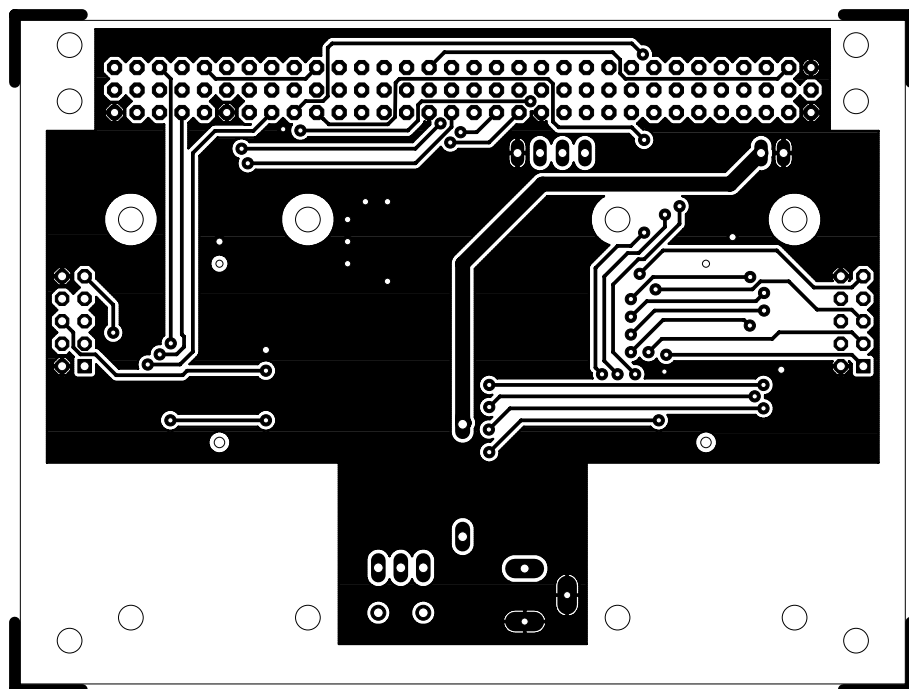
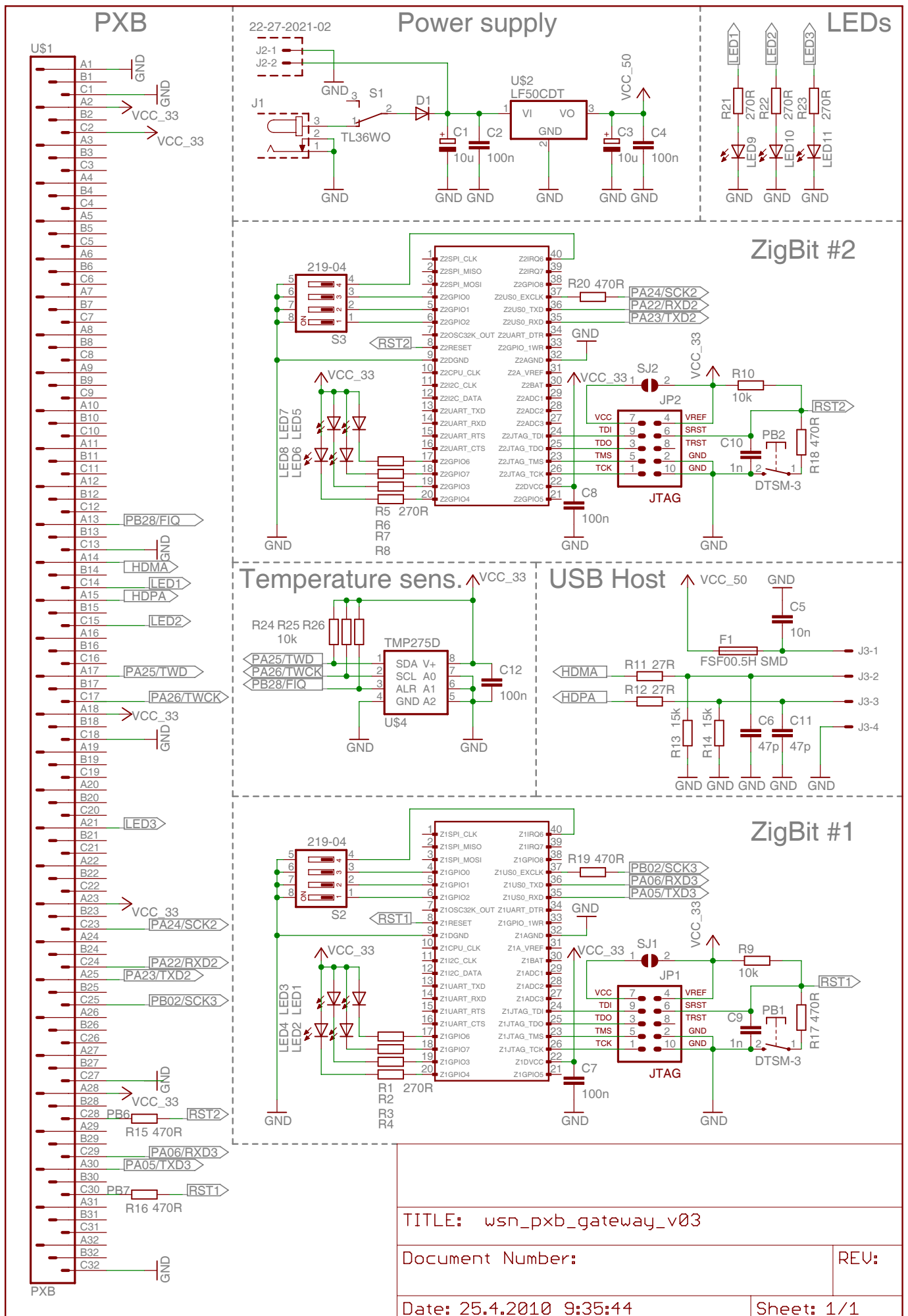


Fig. A.3: WSN Gateway bottom side (mirrored)



TITLE: wsn_pxb_gateway_v03

Document Number:

REV:

Date: 25.4.2010 9:35:44

Sheet: 1/1

Fig. A.4: WSN Gateway schema

A.2 WSN node

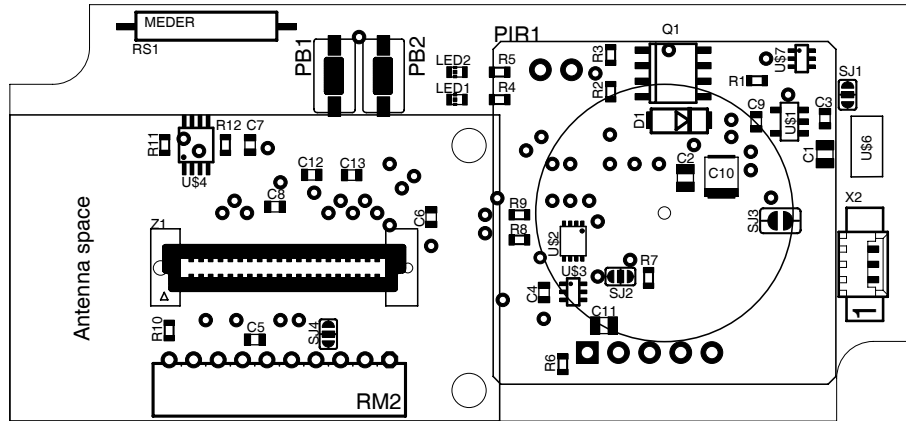


Fig. A.5: WSN PIR node parts placement

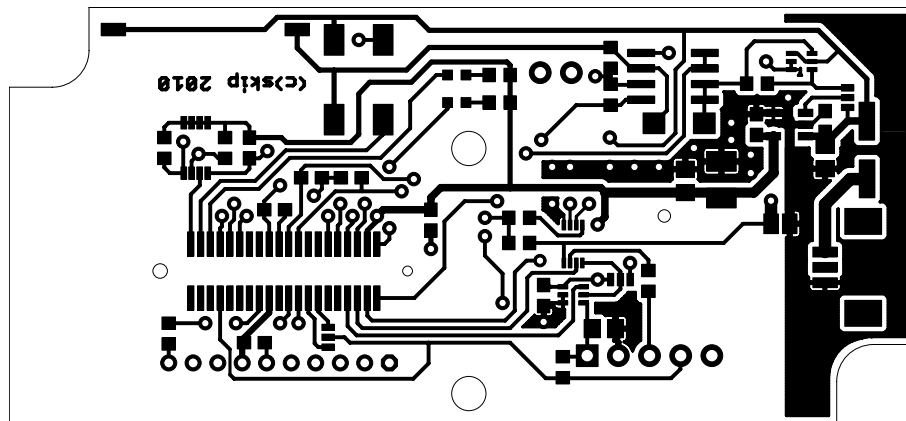


Fig. A.6: WSN PIR node top side

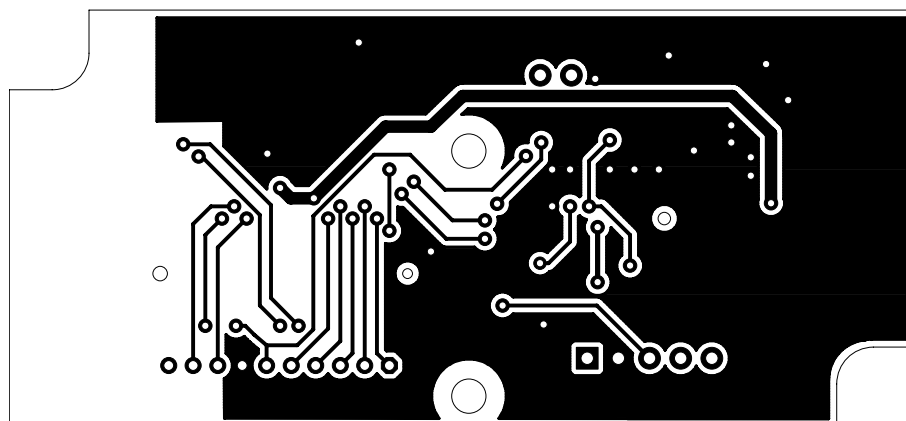
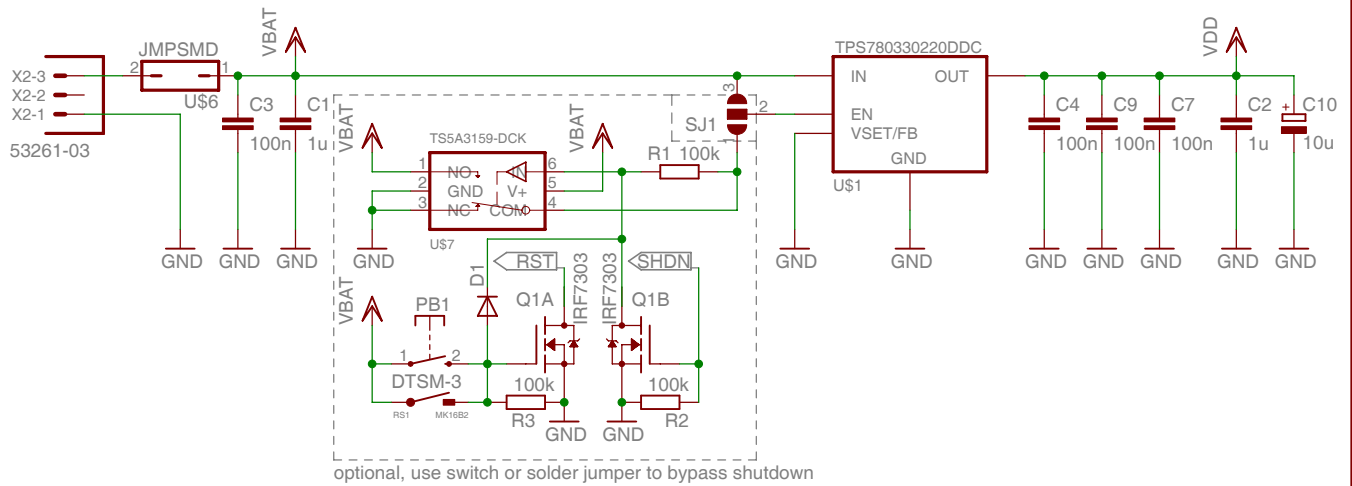
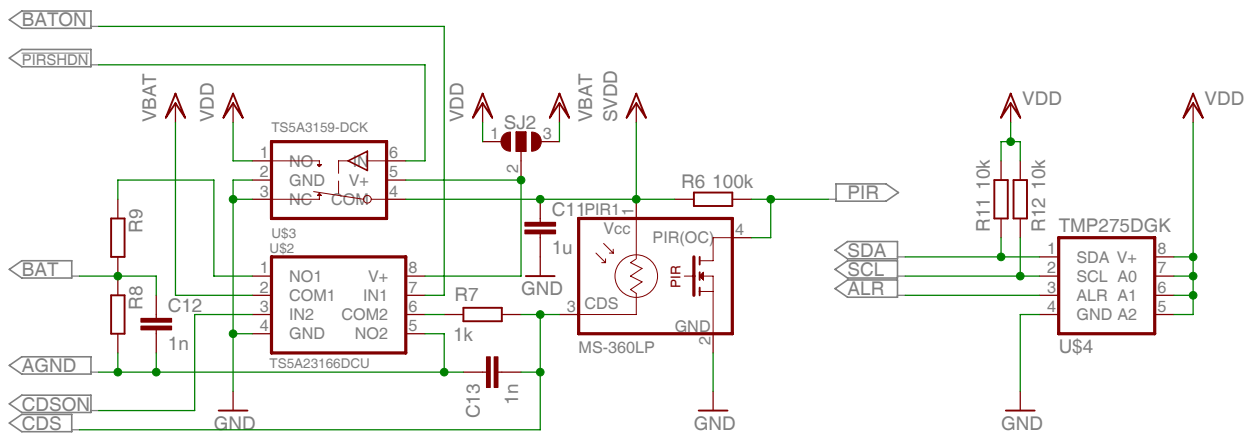


Fig. A.7: WSN PIR node bottom side (mirrored)

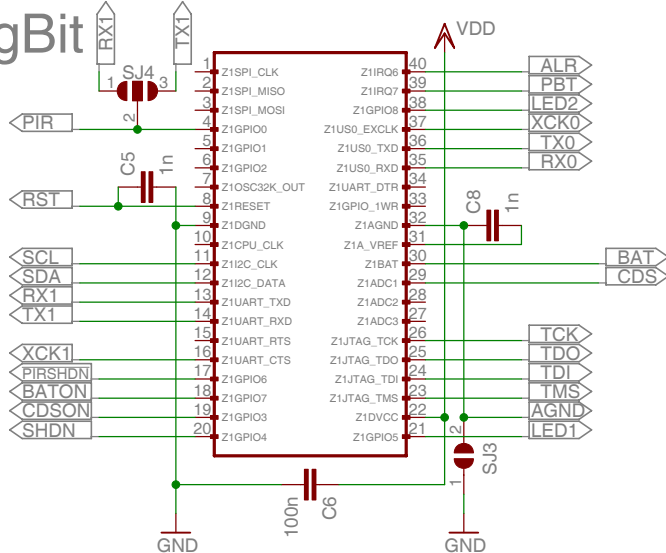
Power part



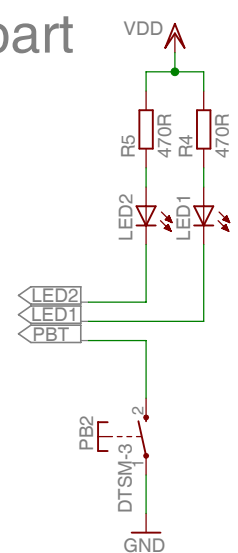
Sensory part



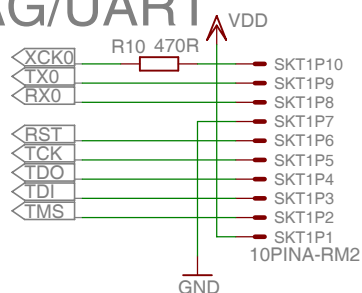
ZigBit



UI IO part



JTAG/UART



TITLE: wsn_pir_node_v04	
Document Number:	REV:
Date: 25.4.2010 12:51:20	Sheet: 1/1

Fig. A.8: WSN PIR node schema

A.3 RF headers (Trilobite)

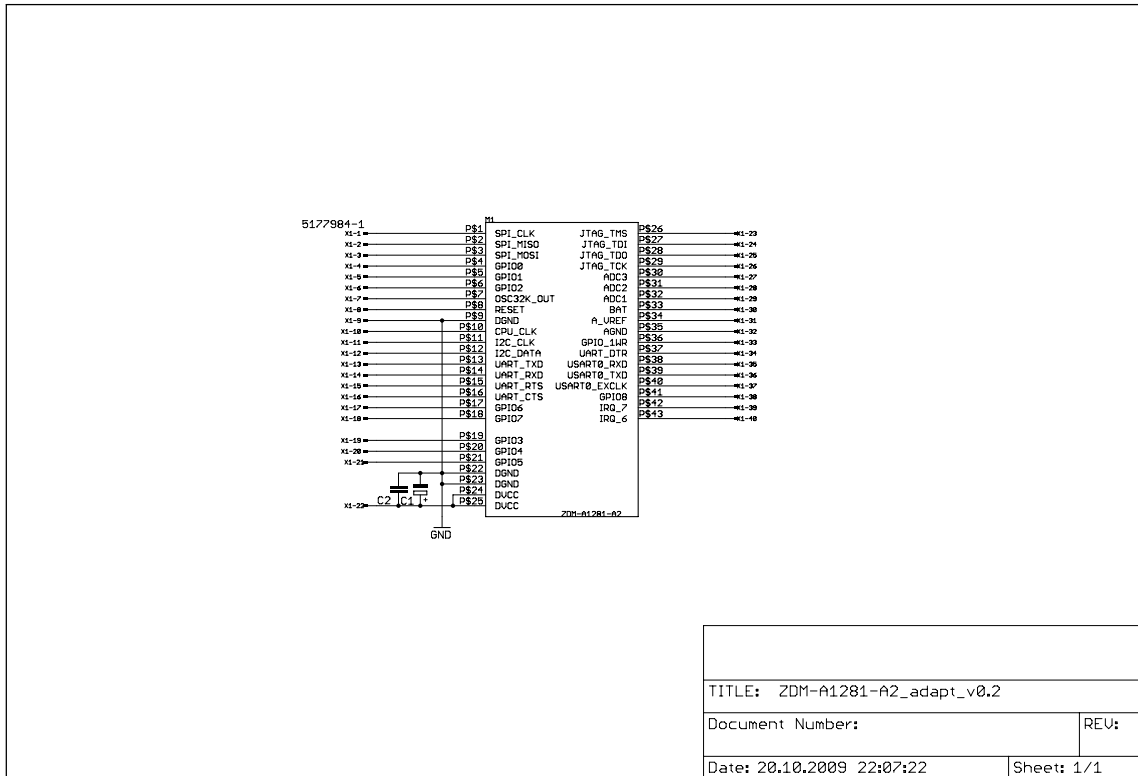


Fig. A.9: Trilobite A2 schematic

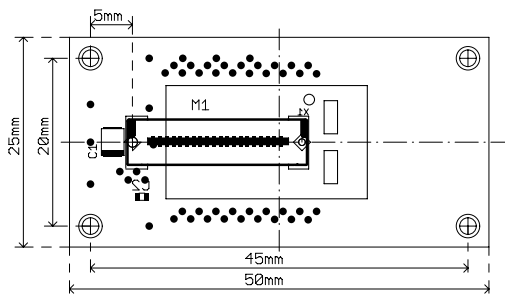


Fig. A.10: Trilobite A2 components placement

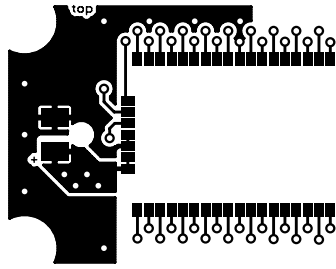


Fig. A.11: Trilobite A2 top PCB layer

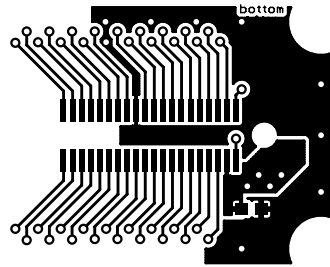


Fig. A.12: Trilobite A2 bottom PCB layer

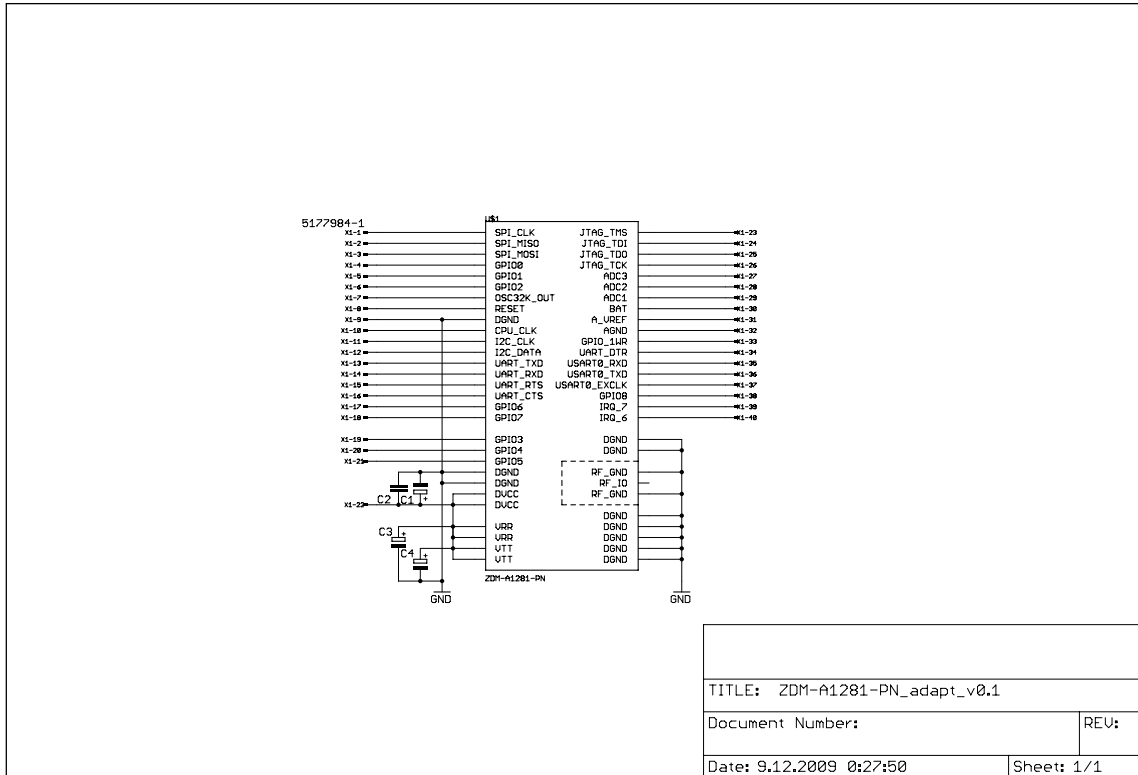


Fig. A.13: Trilobite AMP schematic

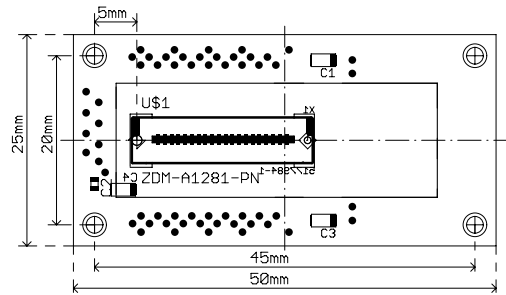


Fig. A.14: Trilobite AMP components placement

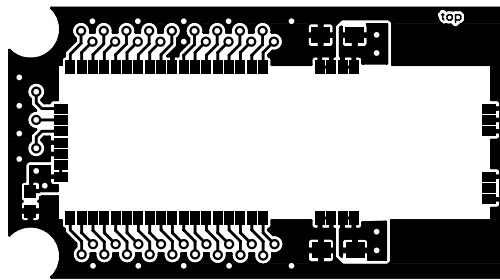


Fig. A.15: Trilobite AMP top PCB layer

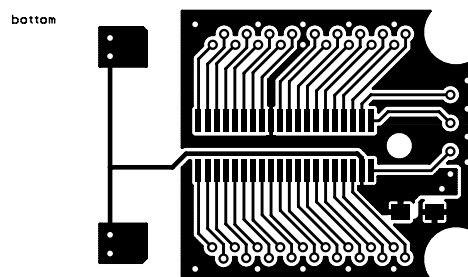


Fig. A.16: Trilobite AMP bottom PCB layer

B PARTLISTS

B.1 WSN Gateway

Listing B.1: Partlist for the Gateway

Partlist

Exported from wsn_pxb_gateway_v03.sch at 25.4.2010 23:26:48

EAGLE Version 5.7.0 Copyright (c) 1988-2010 CadSoft

Part	Value	Device	Package	Library	Sheet
C1	10u	CPOL-EUSMCB	SMC_B	rcl	1
C2	100n	C-EUC0603	C0603	rcl	1
C3	10u	CPOL-EUSMCB	SMC_B	rcl	1
C4	100n	C-EUC0603	C0603	rcl	1
C5	10n	C-EUC0805	C0805	rcl	1
C6	47p	C-EUC0603	C0603	rcl	1
C7	100n	C-EUC0603	C0603	rcl	1
C8	100n	C-EUC0603	C0603	rcl	1
C9	1n	C-EUC0603	C0603	rcl	1
C10	1n	C-EUC0603	C0603	rcl	1
C11	47p	C-EUC0603	C0603	rcl	1
C12	100n	C-EUC0603	C0603	rcl	1
D1		DIODE-DO15-12	DO15-12	diode	1
F1	FSF00.5H SMD	FUSEFSFSMD	FSFSMD	amina	1
J1		JACK-PLUG0	SPC4077	con-jack	1
J2	22-27-2021-02	22-27-2021-02	6410-02	con-molex	1
J3		22-27-2041-04	6410-04	con-molex	1
JP1	AVR-JTAG-10ST	AVR-JTAG-10ST	AVR-JTAG-10	amina	1
JP2	AVR-JTAG-10ST	AVR-JTAG-10ST	AVR-JTAG-10	amina	1
LED1		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED2		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED3		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED4		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED5		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED6		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED7		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED8		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED9		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED10		LEDCHIP-LED0805	CHIP-LED0805	led	1
LED11		LEDCHIP-LED0805	CHIP-LED0805	led	1
PB1	DTSM-3	DTSM-3	DTSM-3	amina	1
PB2	DTSM-3	DTSM-3	DTSM-3	amina	1

R1		R-EU_R0603	R0603	rcl	1
R2		R-EU_R0603	R0603	rcl	1
R3		R-EU_R0603	R0603	rcl	1
R4	270R	R-EU_R0603	R0603	rcl	1
R5		R-EU_R0603	R0603	rcl	1
R6		R-EU_R0603	R0603	rcl	1
R7		R-EU_R0603	R0603	rcl	1
R8	270R	R-EU_R0603	R0603	rcl	1
R9	10k	R-EU_R0603	R0603	rcl	1
R10	10k	R-EU_R0603	R0603	rcl	1
R11	27R	R-EU_R0603	R0603	rcl	1
R12	27R	R-EU_R0603	R0603	rcl	1
R13	15k	R-EU_R0603	R0603	rcl	1
R14	15k	R-EU_R0603	R0603	rcl	1
R15	470R	R-EU_R0603	R0603	rcl	1
R16	470R	R-EU_R0603	R0603	rcl	1
R17	470R	R-EU_R0603	R0603	rcl	1
R18	470R	R-EU_R0603	R0603	rcl	1
R19	470R	R-EU_R0603	R0603	rcl	1
R20	470R	R-EU_R0603	R0603	rcl	1
R21	270R	R-EU_R0603	R0603	rcl	1
R22	270R	R-EU_R0603	R0603	rcl	1
R23	270R	R-EU_R0603	R0603	rcl	1
R24	10k	R-EU_R0603	R0603	rcl	1
R25		R-EU_R0603	R0603	rcl	1
R26		R-EU_R0603	R0603	rcl	1
S1	TL36WO	TL36WO	TL3XWO	switch	1
S2	219-04	219-04	CTS-219-04	switch-dil	1
S3	219-04	219-04	CTS-219-04	switch-dil	1
SJ1		SJ	SJ	amina	1
SJ2		SJ	SJ	amina	1
U\$1	PXB	PXB	PXB	amina	1
U\$2	LF50CDT	LF50CDT	DPACK	amina	1
U\$4	TMP275D	TMP275D	SO8	amina	1
Z1		ZB_MODULEL	ZB_HEADER_LONG	amina	1
Z2		ZB_MODULEL	ZB_HEADER_LONG	amina	1

B.2 WSN Node

Listing B.2: Partlist for the WSN node

Exported from wsn_pir_node_v04.brd at 25.4.2010 23:28:40

EAGLE Version 5.7.0 Copyright (c) 1988-2010 CadSoft

Part	Value	Package	Library	Position (mm)
------	-------	---------	---------	---------------

C1	1u	C0805	rcl	(66.55 21.8)
C2	1u	C0805	rcl	(55.15 19.85)
C3	100n	C0603	rcl	(66.55 24.7)
C4	100n	C0603	rcl	(43.6 10.5)
C5	1n	C0603	rcl	(20 6.65)
C6	100n	C0603	rcl	(34.4 16.65)
C7	100n	C0603	rcl	(19.6 22.55)
C8	1n	C0603	rcl	(21.65 17.5)
C9	100n	C0603	rcl	(60.95 24.45)
C10	10u	SMC_B	rcl	(58.1 19.95)
C11	1u	C0805	rcl	(48.5 7.8)
C12	1n	C0603	rcl	(24.65 20.1)
C13	1n	C0603	rcl	(27.9 20.1)
D1		DO214AC	diode	(54.65 24.55)
LED1		CHIP-LED0603	led	(36.5 26.25)
LED2		CHIP-LED0603	led	(36.5 28.5)
PB1	DTSM-3	DTSM-3	amina	(26.5 28.1)
PB2	DTSM-3	DTSM-3	amina	(30.5 28.1)
PIR1	MS-360LP	MS-360	amina	(53.45 17)
Q1	IRF7303	SO8	amina	(54.15 28.4)
R1	100k	R0603	rcl	(61 27.8)
R2	100k	R0603	rcl	(49.05 26.9)
R3	100k	R0603	rcl	(49.05 29.9)
R4	470R	R0603	rcl	(40 26.25)
R5	470R	R0603	rcl	(40 28.5)
R6	100k	R0603	rcl	(45.15 4.65)
R7	1k	R0603	rcl	(52.15 11.7)
R8		R0603	rcl	(41.6 14.8)
R9		R0603	rcl	(41.6 16.85)
R10	470R	R0603	rcl	(13 7.4)
R11	10k	R0603	rcl	(12.65 22.55)
R12	10k	R0603	rcl	(17.6 22.55)
RS1	MK16B2	MK16*2	amina	(16 32.2)
SJ1		SJ_2S	amina	(68.4 26.65)
SJ2		SJ_2S	amina	(49.85 11.8)
SJ3		SJ	amina	(62.9 16.35)
SJ4		SJ_2S	amina	(26 7.1)
SKT1	10PINA-RM2	ZL265-10SG-A	amina	(22 5)
U\$1	TPS780330220DDC	SOT23-5	amina	(63.65 24.45)
U\$2	TS5A23166DCU	US8	amina	(46.75 16.25)
U\$3	TS5A3159-DCK	SC70-6L	amina	(46 10.55)
U\$4	TMP275DGK	MSOP8	amina	(15.2 22.55)
U\$6	JMP5MD	2P-SMD	amina	(70 22.5)
U\$7	TS5A3159-DCK	SC70-6L	amina	(64.65 29.6)
X2	53261-03	53261-03	con-molex	(69.05 12.8)
Z1		ZB_HEADER_SHORT	amina	(40 25)

C LIBRARIES AND FRAMEWORKS

Listing of libraries, sources and frameworks used for software development.

Linux

Sources for Linux-2.6.22

<http://download.armbedded.eu/software/linux-2.6.22-taskit4.tgz>

BitCloud

Full-Featured, Second Generation Embedded ZigBee PRO Software Stack

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4495

cJSON

An ultra-lightweight, portable, single-file, simple-as-can-be ANSI-C compliant JSON parser

<http://sourceforge.net/projects/cjson/>

JSON-java

A reference implementation of a JSON package in Java

<https://github.com/douglascrockford/JSON-java>

NanoHTTPD

A free, simple, tiny, nicely embeddable HTTP server in Java

<http://elonen.iki.fi/code/nanohttpd/>

JamVM

JamVM v1.5.4, an extremely small Java Virtual Machine

<http://jamvm.sourceforge.net/>

GNU Classpath

GNU Classpath, Essential Libraries for Java, is a GNU project to create free core class libraries for use with virtual machines and compilers for the java programming language

<http://www.gnu.org/software/classpath/classpath.html>

D CONTENT OF CD

The CD contains following folders and items:

Linux

Contains the Linux kernel with all the modifications. The line discipline implementation is in the file

```
linux-2.6.22-taskit4/drivers/char/n_zbt.c
```

JamVM

Contains the Java Virtual Machine and GNU Classpath cross-compiled for the Portux SBC

rootfs

Contains the root filesystem of the Portux which the linux mounts upon startup from SD card with all the modules, drivers and applications necessary

BitCloud

Contains the BitCloud stack with all the modifications for custom boards and example application. The ZigBit firmware part of the application is in the folder `BitCloud_ZIGBIT_1.11.0/Applications/GatewayEx/`

Java

Contains JNI library for interfacing with the device and all the Java source files and classes forming the application

PCB

Contains printed circuit board layouts and schematics for all the hardware

text

Contains electronic version of this thesis