Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Control Engineering

Wireless sensor network for monitoring patients with Parkinson's disease

Diploma Thesis

Author: Martin Auersvald

Supervisor: Ing. Jiří Trdlička

Thesis Due: January 2008

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Martin A u e r s v a l d

Obor: Technická kybernetika

Název tématu: Bezdrátová senzorová síť pro monitorování pacientů s Parkinsonovou chorobou

Zásady pro vypracování:

1. Seznámení se s operačním systémem TinyOS a protokolem IEEE 802.15.4/ZigBee pro senzorové bezdrátové sítě.
2. Seznámení se s aktuální implementací protokolu 802.15.4 pro TinyOS a její rozšíření pro platformu TelosB.
3. Návrh a implementace rozšiřujících modulů pro platformu TelosB použitelných jako koncový senzorový a centrální datový prvek.
4. Návrh a implementace aplikačního softwaru s využitím TinyOS a protokolu 802.15.4 pro sběr dat v senzorové síti založené na výše uvedených prvcích platformy TelosB.
5. Rozšíření navrženého řešení sběru a distribuce dat pro vlastní navrženou senzorovou platformu.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Jiří Trdlička

Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

[illegible]

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S. -4-

v zast. [illegible]

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 21.02.2007

Declaration

I hereby declare that I have written my diploma thesis myself and used only the sources (literature, projects, SW etc.) listed in the enclosed bibliography.

In Prague, ________________

____
Martin Auersvald

Acknowledgements

I am very grateful to my thesis advisor, Ing. Jiří Trdlička, for his guidance, support, comments and willingness to discuss any problems. I also thank my second advisor, Ing. Petr Jurčík, for his helpful suggestions. My special appreciation goes to Ing. André Cunha for his advice and help at working with open-ZB stack, and Bc. Jiří Kubias for helping me during realization of hardware parts.

I would like to special thank my family and friends for their constant support and encouragement during the course of my studies.

Abstrakt

Cílem této práce je návrh bezdrátové sensorové sítě pro monitorování stavu pacientů s Parkinsonovou chorobou.
Síť je vytvořena z pěti zařízení ve hvězdicové topologii pomocí bezdrátové technologie ZigBee. Jako zařízení jsou použity sensorové moduly Tmote Sky (TelosB platforma) firmy Moteiv. Komunikační model topologie hvězda je centralizován, každý ze čtyř uzlů vysílá svoje data do hlavního uzlu, který vystupuje jako koordinátor sítě. Data jsou z těla pacienta snímána pomocí navrženého rozšiřujícího modulu s analogovým nebo digitálním akcelerometrem. Programy pro zařízení pro sběr dat, jejich vysílání, přijímání a přenos do osobního počítače jsou vytvořeny pomocí prostředí TinyOS, jazyka nesC a Open-ZB stacku, který byl vyvinut na Polytechnickém institutu v Portugalském Portu. Open-ZB stack je open-source implementace (poskytována jako sada nástrojů) IEEE 802.15.4 a ZigBee protokolu v TinyOS/nesC. Programy pro osobní počítač pro zpracovávání a zobrazování dat jsou vytvořeny pomocí MS Visual Studio 2005 v programovacím jazyce C# a jazyce C pro UNIX OS.

Abstract

The goal of this work is a proposition of wireless sensor network for monitoring patients with Parkinson's disease.
The network is created from five devices in star topology by using ZigBee wireless technology. As devices are used the sensor modules Moteiv Tmote Sky (TelosB platform). The communication paradigm in the star topology is centralized, each from four nodes sends its data to the principal node, which operates as a coordinator. Data are acquired from the patient body by designed expansion module with analog or digital accelerometer. A Tmote programs for data acquisition, data sending, receiving, processing and data transmission to the personal computer are built by using the TinyOS operating environment, nesC language and Open-ZB stack, which is developed in Polytechnic Institute of Porto, Portugal. Open-ZB stack is the open source implementation (provided as a tool) of the IEEE 802.15.4 and ZigBee protocol in TinyOS/nesC. A PC programs for data processing and displaying are created by MS Visual Studio 2005 in C# language and in C language for UNIX OS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Parkinson's disease

Parkinson's disease is a degenerative disorder of the central nervous system that often impairs the sufferer's motor skills and speech. It is characterized by muscle rigidity, tremor, a slowing of physical movement and, in extreme cases, a loss of physical movement. The most widely used form of treatment is Levodopa in various forms. Levodopa is used as a pharmacological substance (drug) to increase dopamine levels. The occurrence of Parkinson's symptoms is needed to be known for proper diagnosis and treatment evaluation. It is, therefore, necessary to monitoring patients some ways.

1.2 Monitoring patient activity

Since 90's, a small electronic devices have been used for monitoring human activity. These devices, called as *ActiGraphs*, that generally consists of an inertial sensor (accelerometer and/or gyroscope), a filter, a memory, an interface and digital circuitry, and that, when is worn by an individual (typical on wrist, ankle, elbow, shoulder or chest), records and reports levels of activity (number of treshold crossings during time periods). The typical symptoms of Parkinson's disease, tremor and dyskinesias (abnormal involuntary movements), are showed as periods of high activity and is necessary use of several sensors placed over the body because of detecting all of symptoms. The big restrictive disadvantage is cable connection of a number of sensors fixed on patient's body. Such solution is not user friendly. Wireless connection is more suitable.

1.3 Wireless Sensor Networks (WSNs)

A wireless sensor network (WSN) is a collection of spatially distributed autonomous wireless devices, also called as *nodes* or *motes*, using sensors to cooperatively monitor physical or environmental conditions at different locations and that form a certain network topology. A WSNs are used in applications to monitor/collect data that would be difficult or expensive to monitor using wired sensors. A typical wireless sensor contains a sensor(s), a wireless communication interface (such as radio transceiver), an energy source (typically batteries) and a small microcontroller. Establish such a WSN is perfect for monitoring human activity.

1.4 General requirements

The following requirements have a basis in paper [1].

General requirements for this thesis:

- create the wireless network (Body Area Network) for data collection on a patient based on an appropriate wireless technology
- the network consists of one central unit and four (two on both wrists and two on both ankles) to six (other on chest e.g.) measurement units, as depicted in Figure 1.1, therefore, the network will create in star topology
- central unit synchronizing data collection and storing all the measured data values for later processing in a personal computer or in a memory, memory capacity should be sufficient for storage of all-day measurements from all sensors
- each of measurement unit contains an inertial sensor - three-axis (3D) accelerometer offering acceleration data and forms a simple ActiGraph this way
- acceleration range of about 5g is sufficient in Parkinsonian patients, greater range leads to lower sensitivity
- to describe human movement a sufficient acquisition frequency of about 50Hz should be provided
- transmition range is not needed to be greater than size of human body
- low power consumption is required, batteries in all units have to suffice at least for a day-long measurement, batteries can be recharged at night
- low cost solution

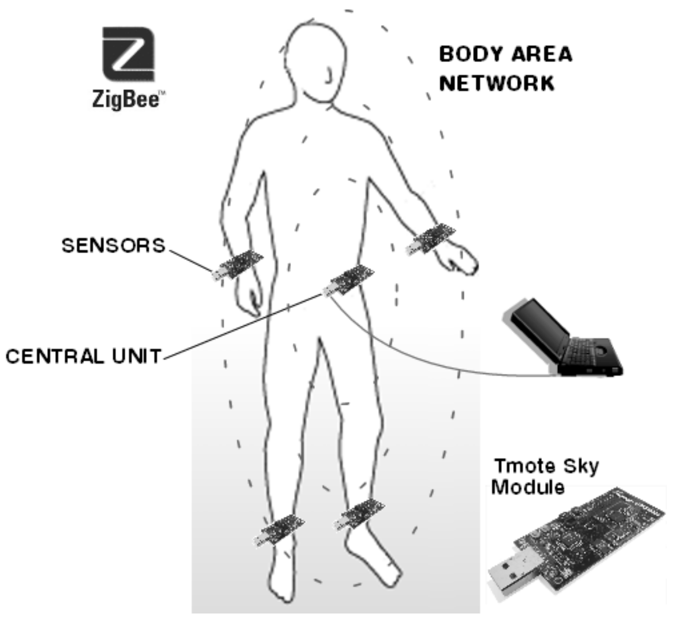


Figure 1.1: Spatial arrangement of the network nodes.

- additive functions as A/D converter in the central unit or free computational capacity for simple algorithms is welcome
- weight of the units is also not trivial

1.5 Analysis of the solution and secondary requirements

The choice of wireless communication protocol depends on the context in which the network is used. According to the requirements was chosen the most suitable wireless technology - ZigBee technology, optimized for low-cost, low-power consumption and short-range radio frequency transmissions. The process of selection also resulted from the comparison of primary parameters of used standards for wireless communication [2]. Comparison of wireless technologies is presented in Table 1.1.
The ZigBee protocol is implemented on top of the IEEE 802.15.4 radio communication standard. The ZigBee specification is managed by a non-profit industry consortium of semiconductor manufacturers, technology providers and other companies, all together designated the ZigBee Alliance. In the Chapter 2. of this thesis is provided an overview of the IEEE 802.15.4 and ZigBee standard.
As hardware platform for wireless communication has been chosen the Tmote Sky (exactly the TelosB platform) sensor module (see Figure 1.1) from Sentilla/Moteiv Corporation with a chip supporting IEEE 802.15.4 standard from Chipcon Corporation. The Sentilla company was previously named Moteiv and focused on mote hardware, now the relaunched

Commercial Name Standard	GPRS/GSM 1xRTT/CDMA	Wi-Fi™ 802.11b	Bluetooth™ 802.15.1	ZigBee™ 802.15.4
Direction of Application	Voice and Data	Web, Email, Video	Substitute for cable	Monitoring and Control
System Resources (Memory)	16MB and more	1MB and more	250KB and more	4KB - 32KB
Battery Life (Days)	1 - 7	0.5 - 5	1 - 7	100 - 1000
Max. Network Size (Number of Nodes/Net)	1	32	7	65000
Data Rate (Kb/s)	64 - 128	11000	720	20 - 250
Communication Range (m)	1000 and more	1 - 100	1 - 10	1 - 100
Advantages	Availability, Quality	Rate, Flexibility	Cost, Simplicity	Reliability, Power/Cost

Table 1.1: Comparison of Wireless Technologies.

Sentilla will create software tools to used on motes form other companies, such as Texas Instruments' MSP430 microprocessor. TelosB platform modules are now only produced by Crossbow Technologies. Currently, the Wireless Sensor Networks Group at Department of Control Engineering have nine operational devices from Moteiv and ten devices from Crossbow that are available. In the Chapter 3. is provided an overview of the Tmote Sky module. Open source implementation of the IEEE 802.15.4 and ZigBee protocols are not available, therefore, the IPP-HURRAY Research Group in Porto developing own open source implementation of IEEE 802.15.4/ZigBee, which is providing as the toolset - the open-ZB stack. The protocol stack is developed in nesC language, under the TinyOS operating system (open-source OS designed for embedded wireless systems) for the MICAz platform from CrossBow. It was necessary to ported the open-ZB stack to the TelosB platform at first when we would like to use the Tmote Sky sensor modules.

Secondary requirement for this thesis:

- porting open-ZB stack implemented for the MICAz platform to the TelosB platform

In the Chapter 3. is described the TinyOS and nesC.
In the Chapter 4. is introduced the open-ZB stack, its structure, and its porting, modifications and final implementation for the TelosB platform.
As a simple ActiGraph devices are designed expansion modules (as peripherals) with an accelerometers. As appropriate accelerometers for measurement units has been chosen the analog MMA7260Q (eventually MMA7261Q) by Freescale Semiconductor and the digital LIS3LV02DQ by STMicroelectronics. Design of expansion modules with these accelerometers for Tmote Sky devices is presented in the Chapter 5.

Data collection applications for Tmote Sky devices, for central node and end nodes, written by using open-ZB stack supporting the TelosB platform is presented in the Chapter 6.
Results, problems, necessary changes and experiences in using the Open-ZB stack and TinyOS/nesC are presented in Chapter 7.
PC applications for data processing and displaying, written in C and C# language, are presented in the Chapter 8.

Chapter 2

IEEE 802.15.4 and ZigBee

2.1 Introduction

The IEEE 802.15.4 [IEEE 802.15 WPAN™ Task Group 4 (TG4)] protocol is often associated with the ZigBee protocol. The relationship between IEEE 802.15.4 and ZigBee is similar to that between IEEE 802.11 and the Wi-Fi Alliance.
The IEEE 802.15.4 protocol is as well as ZigBee technology optimized for low-cost, low-power consumption, low-data-rate and low-complexity short-range radio frequency transmissions in wireless communications, with typically requirements of sensor networks, automation and remote control applications. IEEE 802.15.4 commission started working on standard a short while later. Then the ZigBee Alliance, which is an transnational organization with over 200 member companies (ref. October 2006), and the IEEE Standards Association decided to join forces and ZigBee is the commercial name for this technology. They have been working in conjunction in order to specify a full protocol stack.
The IEEE 802.15.4 focuses on the specification of the lower two layers standard Open Systems Interconnection (OSI) model of the protocol for Low-Rate Wireless Private Area Networks (LR-WPAN's). That are the Medium Access Control (MAC) layer and Physical layer. It is operating in an unlicensed, international frequency band.
ZigBee Alliance aims to provide the upper layers of the protocol, from network to the application layer, for interoperable data networking, security services and a range of wireless home and building control solutions (device objects and profiles). ZigBee Alliance provide also interoperability compliance testing, marketing of standard and advanced engineering for the evolution of the standard.

Standard IEEE 802.15.4-2003 defined the protocol and compatible interconnection for data communication devices. The IEEE 802.15.4-2006 revision extends, makes improvements and removes ambiguities in the IEEE 802.15.4-2003. Specifications are available at the website of the IEEE 802.15 Working Group for WPAN [3].

The ZigBee 1.0 specification was released in December 2004 (first stack called "ZigBee 2004") and is now obsolete. The enhanced specification was released to the public in December 2006 (2nd stack called "ZigBee 2006") and contains several changes. The newest ZigBee specification announced in October 2007 looking to extend the ZigBee 2006 specification capabilities and is now publicly available at the website of the ZigBee Alliance [4]. ZigBee 2007 at the network level is not backwards-compatible with ZigBee 2004/2006.

The organization of the IEEE 802.15.4/ZigBee protocol architecture is presented in Figure 2.1.

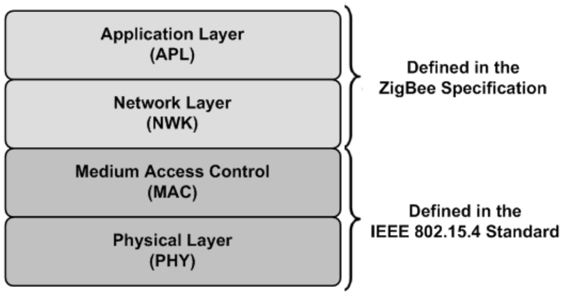

Application Layer (APL)
Network Layer (NWK)
Medium Access Control (MAC)
Physical Layer (PHY)
Defined in the ZigBee Specification
Defined in the IEEE 802.15.4 Standard

Figure 2.1: IEEE 802.15.4/ZigBee protocol stack architecture

2.2 General Description of IEEE 802.15.4

As has allready been noted, the main features of this standard are low cost, low power consumption, low data rate and network flexibility in an adhoc self-organizing network among inexpensive devices, which can be fixed, portable and/or moving. It is developed for applications with limited throughput requirements, which cannot handle the power consumption of heavy protocol stack.

2.2.1 IEEE 802.15.4 WPAN

A. Network Devices

LR-WPAN support two diferent types of devices.

- **Full Function Device (FFD)**
  A FFD is a device that can support three operation modes
  - A *Personal Area Network Coordinator (PAN Coordinator)*
    The main controller of the personal area network. This device identifies its own network. To this network can be associated other devices.

  - A *Coordinator*

    It provides synchronization services through the transmission of beacons and must be associated to a PAN coordinator. A coordinator does not create its own network.
  - A simple *device*

    A device which does not implement the previous functions.

- **Reduced Function Device (RFD)**

  A RFD is a device operating with minimal implementation of the IEEE 802.15.4 protocol. It can only associate with a single FFD at a time. A RFD is intended for applications that are extremely simple, do not need to send large amounts of data.

A LR-WPAN must include at least one FFD acting as a PAN coordinator that provides global synchronization services to the network and manages potential FFDs and RFDs.

B. Network Topologies

1. **Star Topology**

   In the star topology, the communication is established between devices and a single central controller, a unique node operates as the PAN coordinator. After an FFD is activated for the first time, it may establish its own network and become the PAN coordinator. The PAN coordinator chooses a PAN identifier, which is not currently used by any other network in the radio sphere of influence. This allows each star network to operate independently. Each device (FFD or RFD) that is joined the network, communicate with other devices through PAN coordinator, therefore, the PAN coordinator have significant power consumption, hence may be mains powered. The devices will most likely be battery powered.

2. **Peer-to-Peer (Mesh) Topology**

   The peer-to-peer (mesh) topology also includes a PAN coordinator. In contrast to star topology, each device can directly communicate with any other device in its radio range and communication process does not rely on a particular node. This enlarges networking flexibility, but it causes an additional complexity.

3. **Cluster-Tree Topology**

   The Cluster-Tree is a special case of a peer-to-peer network in which most devices are FFDs. An RFD may connect to cluster-tree network as a leave node at the end of a branch. Any of the FFD can act as a coordinator and provide synchronization services to other devices and coordinators. Only one of these coordinators is the PAN coordinator, which identifies the entire network. The standard IEEE 802.15.4 [3] does not define how to build a cluster-tree network. It only indicates that this is possible and may be initiated by higher layers (network layer). In [5, 10] is simply presented, how may be it performed. The network layer introducted in the ZigBee specification [4] uses the primitives provided by the IEEE 802.15.4 MAC sublayer and propose the cluster-tree protocol.

Figure 2.2 presents a network topologies supported by the IEEE 802.15.4 standard.

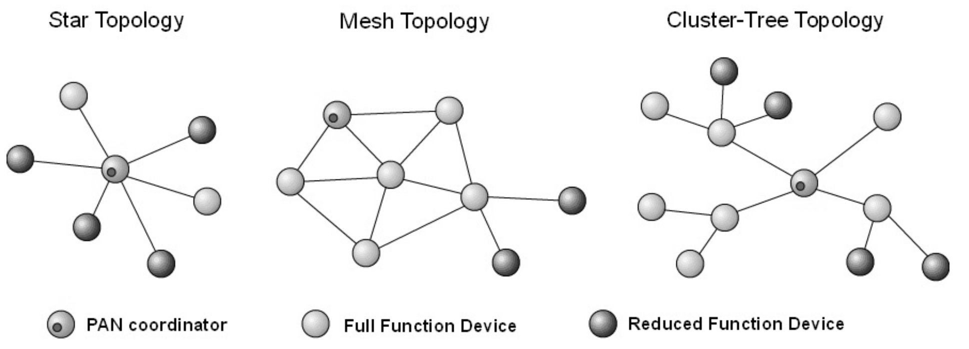


Figure 2.2: Topology Models.

2.2.2 IEEE 802.15.4 Physical layer

The physical layer is responsible for data transmission and reception using a certain radio channel and according to a specific modulation and spreading technique. The IEEE 802.15.4 offers three operational frequency bands: 2.4 GHz, 915 MHz and 868 MHz. There is a single channel between 868 and 868.6 MHz, 10 channels between 902 and 928 MHz, and 16 channels between 2.4 and 2.4835 GHz (see Figure 2.3). The protocol also allows dynamic channel selection, channel switching, a scan function that steps through a list of supported channels in search of a beacon, receiver energy detection and link quality indication.

The data rate is 250 kbps at 2.4 GHz, 40 kbps at 915 MHZ and 20 kbps at 868 MHz. Lower frequencies are more suitable for longer transmission ranges due to lower propagation losses. Low rate transmissions provide better sensitivity and larger coverage area. Higher rate means higher throughput, lower latency or lower duty cycles. All of these frequency bands are based on the Direct Sequence Spread Spectrum (DSSS) spreading technique (see [3, 8]). The features of each frequency band are summarized in Table 2.1.

The physical layer of the IEEE 802.15.4 is in charge of the following tasks:

- Activation and deactivation of the radio transceiver

  The radio transceiver may operate in one of three states: transmitting, receiving or sleeping.

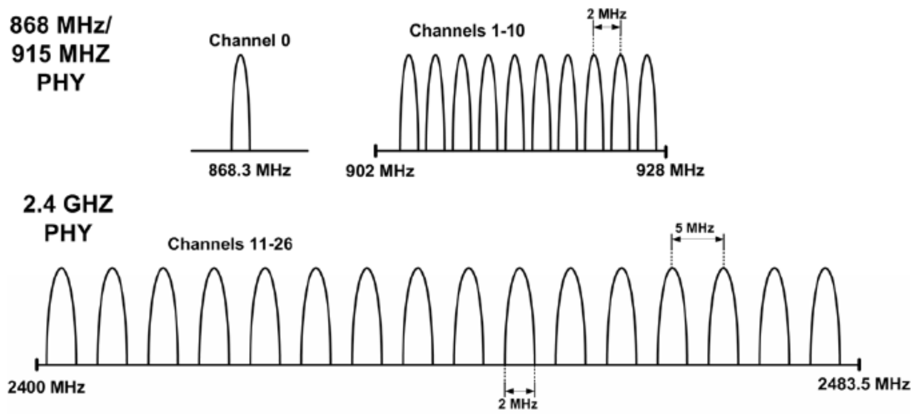


Figure 2.3: Operating frequency band.

PHY	Frequency band	Number of	Spreading parameters		Data parameters		
(MHz)	(MHz)	channels	Chip rate (kchip/s)	Modulation	Bit rate (kbit/s)	Symbol rate (ksymbol/s)	Symbols
868	868 - 868.6	1	300	BPSK	20	20	Binary
915	902 - 928	10	600	BPSK	40	40	Binary
2450	2400 - 2483.5	16	2000	O-QPSK	250	62.5	16-ary Orthogonal

Table 2.1: Frequency bands and data rates.

- Energy Detection (ED) within the current channel

  It is an estimation of the received signal power within the bandwidth of an IEEE 802.15.4 channel. This task does not make any signal identification or decoding on the channel. This measurement is typically used by the network layer as a part of channel selection algorithm or for the purpose of Clear Channel Assessment (CCA), to determine if the channel is busy or idle.

- Clear Channel Assessment (CCA)

  This operation is responsible for reporting the medium activity state: busy or idle. The CCA is performed in three operational modes:

  - *Energy Detection mode*: the CCA reports a busy medium if the detected energy is above the ED threshold.
  - *Carrier Sense mode*: the CCA reports a busy medium only is it detects a signal with the modulation and the spreading characteristics of IEEE 802.15.4 and which may be higher or lower than the ED threshold.
  - *Carrier Sense with Energy Detection mode*: this is a combination of the aforementioned techniques. The CCA reports that the medium is busy only if it detects

a signal with the modulation and the spreading characteristics of IEEE 802.15.4 and with energy above the ED threshold.

- Link Quality Indication (LQI)

  The LQI measurement characterizes the Strength/Quality of a received packet. It measures the quality of a received signal on a link. This measurement may be implemented using receiver ED, a signal to noise estimation or a combination of both techniques.

- Channel Frequency Selection

  The IEEE 802.15.4 defines 27 different wireless channels. A network can support only part of the channel set. Hence, the physical layer should be able to tune its transceiver into a specific channel request by a higher layer.

2.2.3 IEEE 802.15.4 Medium Access Control (MAC) layer

The MAC sub-layer provides an interface between the physical layer and the higher layer of protocol.

MAC protocol supports two operational modes that may be selected by the coordinator:

- **Beacon-enabled** mode

  Beacons are periodically generated by the coordinator to synchronize attached devices (receiving and decoding the beacon) and to identify the PAN. A beacon frame is the first part of a superframe, which contain data frames exchanged between nodes and between nodes and the PAN coordinator.

- **Non Beacon-enabled** mode

  In this mode, the devices can simply send their data by using unslotted CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance). There is no use of a superframe structure in this mode. Synchronization is performed by polling the coordinator for data.

Figure 2.4 presents a structure of the IEEE 802.15.4 operational modes.

2.2.3.1 Operational Modes

A. The Beacon-enabled mode

When the coordinator selects the beacon-enabled mode, it will use of a superframe structure to manage communication between devices (that are associated to that PAN). The format of the superframe is defined by the PAN coordinator and transmitted to other devices inside every beacon frame, which is broadcasted periodically by the PAN coordinator. The

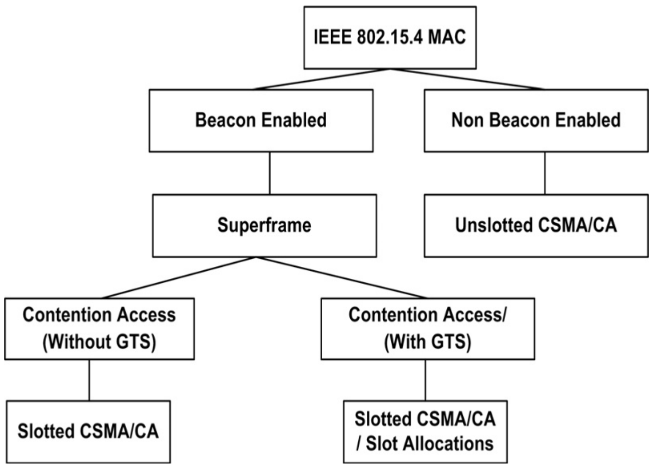

IEEE 802.15.4 MAC
Beacon Enabled
Non Beacon Enabled
Superframe
Unslotted CSMA/CA
Contention Access (Without GTS)
Contention Access/ (With GTS)
Slotted CSMA/CA
Slotted CSMA/CA / Slot Allocations

Figure 2.4: IEEE 802.15.4 operational modes

superframe is contained in a Beacon Interval, which is bounded by two consecutive beacon frames, and has an active period and may has an inactive period. The coordinator interacts with its PAN during the active period, and enters in a low power mode (sleep) during the inactive period. The structure of a superframe (Figure 2.5) is defined by two parameters:

- *macBeaconOrder* (BO): this attribute describes the interval at which the coordinator must transmit beacon frames. The Beacon Interval (BI) is defined as:

$$BI = aBaseSuperframeDuration * 2^{\mathrm{BO}} symbols, \; for \; 0 \leq BO \leq 14$$

- *macSuperframeOrder* (SO): this attributes describes the length of the active portion of the superframe, which includes the beacon frame. The Superframe Duration (SD) is defined as:

$$SD = aBaseSuperframeDuration * 2^{\mathrm{SO}} symbols, \; for \; 0 \leq SO \leq BO \leq 14$$

If $SO = BO \Rightarrow SD = BI$, then the superframe is always active. A PAN that wishes to use the superframe structure must set *macBeaconOrder* to a value between 0 and 14 and *macSuperframeOrder* to a value between 0 and the value of *macBeaconOrder*.
The active portion of each superframe is divided into 16 equally spaced slots of duration $2^{SO} * aBaseSlotDuration$. The attribute *aBaseSlotDuration* represents the number of symbols forming a superframe slot when the SO is equal to zero. One symbol is equal to four bits.

The active portion of the superframe structure is composed of three parts:

- **Beacon**: the beacon is transmitted without the use of CSMA at the start of slot 0. It contains the information on the addressing fields, the superframe specification, the

GTS fields, the pending address fields, etc. For more details on the beacon frame, refer to Appendix A.

- **Contention-Access Period** (CAP): the CAP starts immediately after the beacon frame and ends before the beginning of the CFP (if it exists), otherwise, the CAP ends at the end of the active part of the superframe. The minimum length of the CAP is fixed and it ensures that MAC commands can still be transferred to devices when *Guaranteed Time Slots* (GTSs, see below) are being used.

- **Contention-Free Period** (CFP): the CFP (if it exists) starts immediately after the end of the CAP and must complete before the start of the next beacon frame. The CFP consists in *Guaranteed Time Slots* (GTSs). It is a kind of resource reservation in WPANs. The GTSs may be only allocated by the PAN coordinator and must occupy contiguous slots. The CFP may therefore grow or shrink depending on the total length of all GTSs. According to the standard, the GTS is used only for communications between a PAN coordinator and a device.

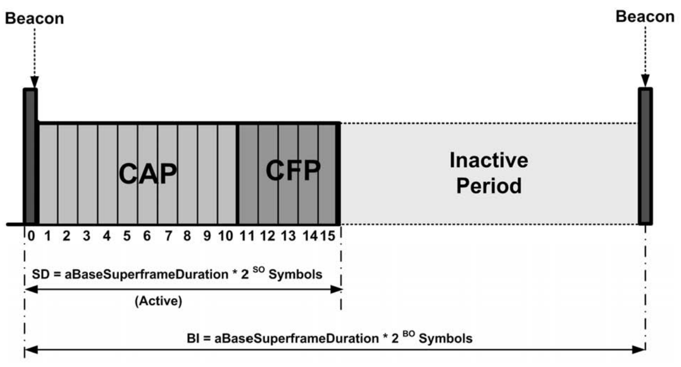

Beacon
Beacon
CAP
CFP
Inactive Period
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
SD = aBaseSuperframeDuration * 2^SO Symbols
(Active)
BI = aBaseSuperframeDuration * 2^BO Symbols

Figure 2.5: Structure of a Superframe

There are two superframe configurations:

1. **The superframe structure without GTSs**

   If communications are restricted to the CAP (defined in the beacon, issued by the PAN Coordinator) a device wishing to communicate must compete with other devices using a slotted CSMA/CA mechanism to access the channel. However, the acknowledgement frames and any data that immediately follows the acknowledgement of a data request command are transmitted without contention. A device that cannot complete its transmission before the end of the CAP, must defer its transmission until the CAP of the next superframe.

2. **The superframe structure with GTSs**

   If some guaranteed *Quality of Service* (QoS) is to be supported, then a CFP is defined. The PAN coordinator may allocate up to seven GTSs and each GTS may occupy more than one time slot. The transmissions in the CFP are contention-free and therefore do not use a CSMA/CA mechanism to access the channel. All contention-based communication must be finished before the start of the CFP, and a node transmitting a GTS must ensure that its transmission will be complete before the start of the next GTS (or the end of the CFP). The GTS management are discussed in Section 2.2.3.4.

In both configurations (CAP only or CAP/CFP), the superframe structure can have an inactive period during which the PAN coordinator does not interact with its PAN and may enter in a low power mode. The inactive periods enable the devices to save energy and thus extend network lifetime.

B. The Non Beacon-enabled Mode

When the PAN coordinator selects the non-beacon enabled mode, there are neither beacons nor superframes. According to the standard, the PAN will operate in a non beacon-enabled mode when the value of *macBeaconOrder* and *macSuperframeOrder* is equal to 15. Medium access control is provided by an unslotted CSMA/CA mechanism. All messages to be transmitted, with the exception of acknowledgment frames and any data frame that immediately follows the acknowledgment of a data request command, must be dispatched according to this mechanism.

2.2.3.2 The CSMA/CA mechanisms

The IEEE 802.15.4 defines two versions of the CSMA/CA mechanism:

- The **slotted CSMA/CA** version – used in the beacon-enabled mode.
- The **unslotted CSMA/CA** version – used in the non beacon-enabled mode.

In both cases, the CSMA/CA algorithm is based on *backoff periods*. Backoff period is the basic time unit of the MAC protocol and the access to the channel can only occur at the boundary of the backoff periods. In slotted CSMA/CA the backoff period boundaries must be aligned with the superframe slot boundaries while in unslotted CSMA/CA the backoff periods of one device are completely independent of the backoff periods of any other device in a PAN. The transmission of the current frame is started only if the remaining number of backoff periods in the current superframe is sufficient to handle both the frame and the subsequent acknowledgement transmissions. Otherwise, the transmission of the frame is deferred until the next superframe.

Figure 2.6 depicts a flowchart describing both versions of the CSMA/CA mechanism.

The CSMA/CA mechanism uses three variables to schedule the access to the medium:

- NB - the number of times the CSMA/CA algorithm was required to backoff while attempting the access to the current channel.
- CW - the contention windows length, which defines the number of backoff periods that need to be clear of channel activity before starting transmission. The CW variable is not used in the unslotted CSMA/CA.
- BE - the backoff exponent, which is related to how many backoff period a device must wait before attempting to assess the channel activity.

MAC sub-layer attributes:

- *macBattLifExt* - indication of whether battery life extension, by reduction of coordinator receiver operation time during the CAP, is enabled. The unslotted CSMA/CA does not support macBattLifExt mode.
- *macMinBE* - specifies the minimum value of the backoff exponent. When macMinBE is set to zero, the waiting delay is null and collision avoidance is disabled during the first iteration of the algorithm.
- *aMaxBE* - specifies the maximum value of the backoff exponent, a constant defined in the standard.
- *macMaxCSMABackoffs* - specifies the maximum number of backoffs the CSMA/CA algorithm will attempt before declaring a channel access failure status.

2.2.3.3 Frame Structure

The frame structures have been designed to keep the complexity to a minimum while at the same time making them sufficiently robust for transmission on a noisy channel. Each successive protocol layer adds to the structure with layer-specific headers and footers. The LR-WPAN defines four frame structures:

- a **beacon frame**, used by a coordinator to transmit beacons
- a **data frame**, used for all transfers of data
- an **acknowledgment frame**, used for confirming successful frame reception
- a **MAC command frame**, used for handling all MAC peer entity control transfers

The structure of each of the four frame types is presented in Appendix A.

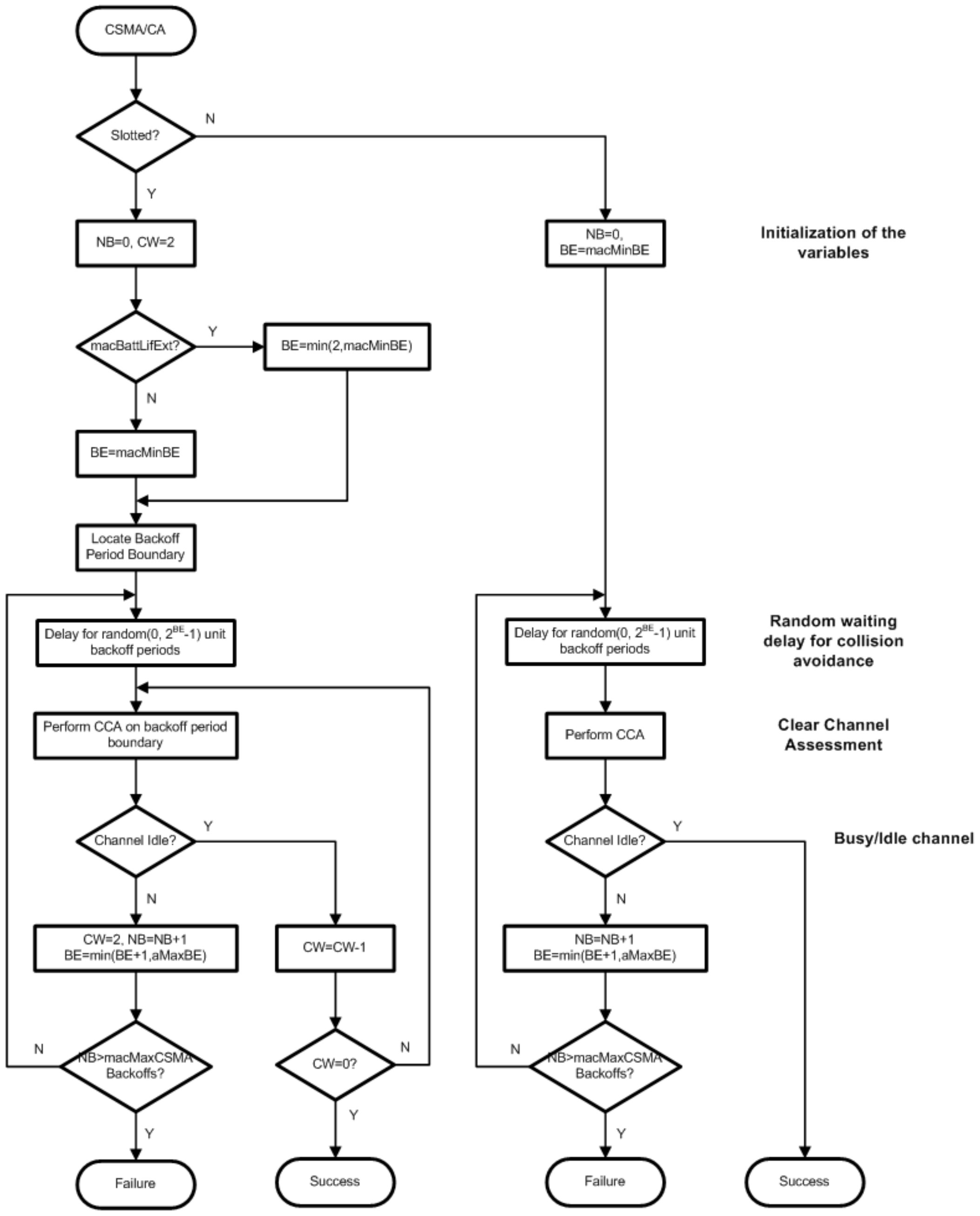

CSMA/CA
Slotted?
N
Y
NB=0, CW=2
NB=0, BE=macMinBE
Initialization of the variables
macBattLifExt?
Y
BE=min(2,macMinBE)
N
BE=macMinBE
Locate Backoff Period Boundary
Delay for random(0, 2^BE-1) unit backoff periods
Random waiting delay for collision avoidance
Perform CCA on backoff period boundary
Perform CCA
Clear Channel Assessment
Channel Idle?
Busy/Idle channel
CW=2, NB=NB+1 BE=min(BE+1,aMaxBE)
CW=CW-1
NB=NB+1 BE=min(BE+1,aMaxBE)
NB>macMaxCSMA Backoffs?
CW=0?
Failure
Success

Figure 2.6: The CSMA/CA Mechanism

2.2.3.4 GTS Management

The GTS (*Guaranteed Time Slot*) is dedicated (on the PAN) exclusively to a given device to whom allows to access the medium without contention in the CFP but it can also transmit during the CAP. The GTS must be allocated by a device before use and can be deallocated at any time at the discretion of the PAN coordinator or the device that originally requested the GTS. The PAN coordinator is the responsible for performing GTS management. If a device misses the beacon at the beginning of a superframe, it must not use its GTSs until it receives a subsequent beacon correctly. If synchronization with the PAN coordinator is lost due to the loss of the beacon, the device considers all of its GTSs deallocated.

A. GTS Allocation

A device that wants to allocate a GTS must send a GTS request command to its PAN coordinator indicating the GTS characteristics. The GTS request command frame is presented in Appendix A. Each device may request one transmit GTS (the direction of data flow is from the device to the coordinator) and/or one receive GTS (the direction is from the coordinator to the device).
The result of the GTS request is reported by the coordinator in the beacon frames using a GTS descriptor (presented in Appendix A) for each requesting device. The GTS descriptor remains in the beacon frame for *aGTSDescPersistenceTime* superframes, after which it should be removed automatically.

B. GTS deallocation

A device is instructed to request the deallocation of an existing GTS through sends a deallocation request to the PAN coordinator, which shall attempt to deallocate the GTS (its stored characteristics are reset). It does not add a GTS descriptor into its beacon frame to indicate the deallocation. If the GTS characteristics contained do not match the characteristics of a known GTS, the PAN coordinator shall ignore the request.
When a GTS is deallocated by the PAN coordinator, it adds a GTS descriptor into its beacon frame corresponding to the deallocated GTS, but with its starting slot set to 0, indicating that the GTS has been deallocated. The descriptor remains in the beacon frame for *aGTSDescPersistenceTime* superframes. On receipt of a beacon frame containing a corresponding GTS descriptor, the device shall immediately stop using the GTS.
The deallocation of a GTS may result in the superframe becoming fragmented. The PAN coordinator must ensure that any gaps occurring in the CFP, are removed to maximize the length of the CAP.

C. GTS expiration

The PAN coordinator must attempt to detect when a device has stopped using a GTS:

- for a transmit GTS, the PAN coordinator assumes that a device is no longer using its GTS if a data frame is not received in the GTS at least every $2 * n$ superframes
- for receive GTSs, the PAN coordinator assumes that a device is no longer using its GTS if an acknowledgment frame is not received at least every $2 * n$ superframes

The value of n is defined as follows:

$$\begin{aligned} n &= 2^{(8-macBeaconOrder)} \quad for\ 0 \leq macBeaconOrder \leq 8 \\ n &= 1 \quad for\ 9 \leq macBeaconOrder \leq 14 \end{aligned}$$

2.2.3.5 Extracting pending data from a coordinator

This communication mechanism is called *indirect transmission*, where a given device polls pending data from its coordinator.
In a beacon-enabled mode, a device is aware whether it has any frame pending by examining the contents of the received beacon frames. If its address is contained in the *Pending Address* field of the beacon frame, then the device sends a data request command to the coordinator in the CAP. The sending of the pending data is based on CSMA/CA.

2.2.3.6 Channel scan procedures

A PAN can be created by an FFD only after performing an Active channel or an Energy Detection (ED) channel scan and choosing an appropriate PAN identifier. Channel scan procedures are explained in [3, 5].

2.2.3.7 Association and Disassociation

A. Association

When a device wants to join an existing network without creating a new PAN, it must be associated with an existing PAN. The association process starts with an channel scan. The results of the scan are then used to choose a suitable PAN characterized by its physical channel, identifier, extended and short addresses. If the coordinator successfully associates the device by allocating a new short address than generates an *association response command* containing the new address. The association response command is sent to the device using *indirect transmission.*

B. Disassociation

The disassociation process may be initiated by either the coordinator or the device itself.

- **Coordinator-initiated disassociation:** the coordinator sends the *disassociation notification* command to the device using *indirect transmission*. All the references to the device are removed by the PAN coordinator.
- **Device-initiated disassociation:** the device sends a *disassociation notification* command to the coordinator. All the references to the PAN must be removed by the device.

2.2.3.8 Orphaned device

A device may conclude that it becomes an orphan device (out of the range of its last PAN) if a predetermined number of transmission attempts have failed. The device than triggers the *orphaned device realignment* procedure or resets the MAC sub-layer and perform the association procedure. The *orphaned device realignment* consists on doing an orphan channel scan (see [3, 5]).

2.3 The ZigBee

The IEEE 802.15.4 standard defines the physical (PHY) layer and the Medium Access Control (MAC) sub-layer. The ZigBee Alliance builds on this foundation by providing the network (NWK) layer and the application framework for the application layer. The application framework (AF) is comprised of the application support sub-layer (APS), the ZigBee device objects (ZDO) containing the ZDO management plane, and the manufacturer-defined application objects.

2.3.1 The ZigBee stack architecture

The ZigBee stack architecture is depicted in Figure 2.7. Each layer performs a specific set of services and capabilities for the layer above: a data entity provides a data transmission service and a management entity provides all other services. Each service entity exposes an interface to the upper layer through a service access point (SAP), and each SAP supports a number of service primitives to achieve the required functionality.

The responsibilities of the ZigBee **NWK layer** shall include mechanisms used to:

- Starting a network - the ability to successfully establish a new network
- Joining and leaving a network - the ability to gain membership (join) or relinquish membership (leave) a network

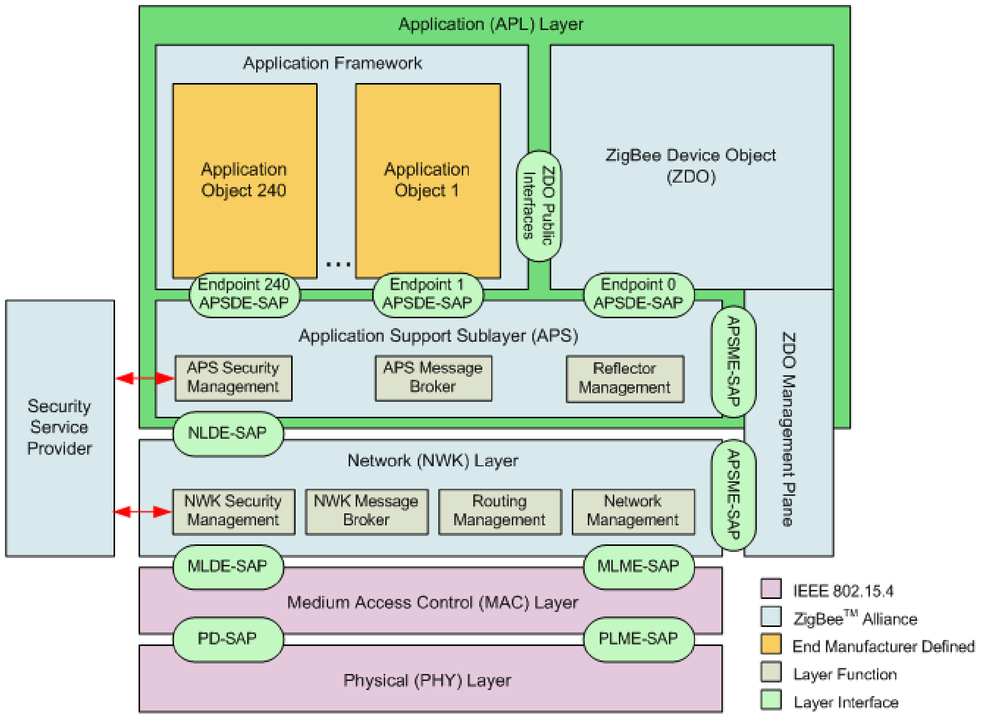


Figure 2.7: Outline of the ZigBee Stack Architecture

- Configuring a new device - the ability to sufficiently configure the stack for operation as required
- Addressing - the ability of a ZigBee Coordinator (see bellow) to assign addresses to devices joining the network
- Synchronization within a network - the ability for a device to achieve synchronization with another device either through tracking beacons or by polling
- Security - applying security to outgoing frames and removing security to terminating frames
- Routing - routing frames to their intended destinations

The ZigBee specification defined **three types of devices** according to the IEEE 802.15.4 standard (see Section 2.2.1):

- *ZigBee Coordinator (ZC)* (FFD)
  - one required for each ZB network
  - initiates network formation and is responsible for the inner workings of the network

- *ZigBee Router (ZR)* (FFD)
  - is used as mediator (multihop routing of messages) for the coordinator in the PAN - allowing the network to expand beyond the radio range of the coordinator
  - acts as a local coordinator for end devices joining the PAN and must implement most of the coordinator capabilities
- *ZigBee End Device (ZED)* (RFD or FFD)
  - does not allow association or routing
  - enables very low cost solutions

The responsibilities of the **APS sub-layer** include:

- Maintaining tables for binding, defined as the ability to match two devices together based on their services and their needs
- Forwarding messages between bound devices
- Discovering devices on the network and determining which application services they provide
- Group address definition, removal and filtering of group addressed messages
- Address mapping from 64 bit IEEE addresses to and from 16 bit NWK addresses
- Fragmentation, reassembly and reliable data transport

The responsibilities of the **ZDO** include:

- Defining the role of the device within the network (e.g. ZigBee Coordinator or end device)
- Initiating and/or responding to binding requests
- Establishing a secure relationship between network devices selecting one of ZigBee's security methods such as public key, symmetric key, etc.

The **Application Framework** in ZigBee is the environment in which application objects are hosted on ZigBee devices. The application objects perform the following functions through the ZDO public interfaces:

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

Inside the application framework, the **manufacturer-defined application objects** implement the actual applications according to the ZigBee-defined application descriptions. The application objects send and receive data through the APSDE-SAP. Up to 240 distinct application objects can be defined, each interfacing on an endpoint indexed from 1 to 240. An endpoint number can be used to identify individual physical devices that are described in terms of the data attributes that they contain. Two additional endpoints are defined for APSDE-SAP usage: endpoint 0 is reserved for the data interface to the ZDO (device management, in other words, used to address the descriptors in the node) and endpoint 255 is reserved for the data interface function to broadcast data to all application objects. Endpoints 241-254 are reserved for future use.

The **Application profiles** are agreements for messages, message formats and processing actions that enable applications to create an interoperable, distributed application between applications that reside on separate devices. These application profiles enable applications to send commands, request data and process commands and requests.

ZigBee vendors develop application profiles to provide solutions to specific technology needs. Application profiles are simultaneously a means of unifying interoperable technical solutions within the ZigBee standard, as well as focusing usability efforts within a given marketing area. ZigBee publishes a set of public profiles and product vendors may add additional features to them.

A complete description of the ZigBee specification can be found in [11].

Chapter 3

Description of used Software and Hardware

The following is a brief introduction to TinyOS [16] and nesC [18]. Furthermore, the Moteiv Tmote Sky [21] sensor module is described.

3.1 TinyOS

TinyOS is a micro-threaded, event-driven open source operating environment implemented using nesC and designed to support the concurrency operations required by embedded networked sensors with minimal hardware requirements. The TinyOS minimizes code size, this means that only the necessary features of the operating system and application are included. TinyOS was initially developed by researchers at the University of California, Berkeley [15] in cooperation with Intel Research and now is actively supported by a large community of users. For this thesis purpose is used the TinyOS version 1.1.15, running under Cygwin (Linux-like environment) on Windows platform, for compatibility reasons. The implementation of the IEEE 802.15.4/ZigBee protocols, so-called open-ZB stack, is developed in this version of TinyOS. Actual version of TinyOS is 2.0.2 and the second series is not compatible with the first series. The source for TinyOS are available on SourceForge [17]. Install instructions for getting TinyOS running under Cygwin are enclosed in Appendix C. See the TinyOS [16] on Linux page if you are interesting in running TinyOS on Linux.

An application implemented in TinyOS is based on a number of *components*, e.g. Leds, Timers, etc. These components are reusable from one application to another. Applications are formed by components, which are *wiring* together to suite the task at hand. Changing whether the device communicates using the wireless channel or the serial port can be changed by simply changing the communication component which the application connects. Components can be abstract concepts (consisting of many diferent components) or a low

level wrapper for a hardware component (e.g. UART).
The implementation of components is based on *tasks*, *commands* and *events*.
Tasks should generally be to perform long processing operations, such as long running computations or background data processing. Tasks are *posted* to a task queue, after which control is immediately returned to the posting component. The TinyOS task scheduler is based on simple FIFO task execution. When no tasks are pending to be executed, the scheduler switch the processor to sleep (main TinyOS ideology - execute jobs quickly and go back to sleep to save power), until the next interrupt is received. Tasks run to completion and cannot preempt each other. The use of tasks causes TinyOS to have only non-blocking operations.
Commands are *called* to execute a given functionality in another component.
Events also run to completion and can preempt the execution of tasks or other events. Events signify either completion of a split-phase operation (see below) or an event from the environment (e.g. time passing). Components wrapping hardware *signal* events in response to hardware interrupts. These events are marked with the `async` keyword. There exist a technique of *split-phase* operation (operation request and completion are separate functions), if long-latency operations are used. Essentially, commands are used to initiate the requested action, e.g. *component.*`request`, posting a task and returning immediately. Events are then signaled in response to the completion of the split-phase operation, e.g. typically using *component.*`requestDone`. These kinds of events do not preempt as those caused by hardware interrupts.

3.2 nesC

nesC (network-embedded-systems-C) is the programming language of TinyOS. It is an extension of C language that uses a custom compiler „nesC". For this thesis purpose is used the nesC version 1.2.7. The source for nesC are available on SourceForge [19].

nesC uses two concepts to represent components: *modules* and *configurations*. Modules contain the code for a single component whereas a configuration is used to *wire* components together. An application can use a configuration wiring one or wiring more components together as a component in itself. A top-level configuration wires all components in the application together.
A module implements one or more *interfaces*. Interfaces in nesC are bidirectional - they contain and make accessible commands and events (both of which are essentially functions) provided by a component. A component provides and uses interfaces. Configurations wire modules using a given interface to a component providing an implementation of this interface. The concurrency model of nesC allows for static compile time detection of race conditions. These can be handled using `atomic` sections to turn off hardware interrupts in a block of code and update the shared state, or by converting the conflicting code into tasks. The

static analysis prohibits some of features used in regular C language, especially function pointers and dynamic memory allocation (i.e. `malloc`).
TinyOS consists of many modules. These are compiled with the application as needed.

3.3 Moteiv Tmote Sky

As hardware platform for wireless communication has been chosen the Tmote Sky sensor module, because the Wireless Sensor Networks Group at Department of Control Engineering have these nine operational devices that are available.

Tmote Sky is an ultra low power IEEE 802.15.4 compliant wireless sensor module for use in sensor networks and monitoring applications. Tmote Sky, just alternative name for Telos Revision B design, is the latest sensor node platform available from Moteiv Corporation. Tmote Sky is replacement for Moteiv's successful Telos design (Revision A), whose open source hardware specification is Tmote Sky based. Both revision are supported by TinyOS/nesC.

3.3.1 Key Features

- 250kbps 2.4GHz IEEE 802.15.4 CC2420 Chipcon Wireless Transceiver
- 8MHz Texas Instruments MSP430F1611 microcontroller (10kB RAM, 48kB Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption and fast wakeup from sleep (<6us)
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector

3.3.2 Module Description

Tmote Sky may be powered by two AA batteries (operating range of 2.1 to 3.6V DC) or if it is plugged into the USB port for programming or communication, it will receive power from the host computer (operating voltage is 3V).
Tmote Sky features the Chipcon CC2420 IEEE 802.15.4 compliant radio for wireless communications, providing the PHY, some MAC functions and programmable output power. Features and usage of the CC2420 is available in Chipcon's datasheet [27]. CC2420 is controlled by the 16-bit RISC TI MSP430 F1611 8Mhz microcontroller TI with 48kB ROM and 10kB RAM, through the SPI (Serial Port Interface) port and a series of digital I/O

lines and interrupts. There are UART (Universal Asynchronous Receiver/Transmitter), I2C (Inter IC), an external 32768Hz watch crystal and 8 external ADC ports (12-bit) provided which can be used to collect external signals. The features of the MSP430 F1611 are presented in detail in the Texas Instruments Datasheet and MSP430x1xx Family User's Guide [24].

A 1MB STM25P80 Flash (1MB or 1024kB) on the Tmote Sky is another extremely useful feature. This can be conveniently used to store data, code and other information permanently on the tmote until the flash gets formatted. The flash shares SPI communication lines with the CC2420 radio and the external SPI pins (on external expansion connector). This means that there is a need for careful bus arbitration while using any of them simultaneously.

Tmote Sky uses a USB controller from FTDI to communicate with the host computer. In order to communicate with the mote, the FTDI drivers must be installed on the host. They may be downloaded from FTDI's website [25]. Tmote Sky appears as a COM port in Windows device manager (or as a device in /dev in Linux). Multiple Tmote Sky motes may be connected to a single computer's USB ports at the same time. Each mote will receive a different COM port identifier. Tmote communicates with the host PC through USART1 on the TI MSP430.

The Tmote Sky module provides 3 leds as user interface and slot to attach an external antenna (range of 125m, internal antenna range of 50m). It is programmed through the onboard USB connector. A modified version of the MSP430 Bootstrap Loader, msp430-bsl [26], programs the microcontroller's flash.

Tmote Sky has two expansion connectors (depicted in Figures 3.1, 3.2), 10-pin and 6-pin, that may configured so that additional devices (analog sensors, digital peripherals) may be controlled by the Tmote Sky and can provide power to the expansion module. The 10-pin connector is the primary connector. An additional 6-pin header provides access to the exclusive features of Tmote Sky.

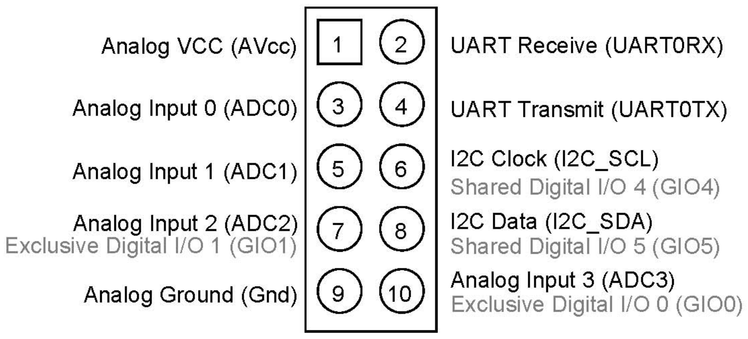


Figure 3.1: Functionality of the 10-pin expansion connector (U2)
Alternative pin uses are shown in gray

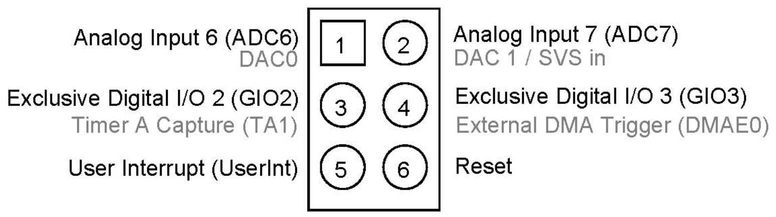


Figure 3.2: Functionality of the 6-pin expansion connector (U28)
Alternative pin uses are shown in gray

3.3.3 Block Diagram

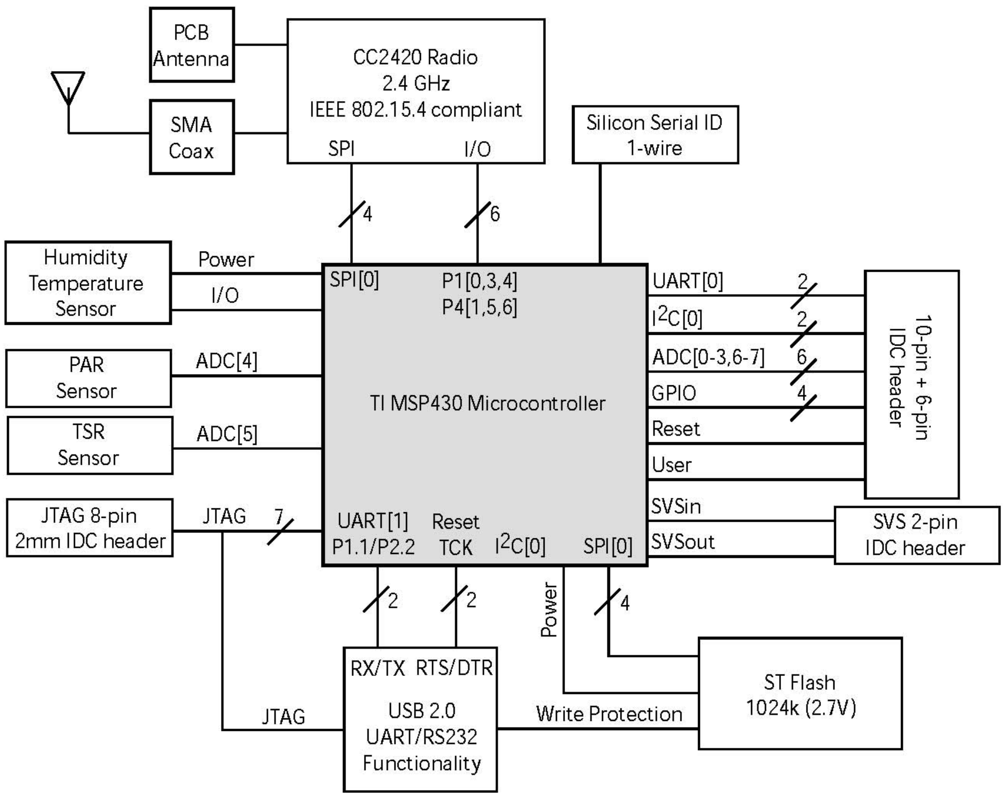


Figure 3.3: Functional Block Diagram of the Tmote Sky module, its components, and buses

Chapter 4

Open-ZB stack and its porting to TelosB platform

4.1 Introduction

The open-ZB stack is the open-source implementation of the IEEE 802.15.4/ZigBee protocols in TinyOS 1.x/nesC, which is providing as the toolset. Protocol stack is being developed within the IPP-HURRAY Research Group (the Polytechnical Institute of Porto, Portugal), by André Cunha, Mário Alves and Anis Koubâa. The protocol stack was primarily implemented for CrossBow MICAz platform.

When we would like to use the Tmote Sky sensor modules for creating an wireless sensor network, we started to porting and finally collaborated with the IPP-HURRAY Research Group (namely André Cunha) in porting this protocol stack to the TelosB platform (CrossBow TelosB or Moteiv Tmote Sky modules). The entire implementation and porting was completed and the version of open-ZB stack that includes the support for the TelosB platform motes is version 1.2.
Actual version of protocol stack already includes the implementation of the ZigBee Network Layer (ZBNL) on top of implementation IEEE 802.15.4 protocol. This implementation enables the cluster-tree network topology with a mechanism for beacon scheduling in order to enable an efficient use of synchronized cluster-tree networks. There is no need to use the ZBNL for the purpose of this diploma thesis, therefore, we used the open-ZB stack version 1.2.

Version 1.2 of the implementation supports the following IEEE 802.15.4 functionalities:

- CSMA/CA algorithm - slotted version
- GTS Mechanism

- Indirect transmission mechanism
- Direct / Indirect / GTS Data Transmission
- Beacon Management
- Frame construction - Short Addressing Fields only and extended addressing fields in the association request
- Association / Disassociation Mechanism
- MAC PIB Management
- Frame Reception Conditions
- ED and PASSIVE channel scan

Functionalities that are not implemented or tested in the version 1.2 yet:

- Unslotted version CSMA/CA - Implemented but not fully tested
- Extended Address Fields of the Frames
- IntraPAN Address Fields of the Frames
- Active and Orphan channel Scan
- Orphan Devices
- Frame Reception Conditions (Verify Conditions)
- Security - Out of the scope of this implementation

The open-ZB stack v1.2 is supported by two hardware models, the MICAz and the TelosB motes.

Comparison of general features these two hardware models is presented in Table 4.1.

The MICAz mote needs to be programmed using an interface board. The TelosB motes do not need any programmer interface because they already have an USB port that is used to upload programs. More information about the TelosB platform (Tmote Sky) was presented in Chapter 3.

MICAz	TelosB
ATMEL ATmega128L 8-bit μcontroller	TI MSP430 16-bit μcontroller
CC2420 RF transceiver	CC2420 RF transceiver
128 KB of program memory	48 KB of program memory
4 KB of EEPROM	10 KB of EEPROM
Supports several sensor boards	Includes a temperature and light sensor
UART communication port	UART communication port (USB converter)

Table 4.1: The supported features of MICAz and TelosB platform.

4.2 Software Architecture

Figure 4.1 presents implementation stack architecture layers.

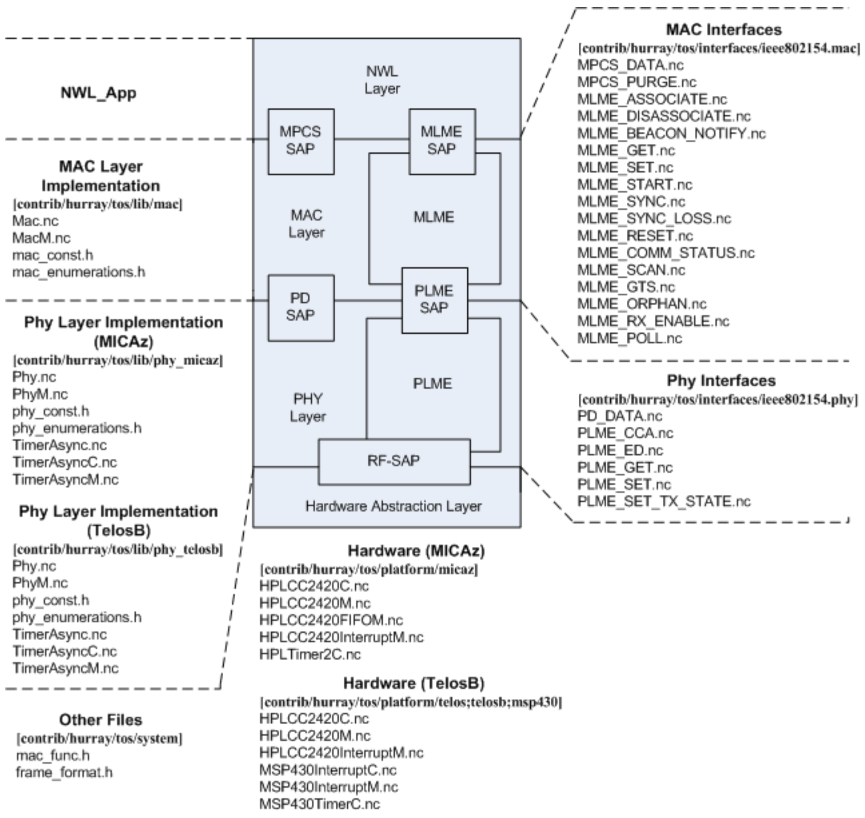


Figure 4.1: Open-ZB Protocol Stack Architecture.

The implementation is organized so that each main module (PhyM and MacM) implements a layer functions. Each of these modules makes use of auxiliary files used for some generic function implementations, constants declarations, enumerations and data structure definitions. IPP-HURRAY Research Group have developed an auxiliary module, the TimerAsync, for the implementation of an asynchronous timer based on the hardware clock. For the synchronous timers, used in non time critical operations, is used the TimerC already provided by TinyOS.

The implementation uses files/components already provided in TinyOS namely the hardware components. All the stack files are located in the `contrib/hurray` folder. The directory structure is similar to the TinyOS root folder. All the files created for implementation and their respective location are depicted in Figure 4.1. A more detailed description is introducted in [6].

The interface files are used to wiring the stack components and represent a service access point (SAP). The RF-SAP comprehends the interface with the physical radio via the RF firmware of the CC2420 and hardware components in TinyOS. The PD-SAP (Phy Data service) comprehends the interface to exchange data packets between MAC and PHY layers. The PLME-SAP (Physical Layer Management Entity) comprehends the interfaces between MAC and PHY layers used for exchanging management information. The PLME-SAP contains an PHY PAN Information Base (PHY PIB), which is a database of physical layer managed objects (e.g. current channels, transmit power) and the PLME-SAP interfaces are used by MAC layer to manage this information.

The MCPS-SAP (MAC Common Part Sublayer) comprehends the MSDU (MAC Service Data Unit) data transfer between the MAC layer and the upper layer. The MLME-SAP (MAC Layer Management Entity) contains an MAC PAN Information Base (MAC PIB), which is a database of MAC layer managed objects (e.g. beacon order, superframe order, short address, PAN identifier, GTS permit options) and the MLME-SAP (MAC Layer Management Entity) interfaces are used by the MAC upper layer to manage this information.

4.3 Comparison between MICAz and TelosB implementation

Figure 4.2 presents the most important component relations diagram of open-ZB stack implementation in TinyOS for both platforms - MICAz and TelosB. The most relevant TinyOS hardware components are highlighted in white. The components of open-ZB stack are highlighted in gray.

The important aspect of IEEE 802.15.4 standard is the synchronization and with it related to the main modification. To accomplish a precise synchronization a important timer component was developed, with an asynchronous behaviour regarding the code execution, based on the hardware clock. The reason is related to the TinyOS management of hardware timer provided by both platforms, which does not allow having the exact values in millisecond of the beacon interval, superframe and time slots as specified by the protocol. Also due to the difference between the hardware timers used by the two platforms is not possible to achieve the same timer granularities. Therefore, the TimerAsync component has two different implementations. These implementation for MICAz and TelosB platform are carefully described in [6].

The physical layer represented by PhyM module is need to be wired with the hardware specific components, that are differentiated depending of the used hardware platform. The interference in the MacM module (implements MAC layer) have a cosmetic character. An graphical representation of the carried modifications is presented in pictures in [7]. In the meantime, a few of comments under pictures are added, but an basic idea of concept of the hardware components wiring should be uderstand. PHY auxiliary files (contain the protocol constants definition and enumeration values used in the PHY) and MAC auxiliary

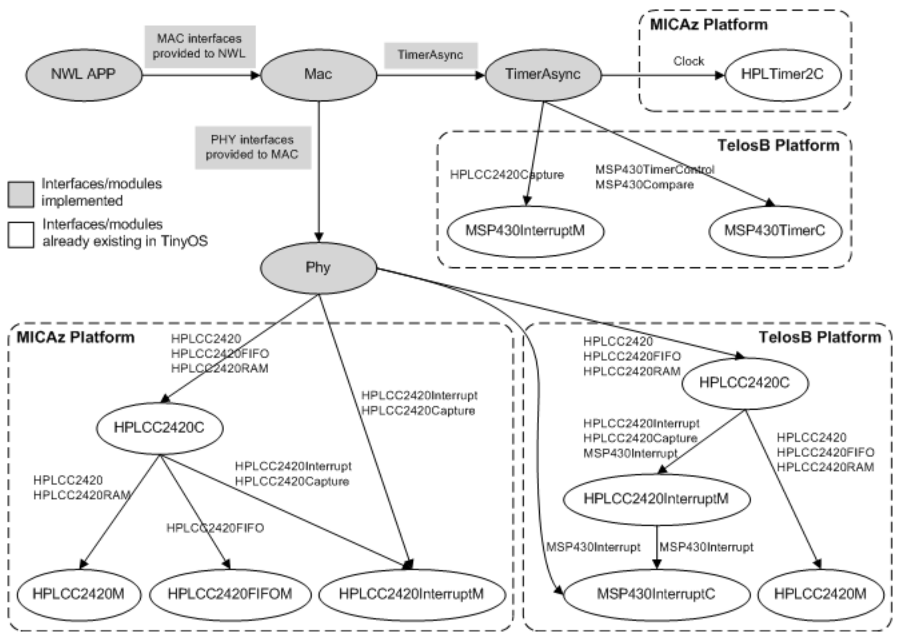


Figure 4.2: Open-ZB stack implementation diagram in TinyOS/nesC for MICAz and TelosB platforms.

files (contain data structures definition, protocol constants definition and enumeration values used in the MAC) are no need to change.

4.4 Main Modifications of the open-ZB stack

We started to porting from last version 1.1 of the implementation IEEE 802.15.4/ZigBee protocol. In this section are presented some of main modifications.

4.4.1 Phy Configuration File

Phy configuration is used to wire the PhyM module to other components. PhyM module is need to be wired with the hardware specific components of TelosB platform. The difference is we use the MSP430InterruptC component for FIFOP interrupt (active when number of bytes in FIFO of CC2420 radio exceeds threshold) instead of HPLCC2420Interrupt component, that is used for MICAz platform.

```
configuration Phy { }
implementation { components MSP430InterruptC;
        PhyM.FIFOPInterrupt -> MSP430InterruptC.Port10; }
```

Take a look at the schematic given in the Tmote Sky datasheet [23] to find out which pin of CC2420 radio is connected to a certain port of microprocessor. „Port10" means port 1.0 (PKT_INT) of MCU. This port is connected to FIFOP pin of CC2420.

4.4.2 PhyM Module File

PhyM module actually provides the implementation of PHY layer. When we use the MSP430InterruptC then we must implement MSP430Interrupt interface, that MSP430 InterruptC component provides.

```
module PhyM {
  uses {
    interface MSP430Interrupt as FIFOPInterrupt;
    }
}
implementation {

//physical events CC2420
async event void FIFOPInterrupt.fired() {
call FIFOPInterrupt.enable(); }

//enable an edge interrupt on the CC2420 FIFOP pin
void enableFIFOP(){
  atomic { call FIFOPInterrupt.disable();
           call FIFOPInterrupt.clear();
           call FIFOPInterrupt.edge(0);
           call FIFOPInterrupt.enable(); } }

//disables CC2420 FIFOP interrupts
void disableFIFOP(){
  atomic { call FIFOPInterrupt.disable();
           call FIFOPInterrupt.clear();   } }
}
```

Functions enableFIFOP() and disableFIFOP() are exactly assume from Telos HPLCC2420 InterruptM component. The FIFOPInterrupt.fired() event is the same as in MICAz PhyM module, it is only completed by reenable FIFOP interrupt.

4.4.3 TimerAsync Components

The timer component for MICAz is based on the hardware clock timer configuration defined in two constants Scale and Interval. Scale defines the scale division of the AVR microprocessor and Interval defines the number of clock ticks per clock firing. The hardware timer for the TelosB platform is based on a 32768 Hz clock and fires at approximately $30.5\mu s$. In comparison with the MICAz timer this does not allow the set of a scale or interval parameters, instead this is a continuous timer that count from 0 to 0xFFFF and when it overflows it triggers an interrupt and starts again from zero. The only allowed parameterization is the number of overflow count before the issuing of the interrupt. The solution for the implementation is described in [6].

4.4.4 Other Changes

In MSP430 microprocesor (used TelosB platform) may come an internal error of unsuported relocation. The reason is that in this 16-bit microcontroller we have to be careful how we declare structures. We cannot declare an 8-bit integer followed by an 16-bit integer, because internally the MCU will alocate 32-bits of memory for that with an 8-bit space between the first declaration and the second. Therefore we place an 8-bit integers always at the end of structure definition.

In MacM module, that actually provides the implementation of MAC layer, we have to use the powf() function for MSP430 MCU instead of the pow() function, that is only for the AVR MCU. The function returns the value of x to the exponent y.

Chapter 5

Expansion Modules with Accelerometers

As a simple ActiGraph devices are designed expansion modules (as peripherals) with an appropriate accelerometers. As analog accelerometer has been chosen the MMA7260Q (eventually MMA7261Q) by Freescale Semiconductor and as digital accelerometer the LIS3LV02DQ by ST Microelectronics. Both accelerometers have advantageous features available in this project.

5.1 Expansion Module with Analog Accelerometer

MMA7260Q is ±1.5g - 6g and MMA7261Q is ±2.5g - 10g three axis low-g low-cost capacitive surface-micromachined integrated-circuit accelerometer, whose main features are:

- Selectable Sensitivity (MMA7260Q: 1.5g/2g/4g/6g, MMA7261Q: 2.5g/3.3g/6.7g/10g)
- High Sensitivity (MMA7260Q: 200mV/g @6g, MMA7261Q: 120mV/g @10g)
- Low Voltage Operation Range: 2.2V - 3.6V
- Low Current Consumption: $500\mu A$, Sleep Mode: $3\mu A$
- Control Timing:
  - Power-Up Response Time: 1ms - 2ms
  - Enable Response Time: 0.5ms - 2ms
- Internal Sampling Frequency: 11kHz
- Integral Signal Conditioning with Low Pass Filter and temperature compensation

Special features:

- g-Select

  The g-Select feature allows for the selection among 4 sensitivities present in the device. Depending on the logic input placed on pins 1 and 2, the device internal gain will be changed allowing it to function with a selected sensitivity that can be changed at anytime during the operation of the product (Table 5.1).

g-Select2 [MSb]	g-Select1 [LSb]	g-Range [g]	Sensitivity [mV/g]
0	0	1.5/2.5	800/480
0	1	2/3.3	600/360
1	0	4/6.7	300/180
1	1	6/10	200/120

Table 5.1: g-Select pin Descriptions for MMA7260Q/MMA7261Q

- Sleep Mode

  The accelerometer provides a Sleep Mode that makes is ideal for handheld battery powered electronics. A low input signal on pin 12 (Sleep Mode) will place the device in this mode, the device outputs are turned off and reduce the current to $3\mu A$ typ. By placing a high input signal on pin 12, the device will resume to normal mode of operation.

5.1.1 Design of Circuit Scheme

The entire design is created in Cadence OrCAD Design Tools 15.2 according to the MMA7260Q datasheet. The electronical scheme is presented in Figure 5.1.

We used more capacitors on VCC to decouple the power source. In PCB (printed circuit board) design is used the very low drop voltage (0.2V) regulator LE33CZ [30] with output voltage 3.3V from ST Microelectronics for the purpose of using the designed expansion board with accelerometer to other applications. The PCB design includes an RC filter on the outputs of the accelerometer to minimize clock noise. The circuit board is connected to the Tmote Sky sensor module by the used 10-pin connector. A pin layout is designed according to the functionality of the Tmote Sky 10-pin expansion connector, see Chapter 3. That means, all of the X-axis, Y-axis and Z-axis are connected through analog input pins (ADC0 - ADC2). Sleep mode and both of g-Select are connected through third analog input remaining (ADC3) and UART0RX/UART0TX. These three pins will be always set only to the high or low state according to using Sleep mode functionality and selecting required sensitivity.

The Tmote Sky module with the manufactured designed expansion circuit board is presented in Figure 5.2.

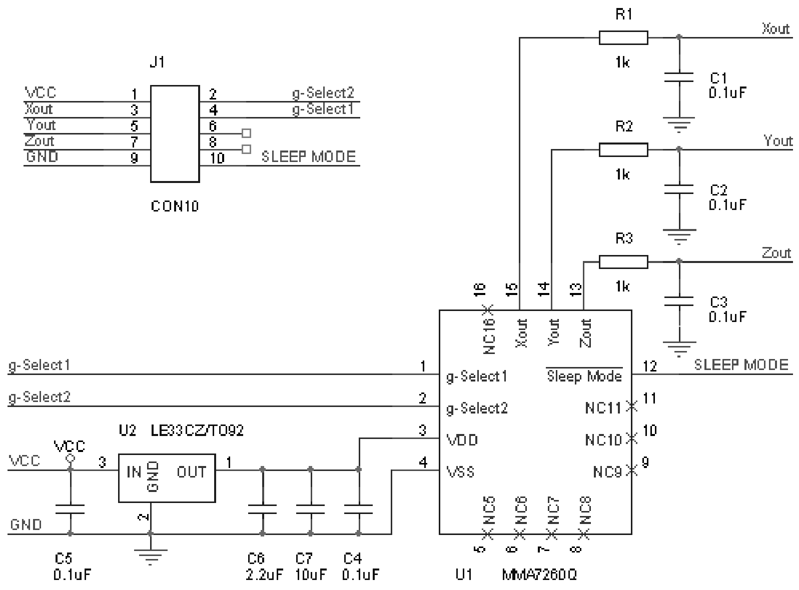


Figure 5.1: Circuit scheme with MMA7260Q.

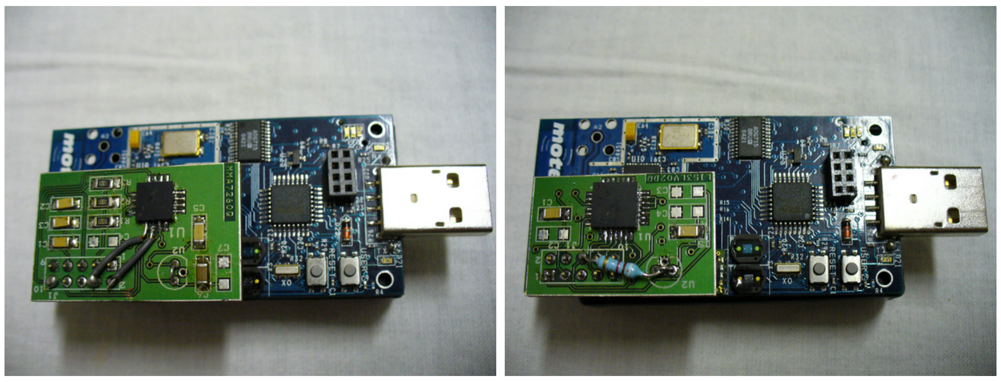


Figure 5.2: Tmote Sky with manufactured expansion module with MMA7260Q analog and LIS3LV02DQ digital accelerometer.

5.2 Expansion Module with Digital Accelerometer

LIS3LV02DQ is ±2g/±6g three axes digital output high-performance low-power linear accelerometer that includes a sensing element and an IC interface able to take the information from the sensing element and to provide the measured acceleration signals to the external world through an I^2C/SPI serial interface.

When we want to design an expansion module (as peripheral) for the Tmote Sky sensor module, we have to use an I^2C serial interface, because the primary Tmote Sky 10-pin connector provides only two I^2C pins of I^2C bus, see Chapter 3.

Main features of the accelerometer:

- selectable full scale of ±2g, ±6g
- measuring acceleration over a bandwidth of 640Hz for all axes
- 2.16V to 3.6V single supply operation
- I^2C/SPI digital output interfaces
- programmable 12 or 16 bit data representation
- interrupt activated by motion and programmable interrupt threshold
- embedded Self Test that allows to test the mechanical and electric part of the sensor

5.2.1 I^2C Serial Interface

The registers embedded inside the LIS3LV02DQ may be accessed through both the I^2C and SPI serial interfaces, but we have to use the I^2C to communicate with the Tmote Sky sensor module, therefore, we have to do a short I^2C operation analyze. To select the I^2C interface, CS line must be tied high (i.e connected to Vdd).

The LIS3LV02DQ I^2C is a bus slave, it means the device addressed by the master which initiates/terminates a transfer and generates clock signals. The I^2C is employed to write the data into the registers whose content can also be read back.
There are two signals associated with the I^2C bus:

- the Serial Clock Line (SCL)
- the Serial DAta line (SDA) - bidirectional line used for sending and receiving the data to/from the interface

Both the lines are connected to Vdd through a pull-up resistor embedded inside the LIS3LV02DQ. When the bus is free both the lines are tied high. The I^2C interface is compliant with Fast Mode (400 kHz) I^2C standards as well as the Normal Mode.

5.2.1.1 I^{2}C Operation

The transaction on the bus is always started by Master through a START (ST) signal. The Master continues by sending a unique 7-bit slave device address, with the most significant bit (MSb) first. The eighth bit is a Read/Write bit and tells whether the Master is receiving („1") data from the slave or transmitting („0") data to the slave. When an address is sent, each device in the system compares the first seven bits after a start condition with its address. If they match, the device considers itself addressed by the Master. The Slave ADdress (SAD) associated to the LIS3LV02DQ is **0011101**b.
Data transfer with acknowledge is mandatory. A receiver which has been addressed is obliged to generate an acknowledge after each byte of data has been received. After the start condition a slave address is sent, once a slave acknowledge (SAK) has been returned, a 8-bit sub-address will be transmitted - the 7 LSb represent the actual register address while the MSb enables address auto increment. If the MSb of the SUB field is 1, the SUB (register address) will be automatically incremented to allow multiple data Read/Write.
Data are transmitted in byte format (DATA) - each data transfer contains 8 bits. The number of bytes transferred per transfer is unlimited. Data is transferred with the Most Significant bit (MSb) first. Each data transfer must be terminated by the generation of a STOP (SP) condition.
In order to read multiple bytes, it is necessary to assert the most significant bit of the subaddress field - SUB(7) must be equal to „1" while SUB(6-0) represents the address of first register to read. The Table 5.2 presented communication format, MAK is Master Acknowledge and NMAK is No Master Acknowledge.

The perfect description of I^{2}C communication is available in The I^{2}C-Bus Specification by Philips Semiconductor [31].

5.2.2 Register mapping

The device contains a set of registers which are used to control its behavior and to retrieve acceleration data. The Table 5.3 given below provides a listing of the most important 8 bit registers embedded in the device and the related address.

Register description is available in LIS3LV02DQ datasheet [32, page 25].

5.2.3 Design of Circuit Scheme

The entire design is created in Cadence OrCAD Design Tools 15.2 according to the LIS3LV02DQ datasheet. The electronical scheme is presented in Figure 5.1.

As well as in PCB expansion module design with MMA7260Q analog accelerometer we used more capacitors on VCC to decouple the power source and the very low drop voltage (0.2V) regulator LE33CZ [30] with output voltage 3.3V from ST Microelectronics for the purpose of using the designed expansion board with accelerometer to other applications.

Master is writing one byte to Slave:

Master	ST	SAD+W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Master is writing multiple bytes to Slave:

Master	ST	SAD+W		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

Master is receiving (reading) one byte of data from Slave:

Master	ST	SAD+W		SUB		SR	SAD+R			NMAK	SP
Slave			SAK		SAK			SAK	DATA		

Master is receiving (reading) multiple bytes of data from Slave:

Master	ST	SAD+W		SUB		SR	SAD+R			MAK
Slave			SAK		SAK			SAK	DATA	

Master		MAK		NMAK	SP
Slave	DATA		DATA		

Table 5.2: I2C Master's and Slave's possibilities of transfers.

Reg. Name	Type	Register Address		Default
		Binary	Hex	
CTRL_REG1	rw	0100000	20	00000111
CTRL_REG2	rw	0100001	21	00000000
CTRL_REG3	rw	0100010	22	00001000
STATUS_REG	rw	0100111	27	00000000
OUTX_L	r	0101000	28	output
OUTX_H	r	0101001	29	output
OUTY_L	r	0101010	2A	output
OUTY_H	r	0101011	2B	output
OUTZ_L	r	0101100	2C	output
OUTZ_H	r	0101101	2D	output

Table 5.3: Main registers address map.

The functionality of the LIS3LV02DQ and the measured acceleration data are selectable and accessible through the I^2C/SPI interface. Since, the expansion circuit board is (as peripheral) connected to the Tmote Sky sensor module by the used 10-pin connector, it is necessary to communicate by I^2C with the Tmote Sky sensor module. A pin layout is designed according to the functionality of the Tmote Sky 10-pin expansion connector, see Chapter 3. We don't use RDY/INT pin, therefore, placing this pin is unimportant. CS and SDO pins are connected to the analog inputs (ADC2 and ADC3) and specify I^2C/SPI mode selection. When we using the I^2C, CS must be tied high while SDO must be left floating. LIS3LV02DQ SCL/SPC and SDA/SDI/SDO pins are connected to I^2C Clock (I^2C_SCL) and I^2C Data (I^2C_SDA) pins of Tmote Sky connector.

The Tmote Sky module with the manufactured designed expansion circuit board is presented in Figure 5.2.

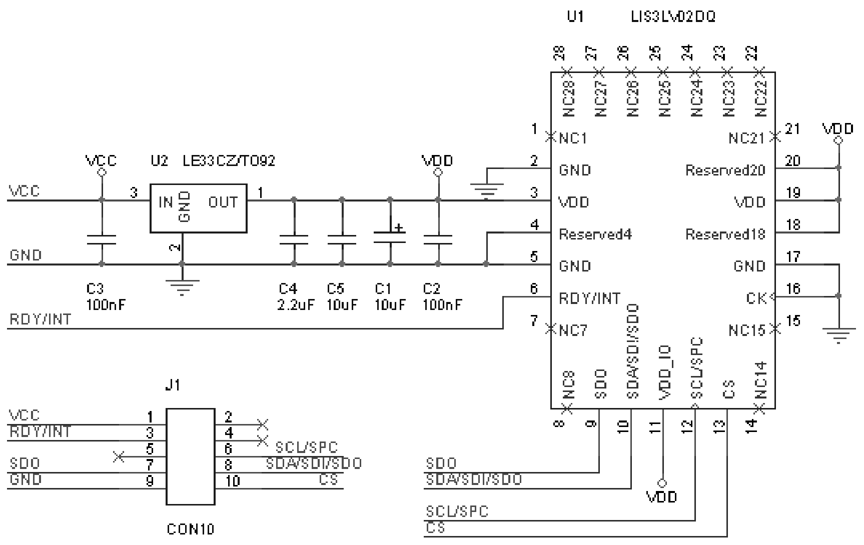


Figure 5.3: Circuit scheme with LIS3LV02DQ.

Chapter 6

Data Collection Applications using Open-ZB stack

A Tmote Sky applications for data acquisition, data processing, sending, receiving and data transmission to the personal computer are built by using the TinyOS operating environment v1.1.15, nesC language v1.2.7a and Open-ZB stack v1.2. The compile and install instructions for the Tmote Sky sensor modules are described in Appendix D.

6.1 Application to data acquisition by modules with analog accelerometer MMA7260Q

6.1.1 General Description of the Application

This application is formed for five devices in the star topology of an wireless network. There are four devices, each with expansion module with analog accelerometer, and operate as a end devices and a unique central device operates as the PAN coordinator.

Messages sending during the CAP period using the CSMA/CA is not useful, because of so much used end devices - there are a very large data rate and number of transmit data is very large too.

Consequently, a keystone of this application is using the *Guaranteed Time Slots* (GTS), see Chapter 2. Therefore, is possible to used up to seven end devices theoretically, when each of these devices allocate only one GTS, because the PAN coordinator accepts GTS requests up to seven. From this results the possibility of using only three end devices, which allocate each two GTS and they can transmit twice as much data in their reserved time slots within an Beacon Interval. This possibility is implemented too.

The communication is established between end devices and PAN coordinator. After the PAN coordinator is activated for the first time, it establish its own network with the noted

PAN identifier. At last each of end devices is joined to the network through this noted PAN identifier.

The devices try to allocate a GTS time slot to send their data by the request a transmit GTS allocation. After the allocation is successfully acknowledge and the PAN coordinator updates the GTS descriptors list in the beacon, the devices start to measuring and processing their data from attached expansion boards with an accelerometer and send periodic these data in payload data field of data packets to the PAN coordinator. When the PAN coordinator receives the data frame, reads and processes the payload field and stores data into the receiving buffer. When the PAN coordinator received all data frames in an Beacon Interval, it sends all data from receiving buffer by UART (Universal Asynchronous Receiver/Transmitter) through COM port to personal computer.

Measuring time of accelerometer data in each of end devices is need to be synchronized. For this purpose sends the PAN coordinator an broadcast sync packet for synchronizing the clocks of end devices at regular intervals.

The yellow led is on during the active period. Note for end devices, every time the data packet is sended to the PAN coordinator the end device, toggles red led. Note for PAN coordinator, every time the data packet is received the PAN coordinator, toggles green led and when a sync packet is sended to an end devices, toggles red led.

6.1.2 Application Components

All application files are located in the application folder. Application is named „DataSendAccel” and is composed of several components and auxiliary files:

- *DataSendAccel.nc* - the configuration file, is used to wire the DataSendAccelM.nc module to other components that the DataSendAccel application requires. All applications require a top-level configuration file, which is typically named after the application itself. It is the source file that the nesC compiler uses to generate an executable file.
- *DataSendAccelM.nc* - the module file, it actually provides the implementation of the DataSendAccel application
- *LocalTimeC.nc* - the configuration file, is used to wire the LocalTimeM.nc module to other components
- *LocalTimeM.nc* - the module file, it contains the implementation of the time component, that provides time services for TelosB platform (e.g. converting local clock ticks into miliseconds)
- *LocalTimeInfo.nc* - the interface file, this file contains the provided interfaces of the LocalTimeM module
- *mma7260q.h* - this file contains the enumeration values related to specify of ADC port

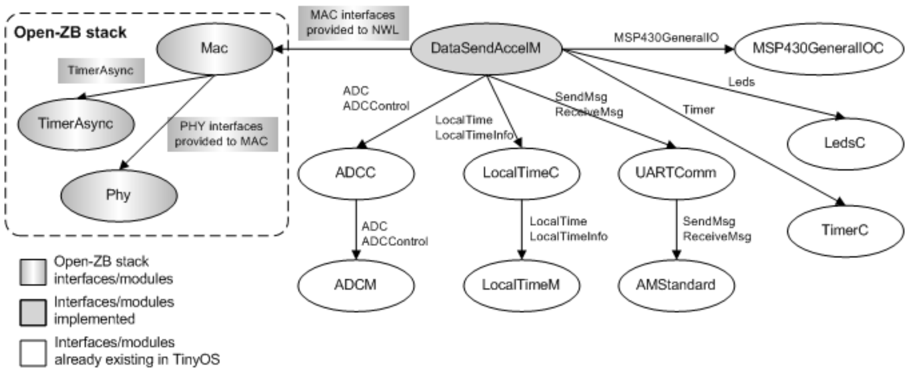


Figure 6.1: DataSendAccel Application - TinyOS Implementation Diagram.

- *datasendaccel.h* - this file contains the application constants definition related to implementation of IEEE 802.15.4/ZigBee protocol (e.g. coordinator or end device, BO, SO)
- *dsa_const.h* - this file contains the application constants definition related to DataSendAccel settings
- *dsa_enumerations.h* - this file contains the enumeration values used in the application
- *Makefile* - make-file for building application

This application is linked directly to the MAC layer of open-ZB stack. The Figure 6.1 illustrates the component wiring to the other component.

6.1.3 Detailed Analysis of the Application

All devices must have a manually assigned short address in the compilation, there must be a PAN coordinator device with an unique node ID 1 and four end devices with an unique node ID from 2.
When the *DataSendAccel* application starts, „Main" is a component that is executed first, therefore, an TinyOS application must have „Main" component in its configuration. More precisely, the Main.StdControl.init() command is the first command executed in TinyOS followed by Main.StdControl.start() and Main.StdControl.stop() when the component is stopped. StdControl is an interface used to initialize and start TinyOS components. In initialization we initiate all necessary variables, Leds, UART and ADC (see below 6.1.4) components, deactivate SleepMode functionality of analog accelerometer and set accelerometer sensitivity by g-Select1 and g-Select2 (see Section 6.1.5). In starting sequence we started a Timer, which ensures in five seconds that the event Timer.fired() is called.

On execution of the Timer there are two different operation modes depending if the device is the PAN coordinator or end device.

6.1.3.1 End Device

In event Timer.fired() is assigned the short address and the PANID to the end device by MLME_SET.request() primitive. Then the end device try to allocate a GTS time slot to sending data by the MLME_GTS.request() primitive and a repeat Timer_send is started. After the allocation is successfully acknowledge and the PAN coordinator updates the GTS descriptors list in the beacon, the device may starts to send data packets with measured data to the coordinator by issuing the MCPS_DATA.request primitive during its allocated GTS (don't use the CSMA/CA algorithm, data is sended directly without any channel assessment). The transmit options or TxOptions parameter, last argument of the primitive, define the transmission options for the data frame, allowing the frame to be send in the GTS (or during the CAP period using the CSMA/CA or like an indirect transmission, and also can be send with an acknowledgment request).

Every time the Timer_send fired, the end device measures and stores its accelerometers data and local time, when the third axis (axis Z) was measured. A difference between X axis measured time and Z axis measured time is 2-3ms. We have to consider with this delay during proccesing of accelerometer data. The device's accelerometer data are sampled by calling ADC.getData() for every axis to get a new sensor value. When the sensor value is available, the ADC.dataReady() event is signaled. This event is asynchronous code, it sould be protected by an atomic statement, because of the possibility of data races on shared data accessed by an async event or command. However, leaving interrupts disabled for a long period delays interrupt handling, which makes the system less responsive, therefore, shoud be an atomic code very small and quick.

When are measured six times all the three axis, the end device posts a task, putZB(), which sends the data frame with six sensor reading by MCPS_DATA.request primitive to the coordinator. Posted tasks with the accelerometer reading are executed by the TinyOS scheduler when the processor is idle. The structure of msdu payload data frame, that is sending by end device, is depicted in Figure 6.2.

The device's local time is measured by LocalTime.read() and LocalTimeInfo.ticksToMs(). LocalTime and LocalTimeInfo components are created by the help of time library, that wraps around platform dependant time services to provide platform independant Time/Timer interfaces and its implementation is very simple. This time library is developed by Jeongyeup Paek for purpose of the RCRT (Rate-Controlled Reliable Transport) protocol for Wireless Sensor Networks and is not included in the standard TinyOS distribution. It is available to download from [33].

When the end device receives the sync packet by MCPS_DATA.confirm() primitive, it reads the msdu payload field of this data frame and gets an PAN coordinator's local time from learned data. Then the end device reads its local time immediately. An offset is specified by:

```
syn_offset = ((c_local_time + SYN_ADD_CONST) - ed_local_time);
```

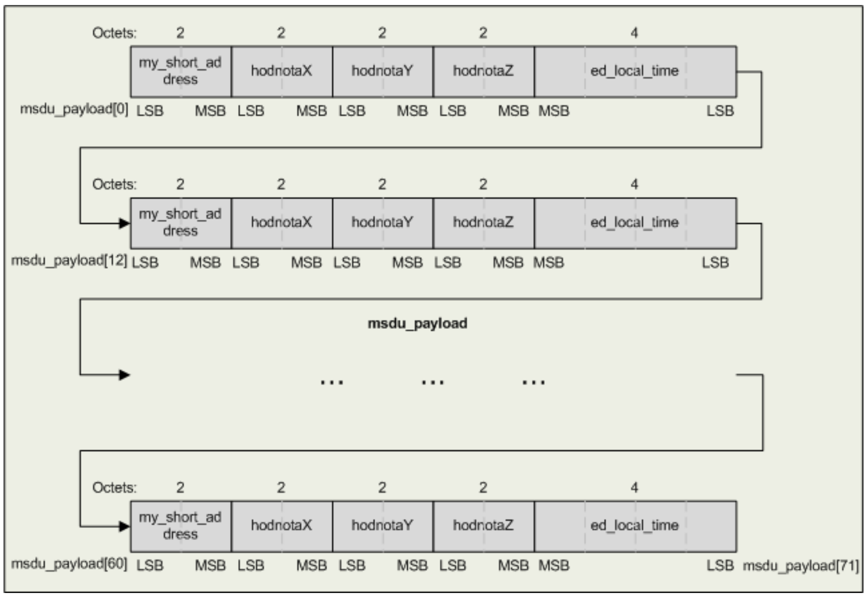

Octets: 2 2 2 2 4
my_short_ad dress
hodnotaX
hodnotaY
hodnotaZ
ed_local_time
msdu_payload[0] LSB MSB LSB MSB LSB MSB LSB MSB MSB LSB
msdu_payload[12] LSB MSB LSB MSB LSB MSB LSB MSB MSB LSB
msdu_payload
...
msdu_payload[60] LSB MSB LSB MSB LSB MSB LSB MSB MSB LSB msdu_payload[71]

Figure 6.2: The msdu payload field structure.

SYN_ADD_CONST is time compensation value, which includes an time delay between sync packet transmitting by PAN coordinator and sync packet receiving by end device. The sync offset value is added to the local time of end device.

6.1.3.2 PAN coordinator

In event Timer.fired() is assigned the short address and the PANID to the coordinator by MLME_SET.request() primitive. Then is started the Beacon sending by MLME_START. request() primitive and a repeat Timer_sync is started too. The coordinator waits for a GTS allocation by an end device. After the GTS allocation request is received and successfully acknowledged the PAN coordinator updates the GTS descriptors list in the beacon. When the PAN coordinator receives the data frame by MCPS_DATA.confirm() primitive, it reads the msdu payload field of this data frame and from learned data creates three TinyOS messages with the structure of payload data frame, that is depicted in Figure 6.3. The msdu payload data field structure is presented in Figure 6.2. The raw data packet and TinyOS message are described in the Section 6.1.6.

These three TOS message are stored into the receiving buffer „uartQueueBufs[]”. This buffer is represented an queue which is processed in FIFO order. When coordinator received a defined number of the data frame, in other words, when it stored a defined number of TOS messages „UART_MIN_BUFF_SEND” into the uartQueueBufs[], it starts a Timer_delay. On execution of the Timer_delay posts the coordinator a task, sendDataUART(), which sends all of the TOS message from buffer through COM port to personal computer. Using

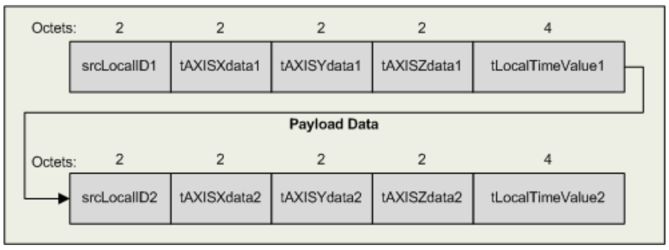


Figure 6.3: TOS message Payload Data field structure.

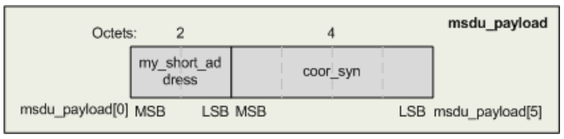


Figure 6.4: The sync mpdu data payload field structure.

the UART bus requires care because this line are physically shared with the data bus that connects the radio to the microcontroller. The Timer_delay defer the sending by UART until later, that the coordinator sends data to the computer in an Inactive Period or in an following Contention-Access Period (CAP), when the radio is inactive. The Contention-Free Period (CFP) is not interrupted by this functionality, thereby incorrupting an possible data packet, that could be received in running out of GTS yet. A size of the receiving buffer and UART_MIN_BUFF_SEND value should be choice suitable according to the number of possible received data packets in GTS slots from all end devices.

The Timer_sync is repeatedly used to synchronization of measuring time of accelerometer data in each of end devices. Every time the Timer_sync fired, the PAN coordinator measures its local time by LocalTime.read() and LocalTimeInfo.ticksToMs() and sends an broadcast synchronization packet to all end devices for synchronizing the clocks by issuing the MCPS_DATA.request primitive. Sending is realized during the CAP period using the CSMA/CA. The structure of sync packet is depicted in Figure 6.4.

6.1.4 Connecting External Sensors

The principal question is, how we connect external sensors. We using an ADC channel on the 10-pin Tmote Sky expansion area and we sampling ADC data from external sensor.

The essence of preparing an external ADC port for TinyOS is in the definition of two numbers: TOS_ADC_xxx_PORT and its partner TOSH_ACTUAL_ADC_xxx_PORT. "xxx" is a name you can pick when defining these values, perhaps picking a name that describes the used sensor. In this case we have an 3D accelerometer, whose three axes can we think of three sensors. Hence we have a three sensors called e.g. MMA_X, MMA_Y and MMA_Z connected to ADC channel 0, 1 and 2 on the 10-pin Tmote Sky expansion connector. We shoud define those two values above for every sensor (axis) in a header file

„mma7260q" like this:

```
enum {
  TOS_ADC_MMA_X_PORT = unique("ADCPort"),
  TOSH_ACTUAL_ADC_MMA_X_PORT = ASSOCIATE_ADC_CHANNEL(
          INPUT_CHANNEL_A0,
          REFERENCE_VREFplus_AVss,
          REFVOLT_LEVEL_2_5),
     };
```

For X axis/sensor, change the „MMA_X" name and the three parameters given to ASSOCIATE_ADC_CHANNEL. Those parameters are:

1. The pin on the microcontroller your device is connected to. INPUT_CHANNEL_AO means „ADC0" or „ADC port 0". Find the names of the pins in the Chapter 3. or in the Tmote Sky datasheet [23] in the section „External Sensors". Here, pin 3 of the 10-pin expansion header is connected to ADC0.

2. Use the internal reference voltage specified with REFERENCE_VREFplus_AVss. Find the other options of the internal reference voltage in the `TOSROOT\tos\platform\msp430\MSP430ADC12.h`. Here, VR+ = VREF+ and VR-= AVss (analog ground). Note: TOSROOT = the root of your TinyOS source directory.

3. For the internal reference voltage VREF, use the 2.5V reference voltage specified with REFVOLT_LEVEL_2_5. The one other option here is the 1.5V internal reference voltage specified with REFVOLT_LEVEL_1_5.

Should study ADC12 module (12-bit analog-to-digital converter) of MSP430 microprocessor for understanding functionality of VR, VREF etc., find this shortcuts of the voltage in the MSP430x1xx Family User's Guide [24] in the Chapter 17 „ADC12".

Then we wire up the ADC and ADCControl interfaces in our component configuration:

```
Components ADC;
Main.StdControl -> ADCC;
DataSendAccelM.ADCControl -> ADCC;
DataSendAccelM.osaX -> ADCC.ADC[TOS_ADC_MMA_X_PORT];
DataSendAccelM.osaY -> ADCC.ADC[TOS_ADC_MMA_Y_PORT];
DataSendAccelM.osaZ -> ADCC.ADC[TOS_ADC_MMA_Z_PORT];
```

The last step is to bind and use the ADC port. In the module file we must use the ADC interface for each ADC port we are going to use, and use the ADCControl interface once to perform the ADC port „binding". Here are some key lines of „DataSendAccelM" module code:

```
module DataSendAccelM {
    uses { interface ADCControl;
           interface ADC as osaX;
           interface ADC as osaY;
           interface ADC as osaZ; }
}
implementation {
command result_t StdControl.init() {
  //Initialize the ADC subsystem
call ADCControl.init();
  //Initialize the MMA_X, MMA_Y and MMA_Z ADC port
  //by binding it to its actual specification
call ADCControl.bindPort(TOS_ADC_MMA_X_PORT,TOSH_ACTUAL_ADC_MMA_X_PORT);
call ADCControl.bindPort(TOS_ADC_MMA_Y_PORT,TOSH_ACTUAL_ADC_MMA_Y_PORT);
call ADCControl.bindPort(TOS_ADC_MMA_Z_PORT,TOSH_ACTUAL_ADC_MMA_Z_PORT);
  }
}
```

Later, you can just call ADC.getData which will response later with an ADC.dataReady event:

```
call osaX.getData();
...
async event result_t osaX.dataReady(uint16_t data)
{ // do something with data }
```

6.1.5 Sensitivity and Sleep Mode Settings

The g-Select feature allows for the selection among 4 sensitivities present in the device, see Chapter 5. The device internal gain will be changed to depending on the logic input placed on pins 1 and 2 of the MMA7260Q analog accelerometer. A low input signal on pin 12 will place the device in Sleep Mode, the device outputs are turned off. By placing a high input signal on pin 12, the device will resume to normal mode of operation.

The MSP430GeneralIO interface is utilized to control all pins of the 10-pin Tmote Sky expansion connector. Take a look at the schematic given in the Tmote Sky datasheet [23] to find out which pin is connected to a certain port. For this example, port 3.4 and port 3.5 are routed to pin 4 and pin 2 of expansion connector and represent g-Select1 and g-Select2. As well as port 6.3 is routed to pin 10 of expansion connector and represent Sleep Mode functionality. To use the MSP430GeneralIO interface wire it to the MSP430GeneralIOC.Portxx where „Portxx" represent the pin of interest. The interface is called in the StdControl.init() function that runs on start-up of the DataSendAccel application.

Then we set the microprocessor port thus expansion connector pin to output and turn on (means writing a „1" to the pin) or off (means writing a „0" to the pin) the pin functionality using the commands MSP430GeneralIO.makeOutput() and MSP430GeneralIO.setHigh() or MSP430GeneralIO.setLow() respectively.

6.1.6 Raw data packet and TinyOS message

This section serves as a beginners guide to deciphering TinyOS serial packet.

The data are retrieved in so called raw data format. The raw data packet is wrapped on both ends by a frame synchronization byte of 0x7E. When is sending a packet over the serial port, the protocol uses the byte values of 0x7E and 0x7D for special purposes. 0x7E is used to detect the start and end of a packet from the stream and 0x7D is used as the escape byte to indicate that the next byte has been AND-ed with 0xDF. All bytes of 0x7E and 0x7D in the raw data packet need to be replaced by a 2-byte sequence before sending. 0x7E is replaced by 0x7D5E and 0x7D is replaced by 0x7D5D. When receiving a packet, the escape byte is discarded and the next byte is OR-ed with 0x20 in order to get the actual byte of the packet.

The following diagram and table describes the raw data packet with a TinyOS message type of the payload data. The TinyOS message is defined by the struct TOS_Msg in the file `TOSROOT\tos\platform\telos\AM.h`.

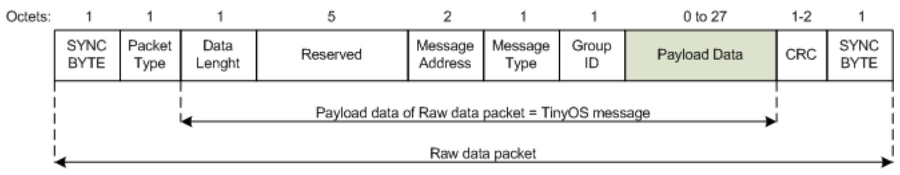
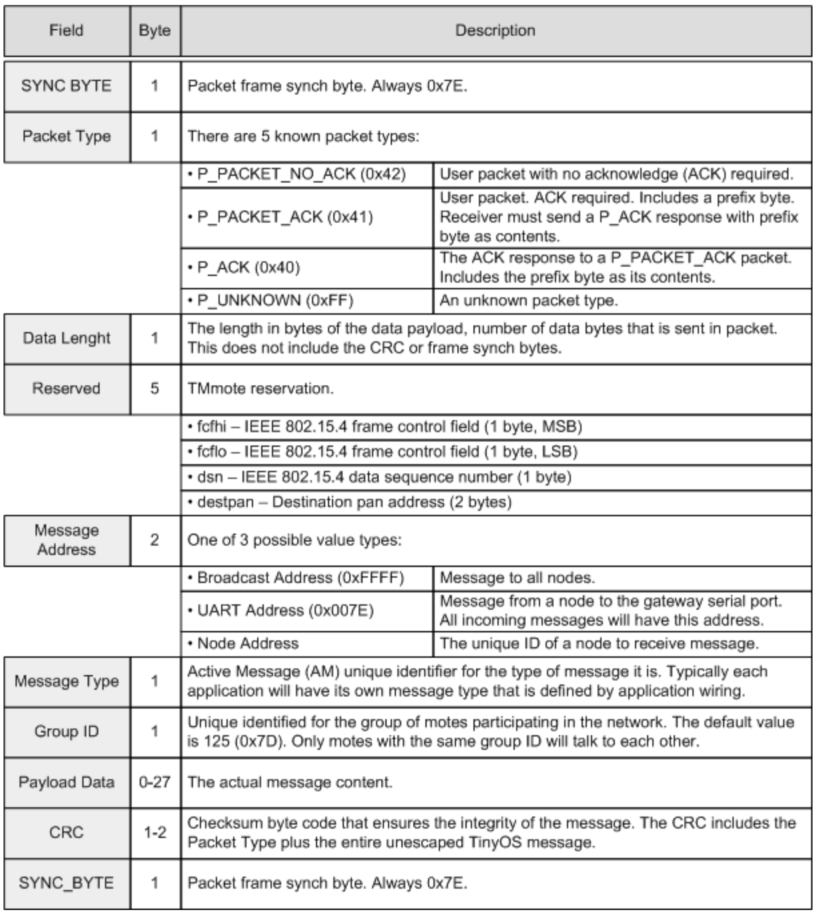


Figure 6.5: TOS-message structure.



Field	Byte	Description	
SYNC BYTE	1	Packet frame synch byte. Always 0x7E.	
Packet Type	1	There are 5 known packet types:	
		• P_PACKET_NO_ACK (0x42)	User packet with no acknowledge (ACK) required.
		• P_PACKET_ACK (0x41)	User packet. ACK required. Includes a prefix byte. Receiver must send a P_ACK response with prefix byte as contents.
		• P_ACK (0x40)	The ACK response to a P_PACKET_ACK packet. Includes the prefix byte as its contents.
		• P_UNKNOWN (0xFF)	An unknown packet type.
Data Lenght	1	The length in bytes of the data payload, number of data bytes that is sent in packet. This does not include the CRC or frame synch bytes.	
Reserved	5	TMmote reservation.	
		• fcfhi – IEEE 802.15.4 frame control field (1 byte, MSB)	
		• fcflo – IEEE 802.15.4 frame control field (1 byte, LSB)	
		• dsn – IEEE 802.15.4 data sequence number (1 byte)	
		• destpan – Destination pan address (2 bytes)	
Message Address	2	One of 3 possible value types:	
		• Broadcast Address (0xFFFF)	Message to all nodes.
		• UART Address (0x007E)	Message from a node to the gateway serial port. All incoming messages will have this address.
		• Node Address	The unique ID of a node to receive message.
Message Type	1	Active Message (AM) unique identifier for the type of message it is. Typically each application will have its own message type that is defined by application wiring.	
Group ID	1	Unique identified for the group of motes participating in the network. The default value is 125 (0x7D). Only motes with the same group ID will talk to each other.	
Payload Data	0-27	The actual message content.	
CRC	1-2	Checksum byte code that ensures the integrity of the message. The CRC includes the Packet Type plus the entire unescaped TinyOS message.	
SYNC_BYTE	1	Packet frame synch byte. Always 0x7E.	

Table 6.1: TOS-message description.

6.2 Application to data read from modules with digital accelerometer LIS3LV02DQ

6.2.1 General Description of the Application

This application is not created for data acquisition using to wireless sensor network. There are only devices with expansion module with digital accelerometer attached to USB port. Is not relevant, whether a device operates as a end device or as a coordinator. The reason is, the I^{2}C and CC2420 radio cannot be used at the same time on the TelosB platform. The I^{2}C pins are shared with the radio's data input pin and the radio clock. Therefore, when there is data transfer between the radio and microprocessor, we don't use I^{2}C input pins on expansion connector, those pins will be actuated with SPI protocol signals, because the radio is controlled by the microprocessor through SPI interface, see [23].

The solution is to read input signals while the radio is not in use, it means that the I^2C bus and the radio operations must be multiplex. In using TinyOS 1.1.15, we would have to use the BusArbitration component to acquire required pins and block the radio from using them. Nevertheless, the BusArbitration doesn't work reliable and it is hard to use correctly. This problem is fixed with Resource components in TinyOS 2.x or Moteiv Boomerang environment based on TinyOS 2.x. The Resource components are used to gain access to shared resources through some predefined arbitration policy.

The application is developed to according to an bachelor thesis [36], in which is described an implementation of I^2C communication for MICA platform (note: in italian language). We created the components for communication with digital accelerometer by using I^2C bus - data writing, reading, and the components for processing of acquired data and data transmission to the personal computer by serial port. These components demonstrate using I^2C bus. As the implementation of IEEE 802.15.4/ZigBee will be ported to TinyOS v2.0, it is proceed at this time, this application will be simple ported to TinyOS v2.0 too and used to creating an wireless sensor network with Tmote Sky sensor modules with expansion boards with digital accelerometer LIS3LV02DQ.

6.2.2 Application Components

All application files are located in the application folder. Application is named „LIS3LV02DQ" and is composed of several components and auxiliary files:

- *I2CAccelerometer.nc* - the interface file, this file contains the provided interfaces of the I2CAccelerometerM module
- *I2CAccelerometerC.nc* - the configuration file, is used to wire the I2CAccelerometerM.nc module to other components
- *I2CAccelerometerM.nc* - the module file, it provides the implementation of the I^2C communication
- *LIS3LV02DQC.nc* - the configuration file, is used to wire the LIS3LV02DQM.nc module to other components that the LIS3LV02DQ application requires. It is the source file that the nesC compiler uses to generate an executable file.
- *LIS3LV02DQM.nc* - the module file, it actually provides the implementation of the LIS3LV02DQ application
- *LocalTimeC.nc* - the configuration file, is used to wire the LocalTimeM.nc module to other components

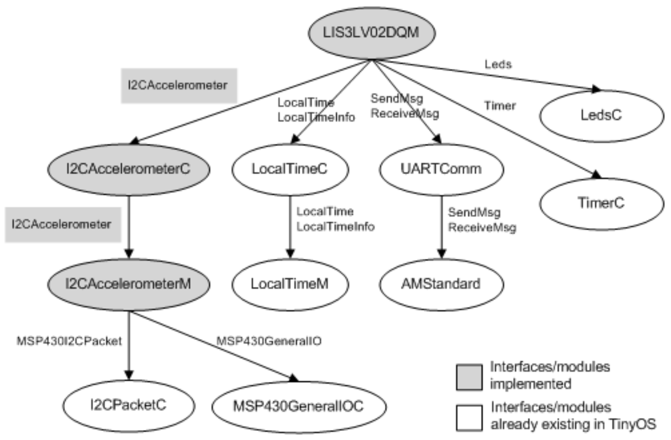


Figure 6.6: LIS3LV02DQ Application - TinyOS Implementation Diagram.

- *LocalTimeM.nc* - the module file, it contains the implementation of the time component, that provides time services for TelosB platform (e.g. converting local clock ticks into miliseconds)
- *LocalTimeInfo.nc* - the interface file, this file contains the provided interfaces of the LocalTimeM module
- *i2c_accelerometer_const.h* - this file contains the application constants definition related to the I^2C communication of digital accelerometer
- *i2c_accelerometer_enum.h* - this file contains the enumeration values related to the I^2C communication of digital accelerometer
- *lis3lv02dq_const.h* - this file contains the application constants definition related to LIS3LV02DQ application settings
- *lis3lv02dq_enum.h* - this file contains the enumeration values used in the application
- *Makefile* - make-file for building application

The Figure 6.6 illustrates the component wiring to the other component.

6.2.3 Detailed Analysis of the Application

When the „LIS3LV02DQ" application starts, the Main.StdControl.init() command is the first command executed in TinyOS followed by Main.StdControl.start(). In initialization we initiate all necessary variables, Leds, UART and I2CAccelerometer components and repeat Timer is started.

When the I2CAccelerometer component starts, it activate the I^2C interface by CS line tied high using the MSP430GeneralIO interface and save the control register 1 address with

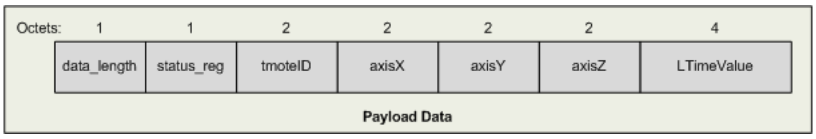


Figure 6.7: TOS message Payload Data field structure for digital accelerometer.

autoincrement sign, control register 1 value and control register 2 value in the data_write[] array. The register description is available in [32]. Then the I^2C address of accelerometer with the data_write[] array are transmitted on the I^2C bus by MSPPacket.writePacket() primitive. This way is digital accelerometer initialized. Then the I2CAccelerometer components performing the I^2C operations according to I^2C-bus specification, see Section 5.2.1.1.

A Timer is used to periodically start the data collection. Every time the Timer fired, the device read the accelerometer's data by I2CAccelerometer.readNReg() primitive from STATUS_REG register to OUTZ_H register by autoincrement sign, see Table 5.3. When the sensor's data is available, the I2CAccelerometer.readNRegDone() event is signaled. The device measures its local time by LocalTime.read() and LocalTimeInfo.ticksToMs() and with accelerometers data together creates a TinyOS message with the structure of payload data frame, that is depicted in Figure 6.7. The created TinyOS message is immediately sended by a posted task „sendDataUART()" by UART through COM port to personal computer.

Every time the data packet is received from accelerometer, toggles yellow led.

Chapter 7

Results and Experience

7.1 Problems and Necessary Changes in the Open-ZB stack and TinyOS/nesC

7.1.1 Open-ZB stack

- We have removed all the usage of PrintfUART() function from open-ZB stack components. PrintfUART() writes output to the UART as like printf function. It should be use only for testing purposes. The problem is, the PrintfUART() function and used UARTComm commponent may send messages by the UART over the serial port at the same time and then the sent message is corrupted.

- When we are using the Guaranteed Time Slots (GTS) and sending messages in allocated GTS slots by calling MCPS_DATA.request() primitive, we could remove these lines from MacM module:

  ```
  total_ticks = call TimerAsync.get_total_tick_counter();
  msdu[0] =(uint8_t)(total_ticks >> 0);
  msdu[1] =(uint8_t)(total_ticks >> 8);
  msdu[2] =(uint8_t)(total_ticks >> 16);
  msdu[3] =(uint8_t)(total_ticks >> 24);
  ```

  so that we resist storing a TimerAsync tick counter value in msdu array and we could use the msdu data payload field of message from index „0".

- When we are using the Guaranteed Time Slots (GTS) and sending messages in allocated GTS slots by calling MCPS_DATA.request() primitive, the msdu payload data field must be defined as the size of msdu data plus two or larger. Then the data are transmit correctly. This is the open-ZB stack bug.

- When we are using the Guaranteed Time Slots (GTS) with the ADC component for connecting external sensors together, we cannot use the equal values of Beacon Order and Superframe Order. That means, the superframe structure must have an Inactive Period during which the PAN coordinator does not interact with its PAN or the Beacon synchronization does not work correctly then.
- Unfortunately, with using the different values of BO and SO always does not work the Beacon synchronization reliably. Sometimes the entire Beacon Interval is lost. It can be observed on toggling of the yellow led which is on during the active period of superframe. It is probably caused by asynchronous commands and events of ADC channel, that are used to sampling ADC data from external sensors.
- In mac_const.h we enlarge the GTS_SEND_BUFFER_SIZE definition from „2” to „11” value, so that we can store more messages in GTS send buffer. Using an even larger value, the open-ZB stack functionality is damaged.
- TinyOS message payload data field is defined to 28 Bytes value and cannot we insert to this structure more than two data packet with accelerometer's data. When we try to increase this value, the open-ZB stack functionality is damaged.

7.1.2 TinyOS/nesC

- It is recommended to use the name of the Timer parameterized interface as an argument to the unique() function, which generates a unique 8-bit identifier from the string given as an argument, e.g. „interface Timer[unique("Timer")];”.
- If any Java application does not work, make sure that the JDK that actually using (i.e., the one that is found first in Windows PATH variable), is actually the TinyOS approved one. Can check in Cygwin by typing
  `$ which java`, it should look something like this:
  `cygdrive/c/jdk1.4.1_02/j2sdk1.4.1_02/bin/java`
- If trying to run Cygwin and still occuring the some error like this: „An unhandled win32 exception occurred in > bash.exe[]”. Cygwin produced this exception in dependence on the some kind of bad interaction with security software (firewall/antivirus/antispyware). For example, the Firewall Agnitum Outpost Pro 4.0 causes this „unhandled exception" issue. Uninstalling and changing this software solve the problem.
- When attach the Tmote Sky module with batteries and a running application, the operation system does not need to identify the hardware correctly. Just unplug the batteries and attach the Tmote Sky again.
- Everything is in order, but „`$ make <platform>`” command still does not work, try to change permissions of the „build” directory and all of files inside by the following

command:
$ chmod 777 <directory_name or file_name> and try to compile of application again.

7.2 Results

7.2.1 Application to data acquisition by modules with analog accelerometer MMA7260Q

According to detected problems above we choose as optimal settings of application and open-ZB stack following values:

- Beacon Order = 7, Superframe Order = 6
- the Timer component, which is based on the hardware timer for the TelosB platform, provides only correct time meassuring in using some time value, it have to be tested for every time value but the best is used 2^ntime values. In the following points is stated an really time value, with that is really measured accelerometer's data and an set value of Timer component in curves.
- accelerometer's data meassuring time interval = 74ms (76ms) in using four end devices - one GTS for each of them
  - this implies the achieved frequency of data aquisition is about 13Hz
- accelerometer's data meassuring time interval = 37-38ms (56ms) in using three end devices (so called three node mode) - two GTS for each of them
  - this implies the achieved frequency of data aquisition is about 26Hz
- theoretically is possible to used up to seven end devices
- the size of the UART receiving buffer = 48 in using four end devices, 75 in using three end devices
- sending by UART defer time value = 500ms
- synchronization time interval = 5000ms
- time compensation value for time synchronization = 2ms
- we achieve 0.04% corrupted packets during sending and receiving
- solution of problem with bad Beacon synchronization is failed. This is the open-ZB stack unpleasant bug. Without it we would achieve double the frequency of data acquisition. This defect is possible to see in depicted graphs, e.g. in Figure 7.1 around time 95s or in Figure 7.2 around time 105s.

Graphs of measured accelerometer's data

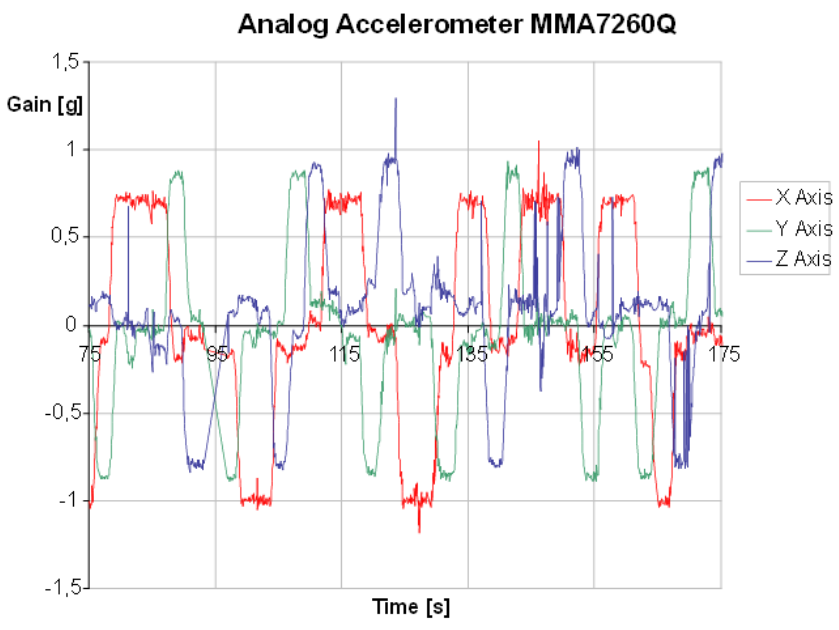


Figure 7.1: Measuring data through MMA7260Q, sensitivity 800mV/g (±1,5g), freq. 13Hz.

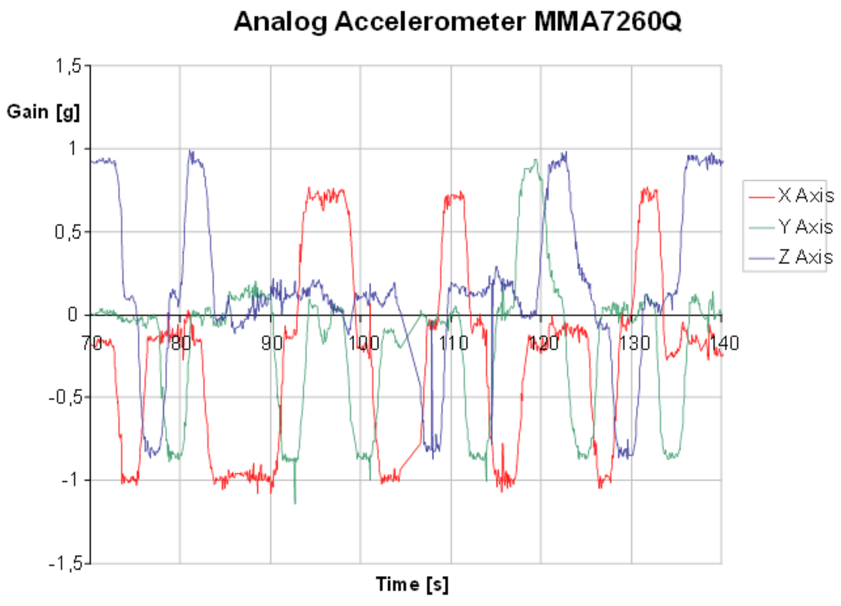


Figure 7.2: Measuring data through MMA7260Q, sensitivity 600mV/g (±2g), freq. 13Hz.

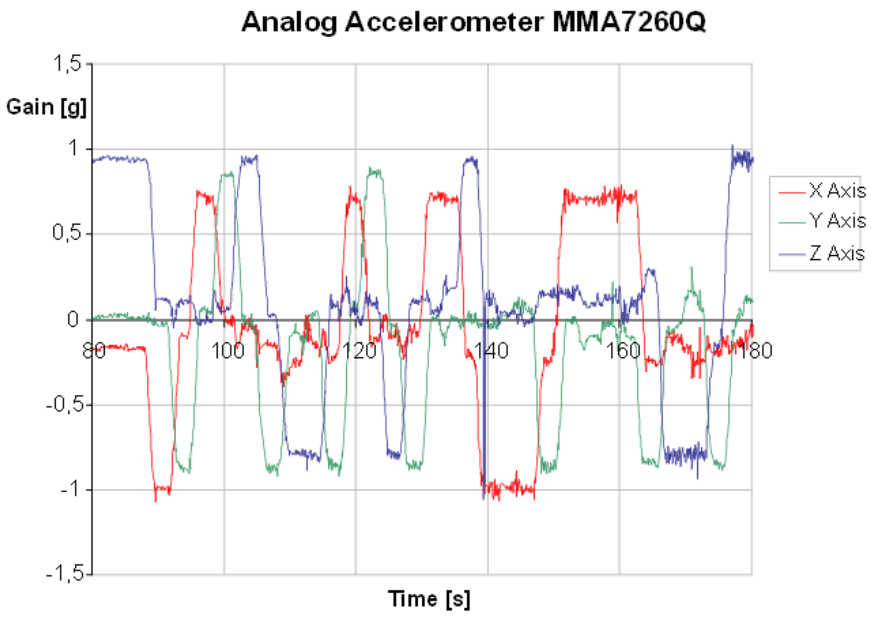


Figure 7.3: Measuring data through MMA7260Q, sensitivity 300mV/g (±4g), freq. 13Hz.

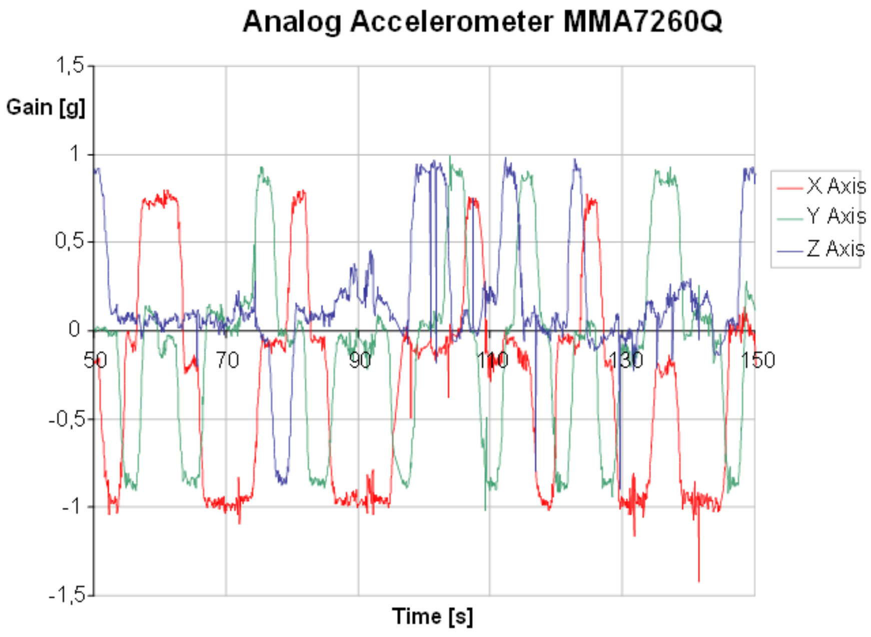


Figure 7.4: Measuring data through MMA7260Q, sensitivity 200mV/g (±6g), freq. 13Hz.

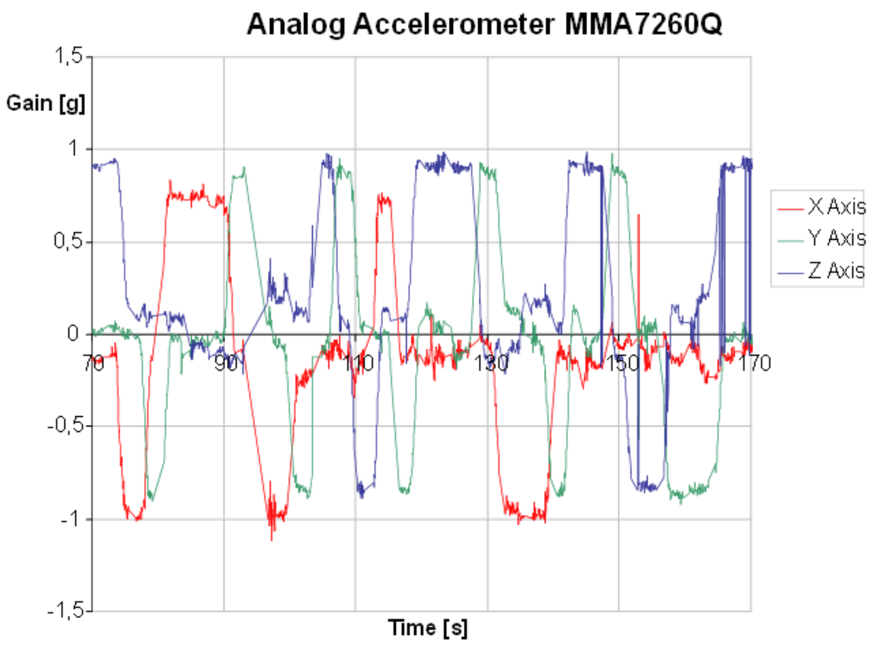


Figure 7.5: Measuring data through MMA7260Q, three node mode, sensitivity 200mV/g (±6g), frequency 26Hz.

7.2.2 Application to data read from modules with digital accelerometer LIS3LV02DQ

Appropriate settings of this application:

- accelerometer's data meassuring time interval = 18-19ms (set value of Timer component: 25ms)
  - this implies the achieved frequency of data aquisition is about 52-56Hz

Graphs of measured accelerometer's data

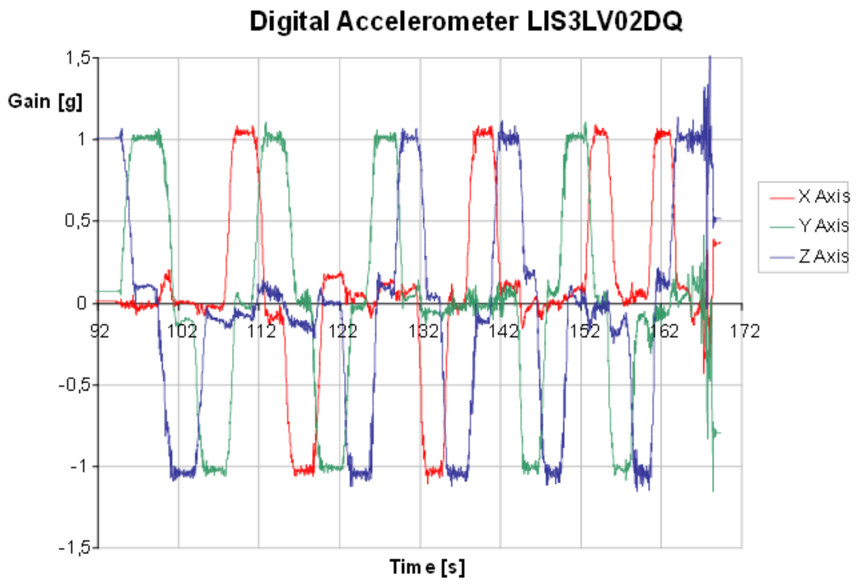


Figure 7.6: Measuring data through LIS3LV02DQ, sensitivity 340 LSb/g (±6g), frequency 52-56Hz.

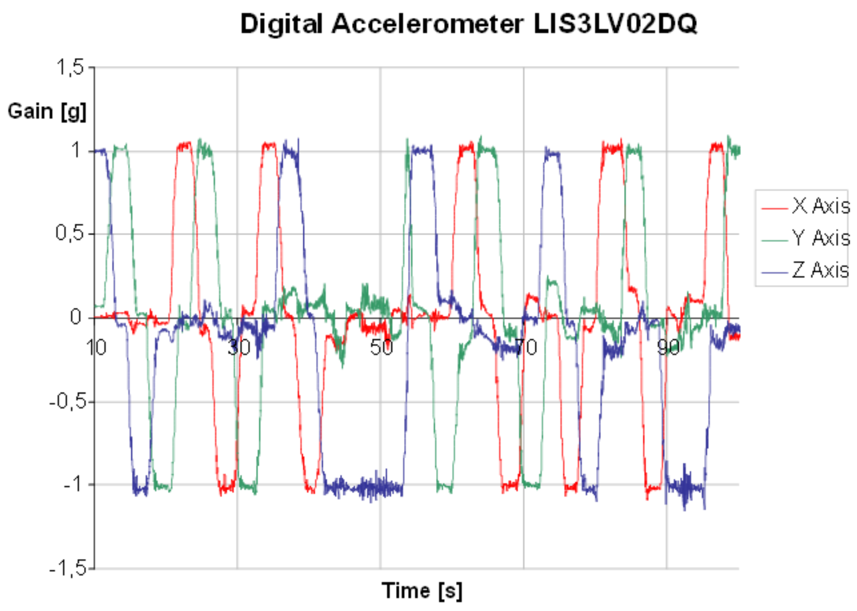


Figure 7.7: Measuring data through LIS3LV02DQ, sensitivity 1024 LSb/g (±2g), frequency 52-56Hz.

Chapter 8

Applications for Data Processing and Displaying

PC applications for data processing and displaying are created by Microsoft .NET Framework in MS Visual Studio 2005 using C# language and in C language under Cygwin linux-like environment for Windows. The compile and install instructions are described in Appendix D.

8.1 Console Application in C under Cygwin

8.1.1 General Describe the Application

When the application started, it attempt to open selected COM port as first by openDevice() function. A port is opened by open() standard function with appropriate parameters and the recieved buffer is emptied. Then setOptions() function is called, that makes proper settings of serial port for communication. Consequently the directory with storage files for received packets and data are created. A time stamp are written to the files for identification. In while cycle is reading data stream from the COM port by read() in getData() function. Data are putting in and getting from the data array by a pointers. Data are recognizing from data stream by 0x7E value, that is used to marking the start and end of a raw data packet from the data stream, see Section 6.1.6. Raw data packets are processing towards the TinyOS message structure of payload data frame is recovered. The acquired accelerometer's data, its time stamp, node ID etc. are displayed on the monitor and saved in the file by printf() and fprintf() function. In the second file are saved all the received raw data packet. Data format in files is chosen suitable for followed processing in Matlab.

8.1.2 Application Files

All application files are located in the application folder. Application is named „spZB" and is composed of:

- *sp.c* - source code of application
- *sp.h* - header file, that is automatically included in source file by the compiler, the file contains the application constants definition related to spZB application settings

8.2 GUI Application in C#

Application is controlled by the main form window, depicted in Figure 8.1. By this main form and other forms such as form for application or serial port settings (depicted in Figure 8.2.) are available all of implemented functionality.

There are used some library, that are not a part of MS Visual Studio installation and provide an useful function. These are the ZedGraph [41, 40] for creating 2D line graphs and Bin library for hexadecimal conversions to signed integers and unsigned integers.

First, after the application started, the user must set the operation mode by Application Settings form - to what end will be the application used - if we want to read data from Tmote Sky module with analog MMA7260Q or digital accelerometer LIS3LV02DQ. In this form is possible to permit and set the sensitivity of used accelerometer in mV/g for analog and LSb/g for digital accelerometer. The user may permit graph depiction (Graph ON), automatic graph depiction once per two seconds (Graph SHOW) and set parameters to graph depiction, such as the choice of requred node ID and the number of data packets, whose data are displayed to the graph.

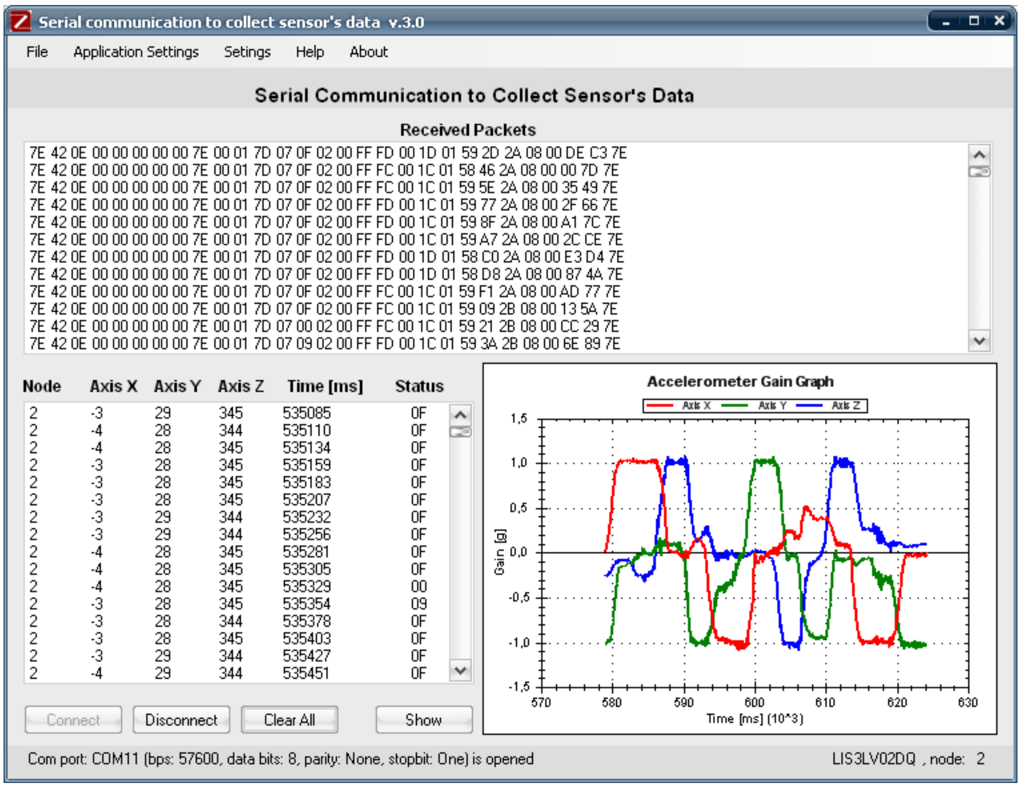
Second, the user must set the properties of serial port where the required mote is plugged by Port Settings form. For user's comfort is implemented an automatic port scan, which finds all of available active ports.

The keystone of this application is using DataReceived event handler, that signals an available data. Consequently, data are read by using SerialPort.Read() from port and are processed in the same way as like in console application. Received raw data packets and processed accelerometer's data are displayed in text boxes, stored in the files and depicted in graph. Data format in files is chosen suitable for followed processing in Matlab.

8.2.1 Application Files

All application files are located in the application folder. Application is named „Serial-ComZB" and is composed of several source files:

- *Form1.cs, FormAbout.cs, FormHelp.cs, FormSet.cs, FormSetAppl.cs* - source code of Main form, About form, Help form, port settings form and application settings form
- *xxx.Designer.cs* files - contain code automatically generated by the Windows Form Designer, in other words a xxx.Designer.cs file contains all of the code about the form that is automatically generated when you drag components to the form from the toolbox
- *xxx.resx* files - contain information about the design of the forms
- *Program.cs* - in this file is the starting point for program, it sets environment parameters and calls for the creation of Form1
- *Help.rtf* - help file, it provides a documentation on use of the application
- *Bin.dll* - Bin is a set of classes to do hexadecimal, binary, and decimal conversions to signed integers and unsigned integers. [39] Library has no explicit license.
- *ZedGraph.dll* - ZedGraph [41, 40] is a set of classes, for creating 2D line and bar graphs of arbitrary datasets. ZedGraph also includes a UserControl interface, allowing drag and drop editing within the MS Visual Studio forms editor and is licensed under the LGPL (GNU Lesser General Public License) [42].



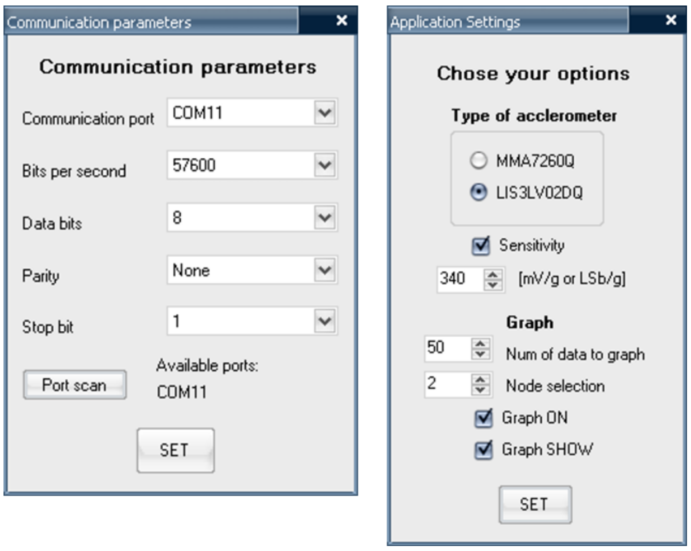
```
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FD 00 1D 01 59 2D 2A 08 00 DE C3 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FC 00 1C 01 58 46 2A 08 00 00 7D 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FC 00 1C 01 59 5E 2A 08 00 35 49 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FD 00 1C 01 59 77 2A 08 00 2F 66 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FD 00 1C 01 59 8F 2A 08 00 A1 7C 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FD 00 1C 01 59 A7 2A 08 00 2C CE 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FD 00 1D 01 58 C0 2A 08 00 E3 D4 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FD 00 1D 01 58 D8 2A 08 00 87 4A 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FC 00 1C 01 59 F1 2A 08 00 AD 77 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 0F 02 00 FF FC 00 1C 01 59 09 2B 08 00 13 5A 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 00 02 00 FF FC 00 1C 01 59 21 2B 08 00 CC 29 7E
7E 42 0E 00 00 00 00 00 7E 00 01 7D 07 09 02 00 FF FD 00 1C 01 59 3A 2B 08 00 6E 89 7E
```

Node	Axis X	Axis Y	Axis Z	Time [ms]	Status
2	-3	29	345	535085	0F
2	-4	28	344	535110	0F
2	-4	28	345	535134	0F
2	-3	28	345	535159	0F
2	-3	28	345	535183	0F
2	-3	28	345	535207	0F
2	-3	29	344	535232	0F
2	-3	29	344	535256	0F
2	-4	28	345	535281	0F
2	-4	28	345	535305	0F
2	-4	28	345	535329	00
2	-3	28	345	535354	09
2	-3	28	344	535378	0F
2	-3	28	345	535403	0F
2	-3	29	344	535427	0F
2	-4	29	344	535451	0F

Com port: COM11 (bps: 57600, data bits: 8, parity: None, stopbit: One) is opened

LIS3LV02DQ , node: 2

Figure 8.1: GUI application in C# - Main window.



Communication parameters

Communication port	COM11
Bits per second	57600
Data bits	8
Parity	None
Stop bit	1

Available ports:
COM11

Chose your options

Type of acclerometer

MMA7260Q [✓ LIS3LV02DQ]

✓ Sensitivity

340 [mV/g or LSb/g]

50 Num of data to graph

2 Node selection

✓ Graph ON

✓ Graph SHOW

Figure 8.2: GUI application in C# - Communication and Application settings options.

Chapter 9

Conclusion

We managed to create an wireless sensor network, that is able to monitoring patient's body. According to the requirements was chosen the most suitable ZigBee wireless technology. For this technology is developing an open source implementation, so called open-ZB stack. Because we choose the Tmote Sky sensor modules from Sentilla/Moteiv Corporation as hardware platform for wireless communication, we have to porting the open-ZB stack to this platform at first by collaborating with its developers. Porting is done successfully. This implementation was tested during the working at this thesis and we found some unpleasant bugs, the requirement using the different values of Beacon Order and Superframe Order when we want to use the Guaranteed Time Slots with the ADC component for connecting external sensors together and problems with the Beacon synchronization, that often does not work correctly. But this is an open source implementation and its development continues. If these bugs was eliminated, we would be approached much more to required frequency of data acquisition.

We designed the expansion modules with analog accelerometer MMA7260Q for application to data acquisition from patient's body. The application was built in TinyOS/nesC and open-ZB stack. In this way we achieved the frequency 13Hz in using one PAN coordinator and four end devices and 26Hz in using three end devices both at the accelerometer sensitivity from 800mV/g ($\pm$1.5g) to 200mV/g ($\pm$6g). We also designed the expansion modules with digital accelerometer LIS3LV02DQ only for data reading application in TinyOS/nesC, due to the using TinyOS v1.1.15 where the BusArbitration doesn't work correctly.

We used the GTS mechanism for a created wireless sensor network, therefore, is possible to used up to seven end devices theoretically. A possibility of using CSMA/CA mechanism in Contention-Access Period of Beacon Interval has been dismissed immediately. When we are transmitting as much of accelerometer's data and as large frequency in using so much devices, the contention mechanism is useless. We still have to think of ZigBee technology is optimized for low data rate transmissions.

The Department of Control Engineering have not available any network protocol analyser or packet sniffers for interpret the IEEE 802.15.4 and ZigBee frames. Effective testing of communication flow in the created network was very difficult.

Alternative solution is leave the open-ZB stack and just using TinyOS components for CC2420 radio chip, that access the implementation of physical layer and reduced MAC layer and create an centralized but peer-to-peer wireless sensor network, in which the coordinator will ask each of end devices for its data and acknowledge its receipt.

Department of Control Engeneering is developing own ZigBee radio platform, therefore, could be use some commercially developed ZigBee stack (full IEEE 802.15.4, reduced/full MAC layer or complete ZigBee implementation). The ZigBee stack offers e.g. Microchip Technology for its PIC processors (PIC18,24,33), this stack is free or Freescale Semiconductor offers the BeeKit Wireless Connectivity Toolkit for its processors (HCS08 family), that includes an unlimited use license for the reduced MAC layer and IEEE 802.15.4 codebases and a 90-day evaluation of Freescale's BeeStack ZigBee codebase.

This thesis will be used to other research on sensing and processing data for Parkinsonian patients at Department of Control Engineering and we believe that contains valuable information to other users and developers using, maintaining and expanding the open-ZB protocol stack.

Future work:

- develop an expansion memory module for central node with Vinculum VNC1L (Embedded USB host Controler) for storing measured accelerometer's data
- develop an application to storing received data in expansion memory module
- when the own ZigBee radio platform will be developed at Department of Control Engeneering, porting the open-ZB stack and all of created application to this new platform
- when the open-ZB stack porting to TinyOS 2.x will be done successfuly, porting all of created application to TinyOS 2.x and create an wireless sensor network for digital accelerometer's data acquisition by developed application for reading digital accelerometer's data

Appendix A

IEEE 802.15.4 Frame Structures

The diagrams in this clause illustrate the fields that are added by each layer of the protocol. The PHY packet structure represents the bits that are actually transmitted on the physical medium. A complete detailed description of the fields and frame's structures can be found in [3].

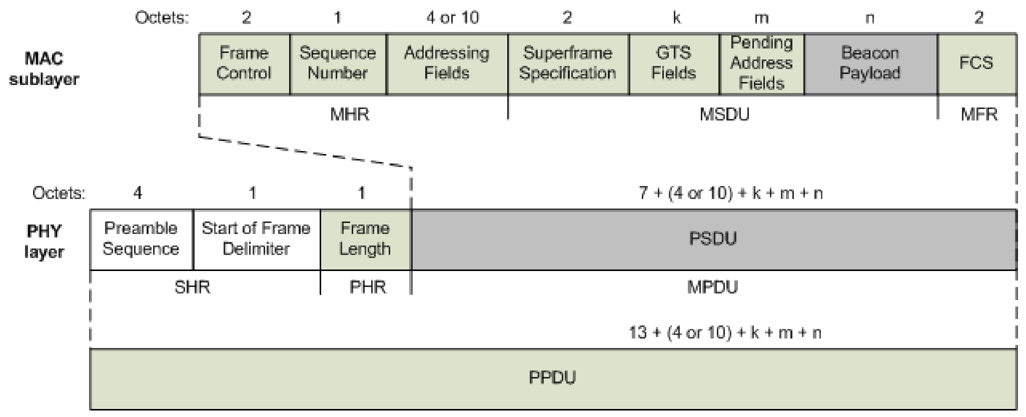


Figure A.1: Beacon frame

MHR	-	MAC header
MFR	-	MAC footer
MSDU	-	MAC service data unit
MPDU	-	MAC beacon/data/acknowledgment/command frame
FCS	-	frame check sequence
SHR	-	synchronization header
PHR	-	PHY header
PSDU	-	PHY service data unit (the PHY beacon/data/acknowledgment/command frame payload)
PPDU	-	PHY beacon/data/acknowledgment/command packet

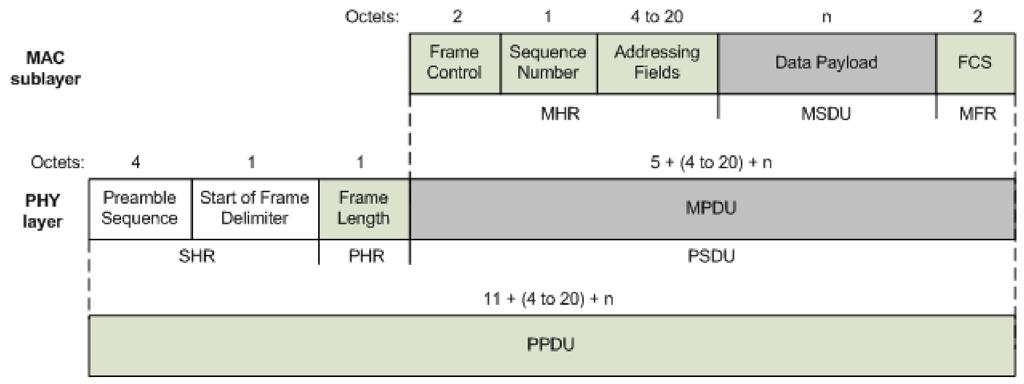


Figure A.2: Data frame

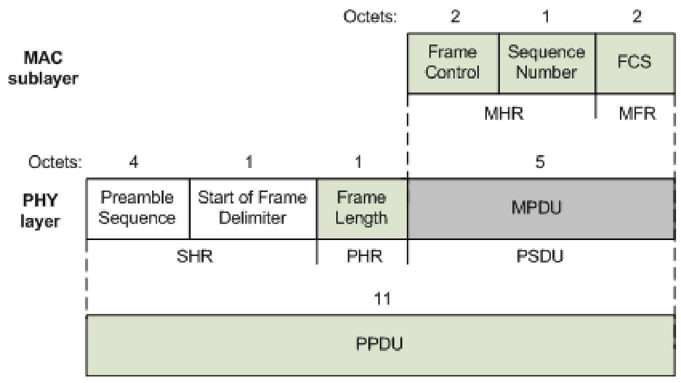


Figure A.3: Acknowledgment frame

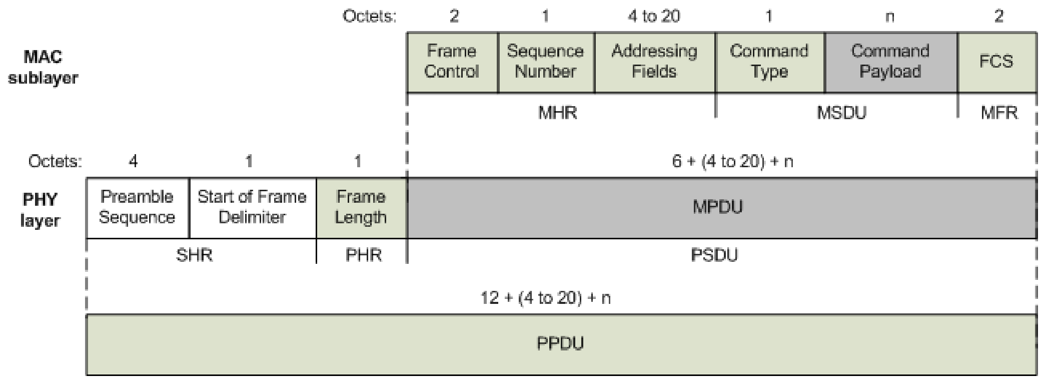


Figure A.4: MAC command frame

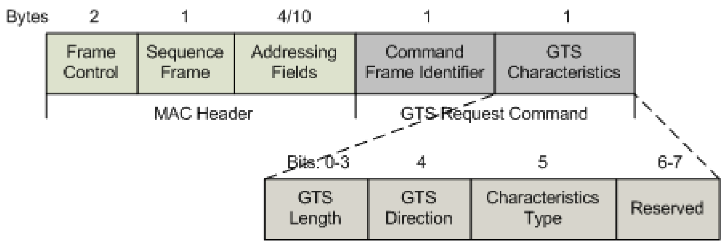


Figure A.5: GTS Request Command Frame

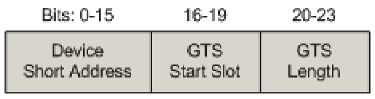


Figure A.6: GTS Descriptor

Appendix B

Electronical Design of Expansion Modules

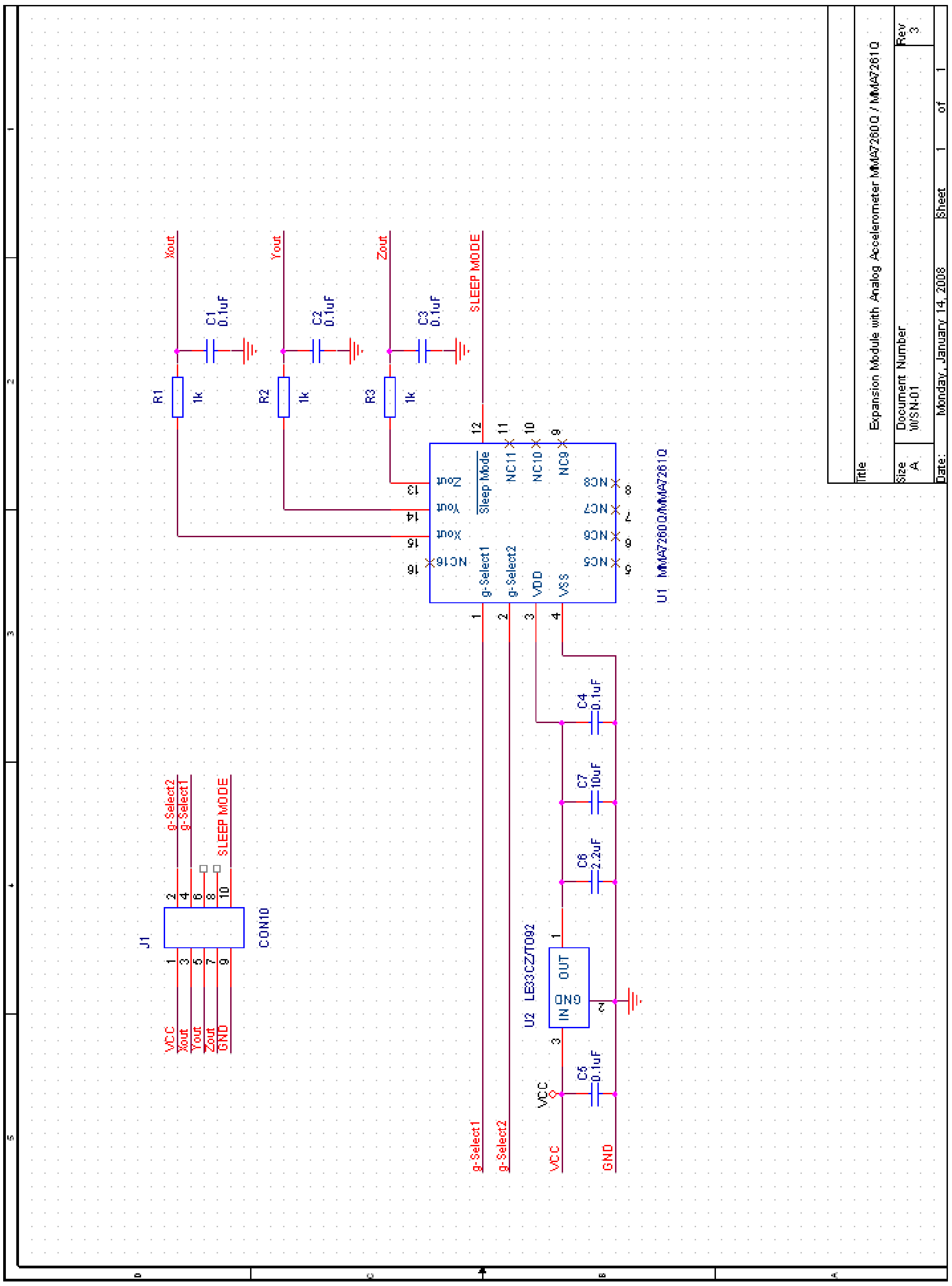


Figure B.1: Design of expansion module with MMA7260Q/MMA7261Q.

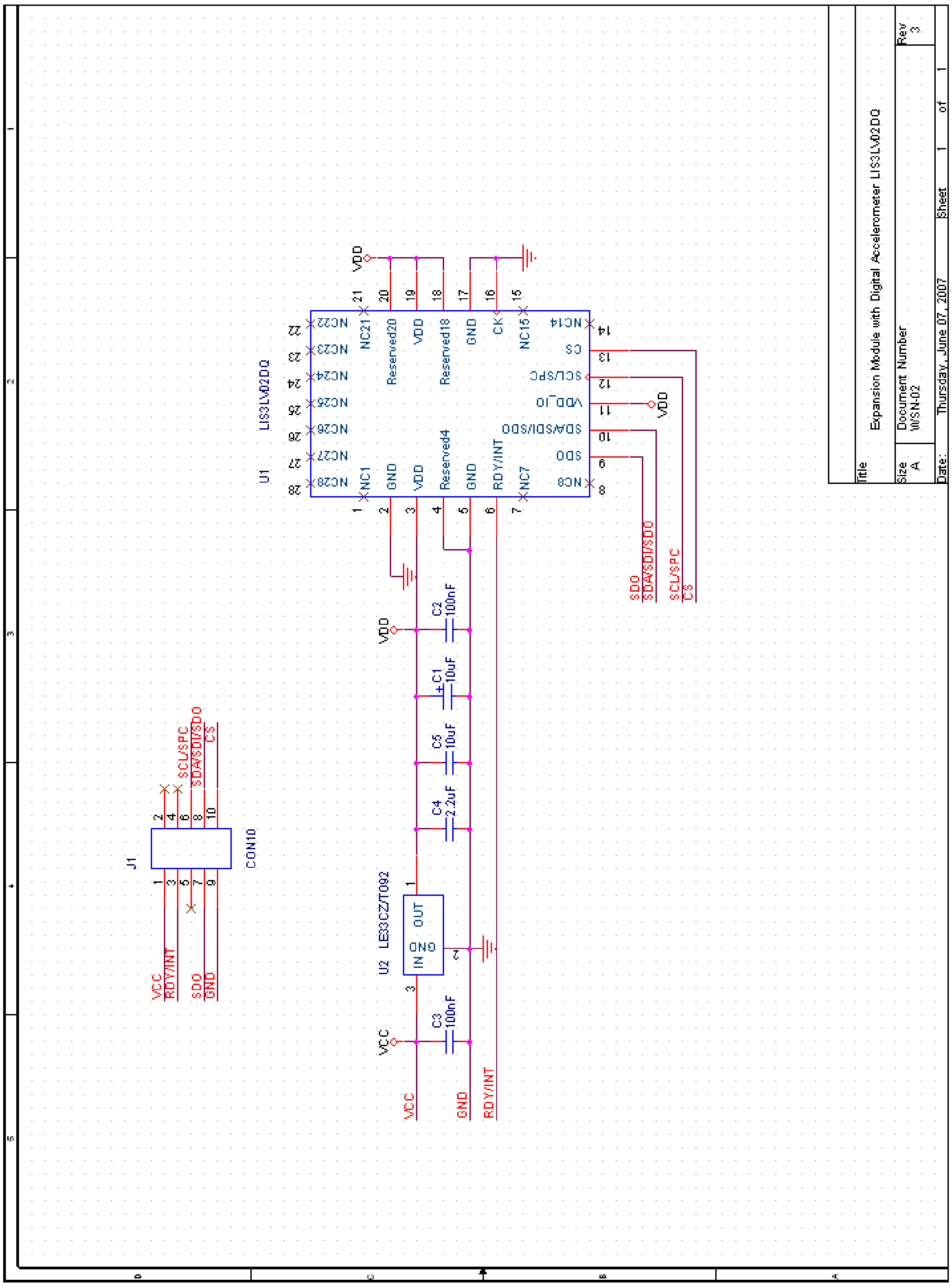


Figure B.2: Design of expansion module with LIS3LV02DQ.

Appendix C

Cygwin and TinyOS Install Instructions

(original article [45])

0. Uninstall existing TinyOS/Cygwin

If you have existing TinyOS or Cygwin installation, uninstall both of them.

Uninstall instructions: (or see website [44])

- Remove „TinyOS" program through control pannel. If you have used Tmote Sky tools disk, you should uninstall "tmote-tools.x.x" in control panel.
- Cygwin Uninstallation
- Remove the following items to fully uninstall cygwin (important):
  - Cygwin shortcuts and start menu entry (Programs/Cygwin)
  - Cygwin registry entries under `HKEY_LOCAL_MACHINE\SoftWare\Cygnus Solutions\` (run regedit or regedt32 to remove these)
  - Everything under the cygwin root directory. Save useful files of course, you could just rename the cygwin root to say, cygwin-old, to be extra safe.

It is possible that during installation is threw the message „The library cygwin1.dll already exist in `Windows/system32/`". It is also probably after an old installation of Cygwin. You will find it as a hidden file. Deleting it manually is recommended.

1. Download InstallShield from TinyOS website

- go to [46] to download Windows Installshield Wizard for TinyOS CVS Snapshot 1.1.11 [47] (less tested, but fresh); or download it from here [48].
- run tinyos-1.1.11-3is.exe
- select "complete" to install all components
- change install directory from `c:/program files/UCB` to `c:/tinyos`, then start installation - important, it stops problems with the space between words in „program files”.
- after the installation is finished, you should get the following directory tree:

```
C:/tinyos/ATT
  /cygwin
  /cygwin-installationfiles
  /jdk1.4.1_02
```

2. Test Compilation Process

- go to `c:/tinyos/cygwin/tinyos-1.x/apps/Blink` (you can choose any other application to do the testing), and test compiling for different platforms by typing:
- `$ make telosb` //should work without any error
- `$ make micaz` //should work without any error

3. Test downloading application into TelosB

- plug Tmote Sky/TelosB mote to USB port on PC
- install driver using Tmote Sky CD or using the driver file downloaded from here [49]
- `$ motelist` //to check COM number
- `$ make telosb`
- `$ make telosb reinstall,7 bsl,9` //ID = 7, Com port = 10 - while using single mote in USB it can be „`$ make telosb reinstall`”
- during the transmission, transmit LED should blink

4. Customize TinyOS for your project

An example:

- copy your project directory `xxx` into "`c:\tinyos\cygwin\opt\tinyos-1.x\apps\`" - remember to include hidden files too by enabling "show all hidden and system files" in File Explorer
- set/create HOME path:

  My computer→Properties→Advanced→Environment Variables→"New" or "Edit" HOME variable as "`c:\tinyos\cygwin\opt\tinyos-1.x\apps\xxx`" or "`/opt/tinyos-1.x/apps/xxx`"
- close Cygwin and restart a Cygwin window

5. Check installation completeness

Run command: `$ toscheck`

If it reports any error, follow the instruction to correct it.

One of the errors will be probably:

"I have installed TinyOS 1.1.11 and get this error when I run toscheck "WARNING: CLASSPATH may not include '.' (that is, the symbol for the current working directory). Please add '.' to your CLASSPATH or you may experience configuration problems." I have checked and I do have '.' in my path."

After the final upgrade of Cygwin this error disappear.

6. Update to TinyOS 1.1.15

(see [50])

Download the 1.1.15 rpm: [51]

Install. As Administrator in a cygwin shell do:

```
$ rpm –force –ignoreos -Uvh tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm
```

in the directory where you saved the rpm. This will take a while (the tinyos package installation includes compiling the java code). TinyOS is installed in `/opt/tinyos-1.x`.

7. Upgrade nesC compiler to newest version

Download [52] to `/opt/` directory.

Install nesc compiler:

```
$ tar -zxvf nesc-1.2.7a.tar.gz
$ cd nesc-1.2.7a
$ ./configure
$ make install
$ ncc -v
```

It should shows the correct nesC version 1.2.7.

8. Upgrade Graphviz to newest version

Run the following command to creation of documentation:

```
$ make telosb docs
```

Is not working? Then run the command:

```
$ dot -V
```

If it does not show *"dot version 2.8"* or higher than 1.8.8, then you need upgrade.
Steps to upgrade:

- remove existing Graphviz program using control panel
- download it here: [53]
- install it to `c:\tinyos\ATT`

If you have the **Matlab** installed, you might find your upgrade does not work. The reason is file conflict because the Matlab has a `dot.exe` too. Then check environmental variables $PATH, put `C:\TinyOS\ATT\Graphviz` to ahead of MATLAB's path.

9. Upgrade Cygwin to support gcc 3.4.4

Type the following command:

```
$ gcc -v
```

If it show that:

```
/opt/oasis>gcc -v
Reading specs from ...
...
gcc version 3.3.3 (cygwin special)
```

$\rightarrow$version is 3.3.3. Then it is need upgrade to newest version.
The way to do it is to upgrade Cygwin:

- close all Cygwin windows
- download [54] and run it.

- choose install from Internet
- change Devel, X11 and Graphics from default to install
- click OK on all warnings

Very long process! Very much hard disc space consumption. Entire folder of TinyOS will have over 2,5GB after this installation process.

You should expect following now:

```
/opt/oasis>gcc -v
Reading specs from ...
...
gcc version 3.4.4 (cygming special) (gdc 0.12, using dmd 0.125)
```

10. Final check installation completeness

Run command: `$ toscheck`. It might say:

```
   toscheck completed with errors:
   --> WARNING: The graphviz (dot) version found by toscheck is not 1.10.
Please up date your graphviz version if you'd like to use the nescdoc
documentation genera tor.
```

Ignore it if only so. We actually use higher version 2.8 of graphviz.

Appendix D

All created Application Install Instructions

In the following it is assumed that you have already installed TinyOS 1.x. We will refer to the root of your TinyOS source directory as TOSROOT.

1. Open source implementation of IEEE 802.15.4/ZigBee (open-ZB stack) installation:

Unzip the "*hurray1.2.zip*" file and copy the "hurray" folder into: `TOSROOT\contrib\` folder.

2. DataSendAccel application

- Unzip the "*DataSendAccel.zip*" file and copy the "DataSendAccel" folder into:
  `TOSROOT\contrib\hurray\apps` folder
  that contains all the example applications of the open-ZB stack.
- Open "Cygwin" environment, in Cygwin go to the appropriate directory where the "DataSendAccel" application you want to load is located (see above).
- Attach the Tmote Sky device to the USB port. When plug a mote to the USB port for the first time, Windows automatically assigns it a COM port number, and it will use this number whenever plug the same mote again.
- Can check which number the mote is assigned by typing:
  ```
  $ motelist
  ```

- To compile the application and generate all the binary code needed in order to run the application in a Tmote Sky module simply typing the bellow command:
  $ make telosb
  in the application directory.

The overall compilation process is like this: All the .nc files are translated into a C file (`build`/app.c) by the nesC compiler. This file needs to be translated into a binary file. Normally for TinyOS, this is done by a gcc backend for the MCU.

Other targets are available:

1. $ make telosb install,<node_id> <programmer_platform>,<port> - to install the application on the sensor mote, the new application replaces the last one.
   <node_id> is an integer between 0 and 255 and specifies the node ID (the address assigned to the mote), make sure it's unique
   <programmer_platform>,<port> Tmote Sky is programmed by msp430-bsl bootstrap loader. <port> is the serial port number where the mote is plugged minus one. Therefore, <programmer_platform> = bsl and it programs the mote on COM(<port>+1)
2. $ make telosb reinstall <programmer_platform>,<port> - to install the application on the sensor mote without recompiling (it skips recompilation), the new application replaces the last one.
3. $ make telosb docs - this target creates documentation for the program you have compiled. You need the graphviz/dot package to get the component graps. The nesdoc documentation files are generated under the `doc/nesdoc` directory.

3. LIS3LV02DQ application

- Make a directory in: `TOSROOT\apps\Your_Directory_Name` (`TOSROOT\apps\` contains most of TinyOS 1.x example applications)
- Unzip the "*LIS3LV02DQ.zip*" file and copy the "LIS3LV02DQ" folder into: `TOSROOT\apps\Your_Directory_Name folder`.
- Open "Cygwin" environment, in Cygwin go to the appropriate directory where the "LIS3LV02DQ" application you want to load is located (see above).
- Attach the Tmote Sky (TelosB platform) to the USB port.
- To compile the application typing:
  $ make telosb
  in the application directory.

Other targets:

```
$ make telosb install,<node_id> bsl,<port>
$ make telosb reinstall bsl,<port>
$ make telosb docs
```

4. spZB application in C language

- For example make a directory in: `tinyos\cygwin\Your_Directory_Name_C` or `tinyos\cygwin\home\Your_Directory_Name_C`
- Unzip the "*spZB.zip*" file and copy the "spZB" folder into the created "`Your_Directory_Name_C`" directory.
- Open "Cygwin" environment, in Cygwin go to the appropriate directory where the "spZB" application you want to executed is located (see above).
- To compile the application typing:
  ```
  $ gcc -o sp sp.c
  ```
  in the application directory.
- To execute the application typing:
  ```
  $ ./sp <port>
  ```
  in the application directory.

5. SerialComZB application in C# language

- For example make a directory in: `\home\Your_Directory_Name_Csharp`
- Unzip the "*SerialComZB.zip*" file and copy the "SerialComZB" folder into the created "`Your_Directory_Name_Csharp`" directory.
- To execute the application run the file SerialComZB.exe in `\home\Your_Directory_Name_Csharp\Release` directory.

Appendix E

Contents of the CD-ROM

The included CD-ROM contains:

- `/Documentation/`: Diploma thesis in PDF format.
- `/Documentation/Datasheet/`: Datasheets in PDF format for components used in design.
- `/HW/LIS3LV02DQ/`: Printed Circuit Board design of expansion module with digital accelerometer LIS3LV02DQ, data in OrCAD format.
- `/HW/MMA7260Q/`: Printed Circuit Board design of expansion module with analog accelerometer MMA7260Q, data in OrCAD format.
- `/SW/DataSendAccel/`: Application to data acquisition by modules with analog accelerometer MMA7260Q, TinyOS source code for the base station and sensor node.
- `/SW/I2CAccelerometer/`: Application to data read from modules with digital accelerometer LIS3LV02DQ, TinyOS source code.
- `/SW/OPEN-ZBstack/`: Implementation of IEEE 802.15.4/ZigBee in TinyOS/nesC v1.2.
- `/SW/SerialComZB/`: GUI application in C#/MS Visual Studio 2005, C# source code.
- `/SW/spZB/`: Console application in C under Cygwin, C source code.
- `/USBDriver/`: Driver to support FTDI USB controller on Tmote Sky sensor module.

Bibliography

[1] Otakar Šprdlík, *Sensor network for inertial data acquisition*, 2006. http://rtime.felk.cvut.cz/hw/images/e/e1/Spec_ParkinsonsDisease.pdf

[2] Antonín Vojáček, *ZigBee - novinka na poli bezdrátové komunikace*, June 2005. ZigBee-novinkanapolibezdrátovékomunikace

[3] IEEE 802.15 WPAN™ Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html

[4] ZigBee Alliance. http://www.zigbee.org/en/

[5] Anis Koubâa, Mário Alves, Eduardo Tovar, *IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview*, version 1.0, July 2005. http://www.open-zb.net/publications/hurray-tr-050702.pdf

[6] André Cunha, Mário Alves, Anis Koubâa, An IEEE 802.15.4 protocol implementation (in nesC/TinyOS): Reference Guide v1.2, May 2007. http://www.open-zb.net/publications/HURRAY_TR_061106_An_IEEE_802.15.4_protocol_implementation%20_in_nesCTinyOS_%20Reference_Guide_v1.2.pdf

[7] André Cunha, Martin Auersvald, *Main modifications of Open-ZB stack*, 2007. http://rtime.felk.cvut.cz/wsn/

[8] Direct-Sequence Spread Spectrum modulation technique (DSSS). http://en.wikipedia.org/wiki/Direct-sequence_spread_spectrum

[9] Pete Cross, *Zeroing in on ZigBee (Part 1): Introduction to the Standard*, Circuitcellar, February 2005. http://www.circuitcellar.com/library/print/0205/Cross175/Cross-175.pdf

[10] Sinem Coleri Ergen, *ZigBee/IEEE 802.15.4 Summary*, September 2004. http://pages.cs.wisc.edu/~suman/courses/838/papers/zigbee.pdf

[11] ZigBee Specification 2006. http://www.zigbee.org/en/spec_download/zigbee_downloads.asp

[12] Patrick Kinney, *ZigBee Technology: Wireless Control that Simply Works*, Kinney Consulting LLC, 2003.

[13] Jacob Munk-Stander, Martin Skovgaard, Toke Nielsen, *Implementing a ZigBee Protocol Stack and Light Sensor in TinyOS*, Bachelor's Thesis, October 2005. `http://www.diku.dk/~bonnet/ba.zigbee.pdf`

[14] David Scherba, Peter Bajcsy, *Communication Models for Monitoring Applications Using Wireless Sensor Networks*, Technical Report, April 2004. `http://algdocs.ncsa.uiuc.edu/TR-20040401-1.pdf`

[15] College of Engineering, UC Berkeley, Electrical Engineering and Computer Sciences. `http://www.eecs.berkeley.edu/`

[16] TinyOS. `http://www.tinyos.net/`

[17] Sourceforge, TinyOS. `http://sourceforge.net/projects/tinyos/`

[18] nesC: A Programming Language for Deeply Networked Systems. `http://nescc.sourceforge.net/`

[19] Sourceforge, nescc. `http://sourceforge.net/projects/nescc/`

[20] David Gay, Philip Levis, David Culler, Eric Brewer, *nesC 1.1 Language Reference Manual*, May 2003. `http://nescc.sourceforge.net/papers/nesc-ref.pdf`

[21] Sentilla/Moteiv Corporation. `http://www.sentilla.com/`

[22] Crossbow Technologies. `http://www.xbow.com/`

[23] Sentilla/Moteiv Corporation, Tmote Sky Datasheet. `http://www.sentilla.com/moteiv-endoflife.html`

[24] Texas Instruments, MSP430F1611 Datasheet and MSP430x1xx Family User's Guide. `http://ti.com/msp430`

[25] FTDI Chip drivers. `http://www.ftdichip.com/FTDrivers.htm`

[26] Gessler Electronic, MSP430-bsl, MSP430 Flash Programming Toolkit. `http://www.gessler-electronic.com/msp430/`

[27] Texas Instruments/Chipcon, CC2420 Datasheet. `http://focus.ti.com/docs/prod/folders/print/cc2420.html`

[28] Freescale Semiconductor, MMA7260Q Datasheet. `http://www.alldatasheet.com/datasheet-pdf/pdf/103487/MOTOROLA/MMA7260Q.html`

[29] Freescale Semiconductor, MMA7261Q Datasheet. http://www.alldatasheet.com/datasheet-pdf/pdf/133711/FREESCALE/MMA7261Q.html

[30] STMicroelectronics, LE33CZ Datasheet. http://www.alldatasheet.com/datasheet-pdf/pdf/22791/STMICROELECTRONICS/LE33CZ.html

[31] Philips Semiconductors, The I^{2}C-Bus Specification, version 2.1, January 2000. http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

[32] STMicroelectronics, LIS3LV02DQ Datasheet. http://www.alldatasheet.com/datasheet-pdf/pdf/117096/STMICROELECTRONICS/LIS3LV02DQ.html

[33] Jeongyeup Paek, *The Time library for Rate-Controlled Reliable Transport protocol for Wireless Sensor Networks.* http://enl.usc.edu/cgi-bin/viewcvs/viewcvs.cgi/tenet/mote/lib/timer/

[34] Inderjit Singh, *Temperature reading application*, February 2007. http://rtime.felk.cvut.cz/hw/index.php/Temperature_reading_application

[35] Inderjit Singh, *Real-time Object Tracking with Wireless Sensor Networks*, Diploma Thesis, August 2007. http://epubl.ltu.se/1653-0187/2007/059/LTU-PB-EX-07059-SE.pdf

[36] Prof. Andrea Nannini, Pasquale Sestito, Ing. Francesco Pieri, *Sviluppo di una interfaccia di lettura per un accelerometro MEMS triassiale inserito in una rete di sensori*, Tesi di Laurea, Accademico 2005/2006. http://etd.adm.unipi.it/theses/available/etd-11222006-104534/unrestricted/tesi.pdf

[37] Moteiv Community, *Connecting External Sensors*, 2007. http://www.moteiv.com/community/Connecting_External_Sensors

[38] UC Berkeley, *Mail-Archive TinyOS-Help.* http://www.mail-archive.com/tinyos-help@millennium.berkeley.edu/

[39] Codeproject, *Hexadecimal, Binary, and Decimal conversions.* http://www.codeproject.com/KB/dotnet/BinaryAndHexConversions.aspx

[40] SourceForge, Download ZedGraph. https://sourceforge.net/projects/zedgraph/

[41] SourceForge, ZedGraph Main Page. http://zedgraph.sourceforge.net/

[42] Wikipedia, GNU Lesser General Public License. http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License

[43] Kim Hamilton, C# Top 5 SerialPort Tips, 2006. http://blogs.msdn.com/bclteam/archive/2006/10/10/Top-5-SerialPort-Tips-_5B00_Kim-Hamilton_5D00_.aspx

[44] Uninstalling TinyOS. `http://www.tinyos.net/tinyos-1.x/doc/uninstall.html`

[45] Sensorweb Vancouver, TinyOS Start Guide. `http://sensorweb.vancouver.wsu.edu/wiki/index.php/Tinyos`

[46] TinyOS Windows Installshield Wizard and Installation. `http://www.tinyos.net/windows-1_1_0.html`

[47] TinyOS download, Wireless Embedded Systems, UC Berkeley. `http://webs.cs.berkeley.edu/users/users.php?download=1&snapshot=1`

[48] Sensorweb Vancouver, TinyOS 1.1.11-3is Download. `http://sensorweb.vancouver.wsu.edu/software/tinyos-1.1.11-3is.exe`

[49] FTDI Chip Drivers VCP Download. `http://www.ftdichip.com/Drivers/VCP.htm`

[50] Upgrading to TinyOS 1.1.15 CVS Snapshots, December 2005. `http://www.tinyos.net/dist-1.1.0/snapshot-1.1.15Dec2005cvs/doc/install-snapshots.html`

[51] Tinyos 1.1.15 RPM package for Cygwin Download, UC Berkeley, December 2005. `http://webs.cs.berkeley.edu/tos/dist-1.1.0/tinyos/windows/tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm`

[52] Sourceforge, nesc 1.2.7a package Download. `http://prdownloads.sourceforge.net/nescc/nesc-1.2.7a.tar.gz?use_mirror=umn`

[53] Graph Visualization Software, Graphviz 2.8 Download. `http://www.graphviz.org/pub/graphviz/ARCHIVE/graphviz-2.8.exe`

[54] Cygwin, Install Setup Download. `http://cygwin.com/setup.exe`