BAKKALAUREATSARBEIT

IEEE 802.15.4 MAC API

ausgeführt am Institut für Rechnergestützte Automation
zum Zwecke der Erlangung des akademischen Grades
eines Bakkalaureus der Technischen Informatik

unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
und
Dipl.-Ing. Mag. Christian Reinisch

durchgeführt von

Lukas Krammer
Matr.-Nr. 0525332
Gumpendorferstraße 140/13c, 1060 Wien

Wien, am 26. Mai 2008 .....

IEEE 802.15.4 MAC API

Kurzfassung

In den letzten Jahren wurden viele verschiedene Funktechnologien entwickelt und haben sich auf dem Markt etabliert. Diese sind jedoch hauptsächlich auf Multimedia Anwendungen bzw. auf die Übertragung großer Datenmengen ausgelegt. Im Gegensatz dazu wurde IEEE 802.15.4/ZigBee entwickelt, das den Fokus hauptsächlich auf die Übertragung von Kontrolldaten legt.
In dieser Arbeit werden IEEE 802.15.4/ZigBee Implementierungen verschiedener Hersteller und deren Eigenschaften diskutiert. Darüberhinaus wird ein kurzer Einblick in die Grundkonzepte von IEEE 802.15.4 gegeben. Das Hauptziel dieser Arbeit ist die Implementierung des MAC Sublayers des IEEE 802.15.4 Stacks, wobei Teile des Stacks von Texas Instruments verwendet werden. Die Implementierung wurde notwendig, da ein großer Teil des IEEE 802.15.4 Stacks von Texas Instruments nur als Library für einen bestimmten Compiler veröffentlicht wurde, jedoch nicht als Quelltext. Ein wichtiges Ziel ist es auch, eine kompatible Schnittstelle zu der ursprünglichen Implementierung von Texas Instruments zu schaffen. Dies ist notwendig um gegebenenfalls den ZigBee Stack von Texas Instruments auf die aktuelle Implementierung aufzusetzen.
Als RF Transceiver wird ein Chipcon CC2420 verwendet, der Teile des Physical Layers (PHY) von IEEE 802.15.4 bereits integriert hat. Dieser Chip ist mit einem MSP430 Microcontroller von Texas Instruments verbunden, auf welchem der Stack sowie die Applikation implementiert sind.
Abschließend wird in dieser Arbeit die Funktion des IEEE 802.15.4 Stacks mithilfe einer kurzen Applikation demonstriert.

IEEE 802.15.4 MAC API

Abstract

In the last few years a large number of wireless technologies was established, but most of them focus on multimedia applications with capabilities to transmit high amount of data. In difference to these technologies the IEEE 802.15.4/ZigBee standard is well suited for wireless control applications.
This thesis discusses different implementations of IEEE 802.15.4/ZigBee and illustrates the basic concepts of IEEE 802.15.4.
The main focus of this thesis is the implementation of the medium access control (MAC) sublayer of the IEEE 802.15.4 stack by using parts of the IEEE 802.15.4 stack of Texas instruments. This implementation is necessary because Texas Instruments published parts of the IEEE 802.15.4 stack only as library for a special compiler and not as source code.
The primary goal of this implementation is the compliance of the current implementation with the implementation of Texas Instruments because it should be possible to set up the ZigBee stack (network layer NWK and above) from Texas Instruments.
The stack uses a smart RF transceiver (Chipcon CC2420) which provides parts of the physical layer PHY layer of the IEEE 802.15.4 stack. As target device a MSP430 microcontroller unit (MCU) is used, which is connected to the CC2420 trough a serial data interface.
Finally this thesis demonstrates the functionality of the IEEE 802.15.4 implementation in a short demo application.

Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. Objectives . . . 2
- 1.3. Related Work . . . 2
- 1.4. Structure of the Thesis . . . 3

2. Concepts **4**
- 2.1. Introduction in IEEE 802.15.4 . . . 4
  - 2.1.1. Introduction . . . 4
    - 2.1.1.1. Device Types . . . 4
    - 2.1.1.2. Network Topology . . . 5
  - 2.1.2. PHY Layer . . . 6
    - 2.1.2.1. Frequency Bands . . . 6
    - 2.1.2.2. Transmit Power . . . 7
    - 2.1.2.3. Physical frame structure . . . 8
  - 2.1.3. MAC Sublayer . . . 8
    - 2.1.3.1. Introduction . . . 8
    - 2.1.3.2. Communication Modes . . . 9
    - 2.1.3.3. Superframe Structure . . . 10
    - 2.1.3.4. CSMA-CA . . . 10
    - 2.1.3.5. Starting a personal area network (PAN) . . . 12
    - 2.1.3.6. Association and Disassociation . . . 14

3. Implementation **15**
- 3.1. Hardware . . . 15
  - 3.1.1. Hardware Description . . . 15
  - 3.1.2. Interface Description . . . 17
- 3.2. Porting of the hardware abstraction layer (HAL) . . . 19
  - 3.2.1. Changing Macros . . . 19
  - 3.2.2. Changing Registers . . . 20
  - 3.2.3. Changing the clock source . . . 20
- 3.3. MAC Stack Overview . . . 20
  - 3.3.1. Provided Functions of the TI Stack . . . 22
    - 3.3.1.1. Basic Radio Functions . . . 22
    - 3.3.1.2. Receive and Transmit Functions . . . 23
    - 3.3.1.3. Timer Functions . . . 23

3.3.2. Source File Structure 25
3.4. MAC API Description 26
3.4.1. General Event Structure 26
3.4.2. Memory Management 27
3.4.3. MAC Data Service 28
3.4.3.1. MCPS Data Request 28
3.4.3.2. MCPS Purge Request 31
3.4.3.3. MCPS Data Allocate 31
3.4.4. Management Services 32
3.4.4.1. MLME Association 32
3.4.4.2. MLME Disassociation 35
3.4.4.3. MLME Start 36
3.4.4.4. MLME Beacon Notify 38
3.4.4.5. MLME Get 40
3.4.4.6. MLME Set 40
3.4.4.7. MLME GTS 41
3.4.4.8. MLME Reset 43
3.4.4.9. MLME RX Enable 43
3.4.4.10. MLME Scan 43
3.4.4.11. MLME Poll 47
3.5. Demo Application 48
3.5.1. Test environment 48
3.5.2. Beacon-enabled Demo Application 50
3.5.3. guaranteed time slot (GTS) verification 52
3.5.4. Non beacon-enabled Demo Application 53

4. Conclusion **55**

Bibliography **56**

A. Acronyms and abbreviations **57**

B. Board Documentation **59**
B.1. Documentation of the RF Board 59
B.2. Schematic of the MCU Board 61

1. Introduction

1.1. Motivation

In the last few years the number of electrical and electronic devices in a household or a functional building raised very fast. Nowadays the term 'smart building' becomes more and more important. Such systems also require smart communication systems which connect the devices together in a cheap and efficient way. In such systems wireless technologies may be preferred instead of wired technologies, because they are more flexible and do not need expensive wire connections.

Actual wireless technologies are only for a small group of products or for products of one manufacturer. Moreover many of these technologies, like Wireless LAN (IEEE 802.11) or Bluetooth (IEEE 802.15.1) have the focus on multimedia applications, like audio/video streaming. However they do not have the focus on control applications nor on security or real time applications. ZigBee and IEEE 802.15.4 ([1]) are protocol standards, which focus exactly on this market segment. Hence ZigBee is an ideal technology for control applications in the home and building area, like lighting control or heating, ventilating, and air conditioning (HVAC). Because of its security capabilities it's also ideal for security applications like alarm systems or door control systems. Moreover it's possible to transmit also audio or video data, but these tasks are not the main application area of this technology.
Since there is a large distance between the sensors and actuators in a building, ZigBee includes also a routing mechanism. This mechanism makes the communication between devices possible, which are not directly connected .
Another advantage of this technology is the low power consumption. This feature is especially important for battery powered devices, like remote controls.
Since a building consists of a large number of sensors and actuators, the expense factor is very important. That is the reason why ZigBee/IEEE 802.15.4 is a very slim standard compared to Bluetooth (IEEE 802.15.1)or wireless local area network (WLAN) (IEEE 802.11), so ZigBee devices need less complex hardware and so they are much cheaper than other wireless products.

1.2. Objectives

There are a lot of manufacturers, which provide ZigBee stacks and ZigBee transceivers, as listed in Chapter 1.3. For the actual project the RF transceiver Chipcon CC2420 in combination with the ZigBee/IEEE 802.15.4 Stack from Texas Instruments were used. The ZigBee/IEEE802.15.4 Stack is implemented in a Texas Instruments MSP430, which is a low cost MCU with a very low power consumption. This stack is written in C and originally built with the proprietary IAR Compiler. Since the IAR Compiler is a commercial compiler and a license is very expensive the GNU C compiler for MSP430 MCUs (mspgcc) was used in this project.
Since the MCU, which was used by Texas Instruments to implement the stack, has only one universal asynchronous receiver transmitter (UART), another type of MSP430 MCU was used, which contains two UARTs. To set up an experimental environment, a development board from TI with the MSP430F149 on it was connected to a RF development board from Microchip which hosts a Chipcon CC2420. The first idea was to take the original stack and only change a few compiler specific code segments and a few register definitions to make the stack running in our development environment using the mspgcc. After changing these few things it focused out, that elementary parts of the stack are only represented in a static library for the IAR compiler. There were only a few low level primitives provided by Texas Instruments, but not the complete IEEE 802.15.4 stack. Moreover trials to convert the static library with binary utilities failed and also an intensive search for these missing files failed. Finally the only possibility to use this stack, was to implement the missing parts of the stack and so the the MAC layer of the IEEE 802.15.4 standard was implemented.

Therefore the focus of this work is the implementation of the IEEE 802.15.4 stack using existing parts of the stack from Texas Instruments. Another very important requirement is to design the self implemented stack fully compatible to the original stack from Texas Instruments. This is so important, because in future projects the higher layers of the stack (Network layer (NWK), Application layer (APL) of ZigBee) should be set up on the actual implementation

1.3. Related Work

The ZigBee/IEEE 802.15.4 standard is used and implemented by many different suppliers, but since the ZigBee standard is an open standard, the

devices of the different supplier should by compatible to each other.

In this section a short overview of the most important and interesting implementations of IEEE 802.15.4/ZigBee is given.
The first implementation which is mentioned is the ZigBee stack from Atmel. Atmel produces an own Radio Transceiver, which is compliant to the IEEE 802.15.4 standard. The function of this RF transceiver (Atmel AT86RF230) is quite similar to the Chipcon CC2420. The stack of Atmel is easier than the of Texas Instruments, but primitives for task handling and messaging are not included. A main advantage of the stack from Atmel is the support of GTS, because GTS is not supported in the stack from Texas Instruments. Microchip also published a ZigBee stack, which uses the same RF transceiver as Texas Instruments. Unfortunately this stack do not support GTS as the stack from Texas Instruments.
Open-ZB is a community driven implementation of a ZigBee stack for TinyOS. This implementation is licensed under the Academic Free License (AFL) and does not necessarily need a complex RF transmitter e.g. Chipcon CC2420 or Atmel AT86RF230, nevertheless such chips are also supported.

1.4. Structure of the Thesis

This thesis is structured as follows: Chapter 2 gives a short introduction in the IEEE 802.15.4 standard, including the basic operations and primitives. The first part of Chapter 3 describes the hardware arrangement of the development environment, including a detailed description of the interface between the MCU and the RF transceiver. The second part of this chapter describes the software interface of the MAC layer and the restrictions for the use of this API. Chapter 4 summarizes the thesis and gives an outlook on possible applications with the results of this thesis and general applications, which are possible with ZigBee.

2. Concepts

2.1. Introduction in IEEE 802.15.4

2.1.1. Introduction

This chapter gives a short introduction in the IEEE 802.15.4 and ZigBee standard. The ZigBee protocol stack can be described by using the International Organization for Standardization (ISO) - Open Systems Interconnection (OSI) model as shown in Figure 2.1. The actual ZigBee standard ([2]) is located on the NWK and APL layer of the OSI model and above. ZigBee is based on IEEE 802.15.4 standard ([1]), which defines the physical layer (PHY) and MAC layer of the OSI model. The IEEE 802.15.4 standard stack is not necessarily used in combination with ZigBee, because it is also possible to set up other communication protocols like Internet Protocol (IP) or fieldbus systems like Wireless Highway Addressable Remote Transducer (HART). Moreover an interesting fact is, that the ZigBee standard is defined and maintained by the ZigBee Alliance, while 802.15.14 is maintained by IEEE.

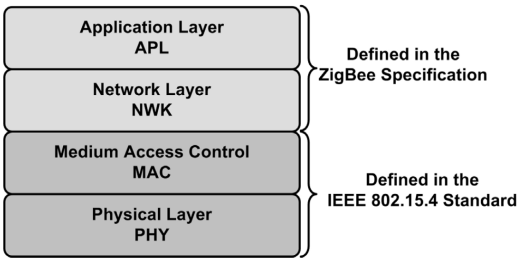

Figure 2.1.: ZigBee/IEEE 802.15.4 protocol stack architecture (taken from [3])

2.1.1.1. Device Types

In the IEEE 802.15.4 standard two different device types are defined:

- Full-Function Device (FFD)
  Such a device has implemented the whole primitives and functions, which are defined in the standard. A full-function device (FFD) is able to act

as coordinator, which provides beacon transmission or to act as simple device.
Hence such a device should be able to act as coordinator, more memory capacity is needed and as a consequence of that, the power consumption and the hardware costs are higher.

- Reduced-Function Device (RFD)
  In contrast to a FFD, the reduced-function device (RFD) only consists of the minimum implementation of the standard. So it is not possible for such a device to act as coordinator. Moreover it is only possible to connect it to one single FFD.
  Because of the low power consumption and the low hardware requirements, these devices are quite cheaper than FFDs and are used as e.g., small sensors or actuators.

The distinction between these two device types offers the way to design systems in a very flexible and economic way.

2.1.1.2. Network Topology

The IEEE 802.15.4 standard offers a few different types of network topologies:

- Star Topology
  In such a topology as shown in Figure 2.2, one FFD acts as PAN coordinator and all devices (either a FFD or RFD) in the area of the coordinator can associate with it. If a device wants to communicate with another device in the PAN, it has to send the message to the coordinator, which forwards the message to the destination device.

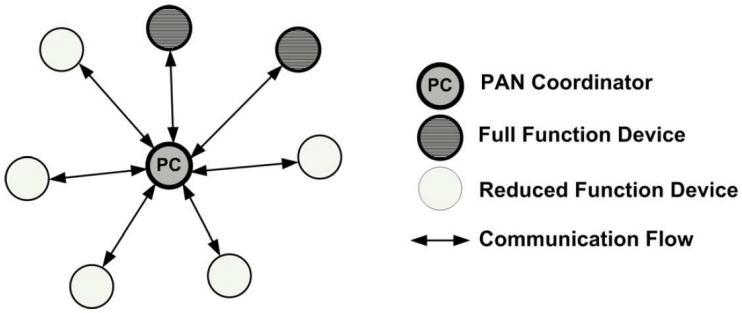

Figure 2.2.: Structure of a Star Topology (taken from [3])

- Peer-to-Peer Topology
  In a peer-to-peer network (as shown in Figure 2.3), basically each device can communicate with each other device in its environment directly. If one device wants to communicate with a device, which is not in the

immediate environment, a routing mechanism can be introduced. This mechanism should forward messages through a number of routing devices to the destination device, with routing is performed by FFDs. Such a routing mechanism is not in the scope of the IEEE 802.15.4 standard, but it is implemented in the NWK layer of ZigBee.

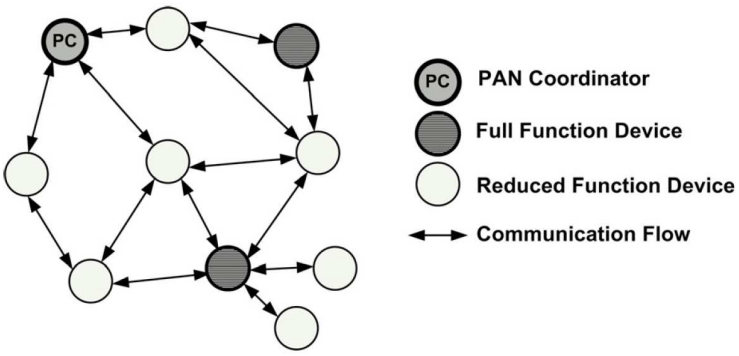

Figure 2.3.: Structure of a Peer-to-Peer Topology (taken from [3])

2.1.2. PHY Layer

The PHY of IEEE 802.15.4 is used to transmit a basic data frame over the physical medium. Moreover the PHY layer is responsible for modulation and encoding. It also defines frequency band and transmit power.

2.1.2.1. Frequency Bands

Different frequency bands and modulation methods are defined in this standard which makes it possible to use this standard in many regions and countries allover the world. The following list gives a short overview about the different frequency bands and the distribution area of them:

- The frequency band around 868,3 MHz is especially defined for Europe.
- The frequency band around 915 MHz is used in the USA.
- The frequency band around 2,46 GHz can be used everywhere around the world.

The most popular frequency band in our region is the 2.46 GHz band. This frequency band is also used by many other wireless technologies and applications. Bluetooth and wireless LAN are the most popular technologies in this frequency band. The use of Bluetooth (IEEE 802.15.1) or Wireless local area network (LAN) (IEEE 802.11) in combination with IEEE 802.15.4

in the 2,46GHz band causes some problems, like packet errors, which are described in [4].

2.1.2.2. Transmit Power

The physical layer also specifies the transmit power, whether the maximum physical transmit power depends on the frequency band and the local regulations. The value of the transmit power is not a fixed value, but a variable value, which also can be set by higher layers. The major advantage of an adjustable transmit power is the reduction of power consumption. So if e.g., two communication devices are only a few centimeters away from each other, it is not necessary to transmit a frame with the maximum power. There exists a number of channel pages, which basically indicate the modulation mode. Currently there are only 3 channel pages used, but there is altogether a number of 32 channel pages defined, from which 29 are reserved for future purposes. Basically there exists a number of 28 channels per channel page. Depending on the channel page properties, there are not all combinations of channels with channel pages valid. A short overview of channels and channel pages is given in Figure 2.4.

Channel page (decimal)	Channel number(s) (decimal)	Channel number description
0	0	Channel 0 is in 868 MHz band using BPSK
	1–10	Channels 1 to 10 are in 915 MHz band using BPSK
	11–26	Channels 11 to 26 are in 2.4 GHz band using O-QPSK
1	0	Channel 0 is in 868 MHz band using ASK
	1–10	Channels 1 to 10 are in 915 MHz band using ASK
	11–26	Reserved
2	0	Channel 0 is in 868 MHz band using O-QPSK
	1–10	Channels 1 to 10 are in 915 MHz band using O-QPSK
	11–26	Reserved
3–31	reserved	Reserved

Figure 2.4.: Table of Channel pages (taken from [1])

2.1.2.3. Physical frame structure

The basic data frame structure, as shown in Figure 2.5, is divided into the synchronisation header (SHR), the PHY header (PHR) and the PHY payload. The SHR contains the preamble field, which is used for symbol synchronization at the receiver and has a variable length depending on the frequency band and the modulation. Moreover the SHR contains the start frame delimiter (SFD) field which has also a variable length and indicates the end of the SHR.
The PHR contains only one field which indicates the length of the PHY protocol data unit (PPDU) frame. This field has a length of 8 bits, while one bit is reserved. A valid frame length is between a value of 9 to the maximum defined frame length. The frame length field may contains a special value of 5, which indicates an acknowledgment frame.

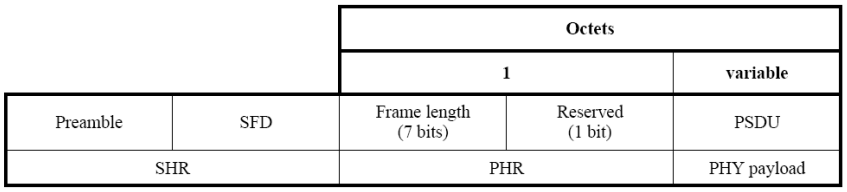

Figure 2.5.: PPDU frame format (taken from [1])

2.1.3. MAC Sublayer

The data link layer (DLL), the second layer of the OSI model is divided into two sublayers, the MAC and the logical link control (LLC). The MAC sublayer is close to the PHY layer and uses services of it. Normally the LLC sublayer sits above the MAC layer and provides services to the NWK layer, but in the IEEE 802.15.4 Standard the LLC sublayer does not exists. Normally the NWK layer of ZigBee sits directly above the MAC layer, but it is also possible to set up an IEEE 802.2 LLC layer above the MAC layer through a service specific convergence sublayer (SSCS) as described in the Annex of the IEEE 802.15.4 standard reference [5].

2.1.3.1. Introduction

The MAC sublayer of IEEE 802.15.4 provides an interface between the physical layer and the next higher layer and it can be divided into two different parts, as shown in Figure 2.6. The MAC common part sublayer (MCPS) provides the basic data services, like data requests and data indications. The second part of the MAC sublayer is the MAC sublayer management entity (MLME),

which is responsible for management services within the MAC sublayer, e.g., association and disassociation.

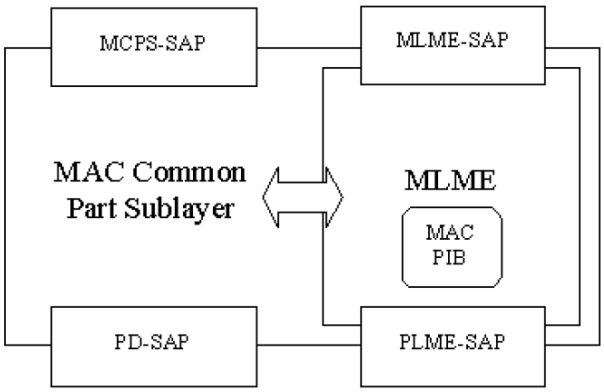

Figure 2.6.: MAC sublayer overview (taken from [1])

For a communication in a multi node network a unique address is absolutely necessary. In the IEEE 802.15.4 standard there are two addressing modes specified. Each of them has a unique 64 bit IEEE address which should be assigned to the device during the initialization. This address can be compared to the MAC address in Ethernet (IEEE 802.3). There also exists a 16 bit short address, which is not assigned in the initialization phase. A short address can be allocated during the association procedure (described in Section 2.1.3.6). If no short address is defined for a device, its internal address value should be set to `0xffff` (broadcast address). Moreover there exists a PAN address, which determines the membership of a node to a PAN (cf. the star-topology in Section 2.1.1.2). If the source PAN ID of a message is different to the destination PAN ID, the message should be routed through other nodes to the destination PAN, but this routing is out of the scope of the IEEE 802.15.4 standard.

2.1.3.2. Communication Modes

The MAC sublayer basically provides two options for message transmission. The first and simpler method is the non beacon-enabled mode. In this mode two devices communicate with each other using the medium access protocol called carrier sense multiple access with collision avoidance (CSMA-CA), which is described in Section 2.1.3.4. Since also time triggered messages should be transmitted, a method called superframe structure in a beacon-enabled mode is introduced, which is described in Section 2.1.3.3. In this mode the time interval, limited by so called beacons, is divided into two intervals. In the first

part of the superframe, CSMA-CA is used to transmit messages, whereas in the second part real time communication through scheduled TDMA is possible.

2.1.3.3. Superframe Structure

For critical messages the beacon-enabled mode is used. In this mode a coordinator manages the associated nodes and periodically transmits beacon frames. The time interval between two beacons, called beacon interval, is specified as follows:
$aBaseSuperframeDuration * 2^{BO}$ where BO is the Beacon Order. This value is valid between a Range of $0 <= BO < 15$. $BO = 15$ indicates a nonbeacon-enabled PAN. The $aBaseSuperframeDuration$ is defined as the minimum superframe duration.

The time interval between two beacons is divided (shown in Figure 2.7) into two sections, the active period and the inactive period. The length of the active period of the superframe is defined as $aBaseSuperframeDuration * 2^{SO}$ where SO is the Superframe Order. This value is valid in a range of $0 <= SO <= BO < 15$. In the inactive portion it is possible to switch the coordinator and all associated devices in sleep mode to reduce power consumption. The active portion of the superframe is also divided into two sections, the contention access period (CAP) and the contention-free period (CFP). Figure 2.7 shows the portion of the superframe.
In the CAP every node, which is associated with the coordinator is allowed to transmit packets using the CSMA-CA algorithm. If a node starts transmitting at the end of the CAP and it cannot finish the transmission before the CAP ends, it has to cancel its transmission.
The CFP is used to provide one single node of the PAN exclusive access rights on the channel. Such a time slot is called GTS. In this slot no other device is allowed to transmit a frame. If a node wants to allocate such a GTS it has to request the GTS from the coordinator in a specified time slot. The coordinator checks, if there is a free slot and updates the superframe structure. GTS can be allocated as long as the minimum CAP length is not reached.
If a node receives the beacon it has to check if the requested GTS is allocated in the GTS list. If so, it is allowed to transmit a message within this slot without using CSMA-CA. The whole active period including CAP and CFP is divided into a number of time slots (16 by default).

2.1.3.4. CSMA-CA

The CSMA-CA algorithm is basically used to transmit message frames on a shared medium, in which every node is allowed to send. There are existing

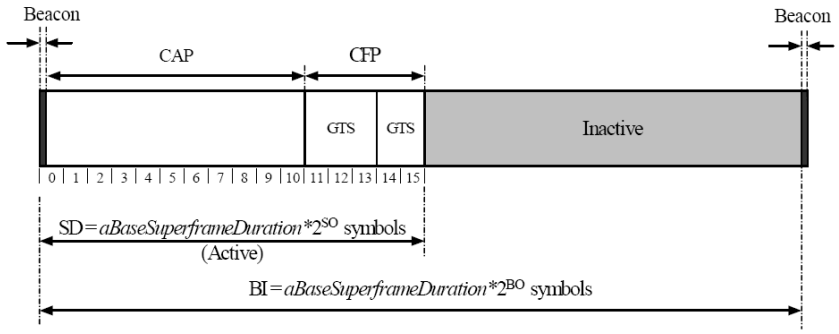

Figure 2.7.: Superframe timing diagram (taken from [1])

two versions of the CSMA-CA algorithm, as shown in Figure 2.8. One is used in non beacon-enabled PANs and the other is used in the CAP in a beacon-enabled PAN.
Before the algorithm is explained, the basic timebase of this algorithm (in general of the whole MAC sublayer) is described.

- A backoff period is the basic time period, which is defined as 20 symbols. The length of a symbol period depends on the frequency band and the modulation method and can be considered as constant. In a beacon-enabled PAN the backoff period depends on the internal clock of the coordinator, but in a non beacon-enabled PAN, the backoff period only depends on the internal clock of the MCU.

Basically there are existing three different variables which are necessary to understand the algorithm.

- The beacon exponent (BE) variable determines the maximum time which the algorithm waits before it checks the channel. This delay time is a random number between 0 and $2^{BE} - 1$ backoff periods.
- The number of backoffs (NB) variable represents the number of attempts to access the channel. If this number reaches a predefined value the algorithm terminates with a failure.
- The contention window (CW) variable determines the number of backoff periods the channel has to be cleared before transmission of the frame can be started.

CSMA-CA in a non beacon-enabled PAN In a non beacon-enabled PAN the algorithm acts as follows:
When the algorithm starts, the NB variable is set to zero and the BE variable is set to $minBE$, then a random number of backoffs between 0 and $2^{BE} - 1$ is waited. After the delay the clear channel assessment (CCA) is performed.

This means that the receiver listens to the channel for one backoff period. If the receiver detects no active transmission, the transmitter is enabled and the message can be transmitted. If the receiver detects a transmission on the channel, it increases the NB by one and also increase the BE by one. If the BE reaches the maximum backoff exponent, it does not increase the BE. After this, the algorithm checks, if the number of maximum CSMA-CA backoffs have been reached. If so, the algorithm terminates with a failure. Otherwise the algorithm returns to the begin and waits for a random period, before it performs CCA.

CSMA-CA in a beacon-enabled PAN In beacon-enabled PANs, the algorithm is quite similar, except a few differences. If the battery life extension is enabled, the BE variable is calculated as follows $BE = min(2, minBE)$ and otherwise the BE variable will be set to the same value as in the non beacon-enabled mode ($BE = minBE$). After this, the algorithm checks the boundary of the CAP. If the algorithm can't be terminated before the CAP ends, the algorithm stops and resumes its execution in the next superframe. After the backoff delay the algorithm performs CCA and if the channel is idle, the CW variable is decremented by one and if the value is not equal to zero, the algorithm performs another CCA. Since the CCA was set to 2 at the beginning of the algorithm, the CCA is performed at least twice before the message can be transmitted. If there is any traffic on the channel, the CW count will be reset to 2, the BE will be increased by one and the NB will also be increased by one. If the NB reaches the number of maximum CSMA backoffs, the algorithm terminates with a failure. If the NB variable has not reached its maximum value of two, a new CCA will be started after the backoff delay.

2.1.3.5. Starting a PAN

Before a device can associate to a PAN, it first has to know about reachable PANs and the link quality of them, which can be determined by the scan procedure. If a node is not associated with a PAN, it has the possibility to check, if anything is transmitted on the each channel. So it is possible to detect the signal energy and link quality (Energy detection scan) of each channel. Moreover it is possible to make active or passive scans.

During a passive scan the device turns on its receiver and listens to the channel for a defined period of time. After that it switches to the next specified channel and performs the scan again. If all requested channels were scanned, the MAC sublayer indicates the next higher layer and returns all discovered PANs. The active scan is quite similar to the passive scan. It additionally transmits a beacon request frame and waits a defined period of

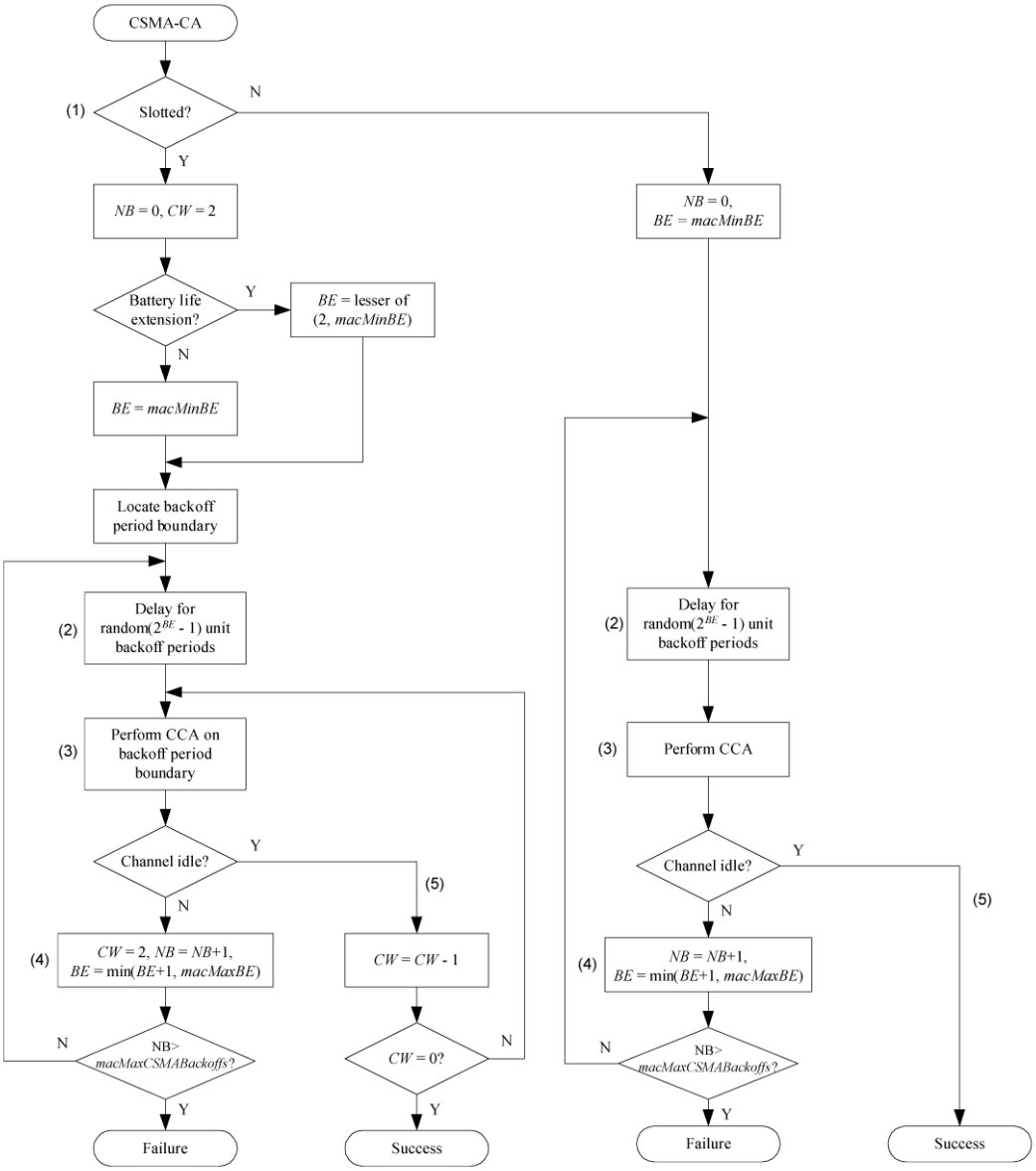

Figure 2.8.: CSMA-CA algorithm (taken from [1])

time for a beacon frame of the coordinator, hence the active scan is used for non beacon-enabled PANs. If a PAN coordinator in a non beacon-enabled PAN receives a beacon request, it has to answer with a single beacon frame.

If on the other hand a PAN coordinator wants to start a PAN, it first has to reset its internal state. After that the device updates its internal variables with the values provided by the next higher layer (e.g., PAN ID, *shortAddress*). After that the device starts operating as PAN coordinator.

2.1.3.6. Association and Disassociation

- If a device discovered a PAN coordinator through a previous scan, it can start with the association procedure. Before an association request is sent, the internal state of the device must be updated. So for example the address of the coordinator or the PAN ID must be updated. After that an association request command can be sent. If the according coordinator receives this message, it indicates the next higher layer. The higher layer decides whether the association is permitted or not. Depending on this decision the MAC sublayer sends a response to the device which requested the association. If the device receives this response, it indicates the next higher layer. If the association was successful, it stores the received short address into its internal data structure.
- If a device wants to disassociate from a PAN the next higher layer of the device sends a disassociation notification frame to the MAC. The MAC sublayer generates a disassociation notification command and transmits it to the coordinator. After receiving an acknowledgment frame from the coordinator, the MAC indicates the next higher layer the disassociation. If the device does not receive an acknowledgment frame, it indicates the disassociation anyway.

  If the coordinator wants a device to disassociate it sends a disassociation request to its MAC sublayer. The MAC sends a disassociation notification frame to the device which should be disassociated. This device should send an acknowledgment frame back to the coordinator and should indicate its next higher layer the disassociation. If the coordinator does not receive the acknowledgment frame or the timeout expires, the MAC of the coordinator indicates the next higher layer the disassociation anyway.

3. Implementation

3.1. Hardware

3.1.1. Hardware Description

The hardware basically consists of three different printed circuit boards (PCB) which are arranged and connected as shown in Figure 3.1. This figure is not a detailed nor complete schematic diagram, but it gives an overview about the hardware arrangement and shows the essential connections. The necessary pins are described in Section 3.1.2. A detailed schematic diagram can be found in Annex B.

- The controller board is a commercial development board from Texas Instruments and includes an MSP430F149 MCU. A 10 pin connector provides the JTAG interface for programming and debugging the MCU. Moreover, there is a crystal with 8MHz connected to the MCU. On this board all pins of the MCU are connected to multi-pin connectors.
- The environment board provides the power supply and basic IO functions. So there are 4 LEDs on the board and 4 Buttons on the board. There are two different voltages (5V, 3.3V) provided by the board. A level converter (MAX232) on the board provides a serial data connection (RS232) to the PC through a SUBD9 connector.
- The RF board is a commercial RF board from Microchip and consists of a Chipcon CC2420 RF chip and a few peripheral analogue parts. There is also a 12 pin connector on the board which is used for the communication with the MCU. Moreover there is a small printed RF antenna on the board.

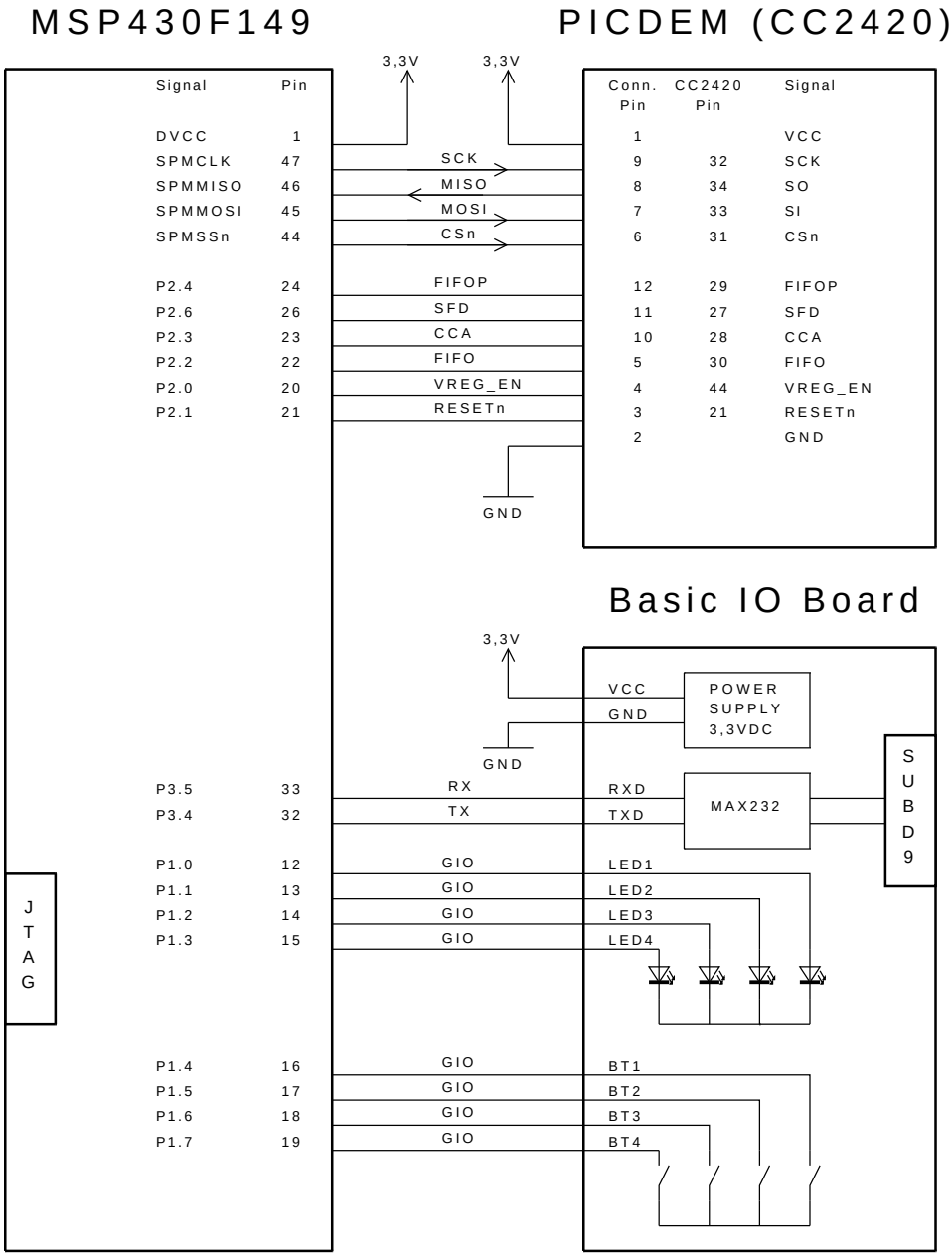

Figure 3.1.: blockdiagram of hardware arrangement

3.1.2. Interface Description

The MCU is connected to the CC2420 through a serial peripheral interface (SPI) as shown in Figure 3.1. Additionally to the SPI, the MCU and the CC2420 are connected by a few other important signal lines (Figure 3.2 and Figure 3.3).

- SFD Pin
  During the reception of a frame, this pin goes high after the SFD was completely received. It returns to low after the last byte of the message was received. If the address recognition fails, the SFD pin immediately goes low.
  In the transmit mode the SFD pin goes high, if the SFD field has been transmitted and it goes low again when the message has been completely transmitted.
- FIFO Pin
  This pin is high, if there are data bytes in the receive buffer and it gets low, when the buffer is empty again.
- FIFOP Pin
  This pin goes high if the maximum threshold of the receive buffer has been reached. Moreover it gets high if the last byte of a message is received.
- CCA Pin
  After starting the CSMA-CA algorithm the CCA pin is activated, if the receiver listened to the channel for a defined time period and nothing is detected on the channel, which means that the channel is free. The polarity of this pin can be inverted be changing a register in the CC2420.

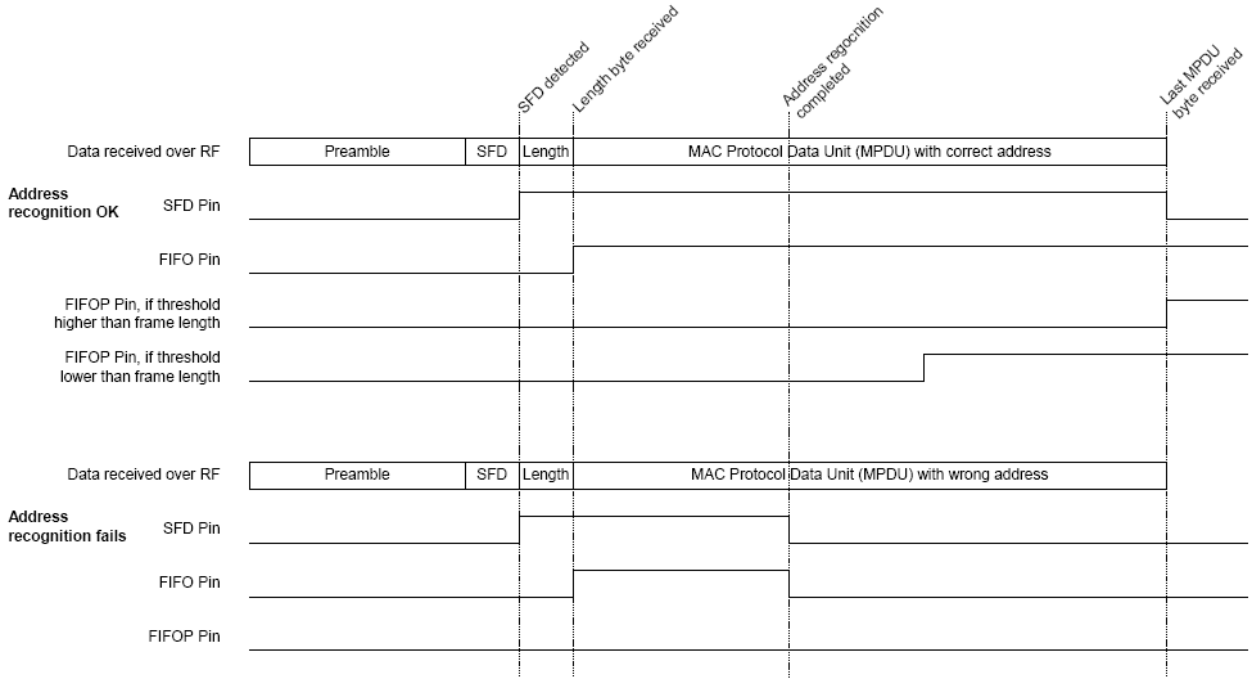

Figure 3.2.: Timing diagram during receiving (taken from [1])

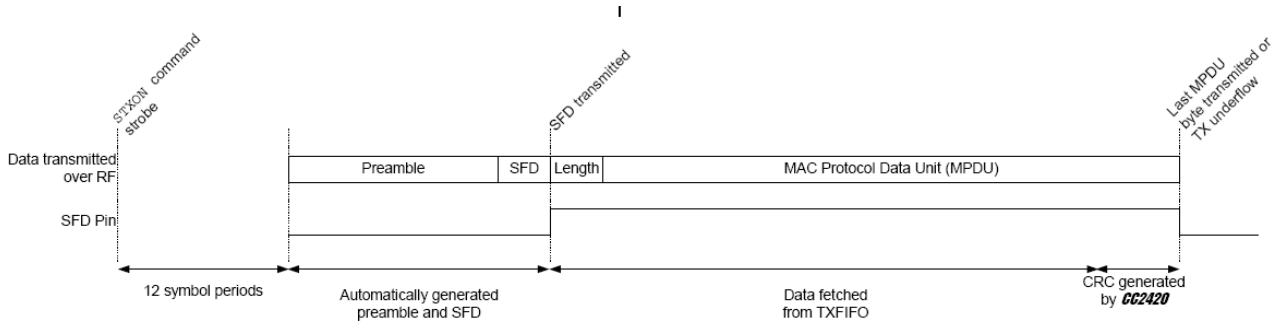

Figure 3.3.: Timing diagram during transmitting (taken from [1])

3.2. Porting of the HAL

3.2.1. Changing Macros

Since the original stack from Texas Instruments was written for the IAR compiler, some compiler specific macros and code segments were changed to make them compilable for the mspgcc:

- The macros `HAL_ENABLE_INTERRUPTS` and `HAL_DISABLE_INTERRUPTS` were changed to the mspgcc macros `_EINT()` and `_EINT()`.
- Moreover there are existing macros which disable interrupts within a critical section and make this section to an atomic operation sequence. To restore the interrupt state again after leaving this section, the interrupt state has to be stored. The IAR compiler provides macros and a predefined data structure to store the interrupt state. Since such macros are not provided by the mspgcc, the data structure and the macros had to be implemented as follows:

  ```
  typedef struct {
  uint8_t ifg1; /* Interrupt flag register 1 */
  uint8_t ifg2; /* Interrupt flag register 2 */
  uint8_t ie1; /* Interrupt enable register 1 */
  uint8_t ie2; /* Interrupt enable register 1 */
  uint16_t mysr; /* status register */
  } istate_t;
  ```

  When the critical section is entered, the interrupt flag registers, the interrupt enable registers and the global status register are stored in a special data structure and all interrupts are disabled. When the critical section is left, these registers are restored with the original values, stored in the data structure. The macros, which are used to handle these sequences are listed below:

  ```
  #define HAL_ENTER_CRITICAL_SECTION(x) \
    x.ifg1 = IFG1; x.ifg2 = IFG2; x.ie1 = IE1; \
    x.ie2 = IE2; x.mysr = READ_SR; \
    HAL_DISABLE_INTERRUPTS();
  #define HAL_EXIT_CRITICAL_SECTION(x) \
    IFG1 = x.ifg1; IFG2 = x.ifg2; \
    IE1 = x.ie1; IE2 = x.ie2; \
    WRITE_SR(x.mysr);
  ```

  Since the ISR macros are quite different between the IAR compiler and the mspgcc, the following macro was changed.

```
#define HAL_ISR_FUNCTION(y,x) \
  interrupt( x ) y (void); interrupt( x ) y (void)
```

This macro calls a function `y`, if the interrupt at the vector `x` is called.

- Moreover there exists a macro which reads the current interrupt state.

```
#define HAL_INTERRUPTS_ARE_ENABLED() ( READ_SR & GIE )
```

3.2.2. Changing Registers

Since the MCU of the original stack is different to the used MCU, a few register and pin names were changed.

- Registers of the UART in the `hal_uart.h`
- Registers of the SPI in the `hal_spi.h`
- General IO Pins of the UART and the SPI

3.2.3. Changing the clock source

The MCU which is used by the stack of Texas Instruments uses a different clock source than the actual MCU. In this project the `LFXTL` is used as clock source, with a 8MHz quartz oscillator. This clock source is also used for the SPI and the UART without a division factor.

3.3. MAC Stack Overview

The ZigBee stack from Texas Instruments is designed as shown in Figure 3.4. Apart from the ZigBee stack itself, some useful functions are provided by the HAL and the operating system abstraction layer (OSAL). The HAL library provides access to basic IO devices like buttons and LEDs. Moreover the HAL provides useful functions for UART communication. Optionally the HAL provides functions for liquid crystal display (LCD) and timer support, which are not used by the Stack itself.
The OSAL library of the stack provides basic task handling, dynamic memory management, timer functions and a messaging system for the communication between different tasks. Moreover it has a few useful functions for random number generation and memory copies.
As mentioned in a previous chapter, the stack of Texas Instruments is not completely published, so the major part of the MAC sublayer had to be reimplemented. An overview of the different parts of the provided functions is given in Figure 3.5. The most important functions, which are used in our

implementation, are basic radio functions (e.g., channel, transmit power or address). There are also functions included for the transmission and reception of a frame and additionally timer functions, which make the implementation of a superframe structure much easier.

The major disadvantage of the IEEE 802.15.4 Stack from Texas instruments is, that there is no support of GTS allocation. Hence it is not possible to realize a real time communication system with this stack. Since the original stack does not support GTS, a self defined application programming interface (API) was implemented to provide this functionality. It is quite similar to the structure of the original MAC stack and also uses many of the provided functions by Texas Instruments. A detailed description of the GTS usage is given in Section 3.4.4.7.

Even though the security capabilities are a part of the IEEE 802.15.4 standard, security was not considered in this thesis. All security parameters are ignored in the actual implementation. The security capabilities may be added in future thesis.

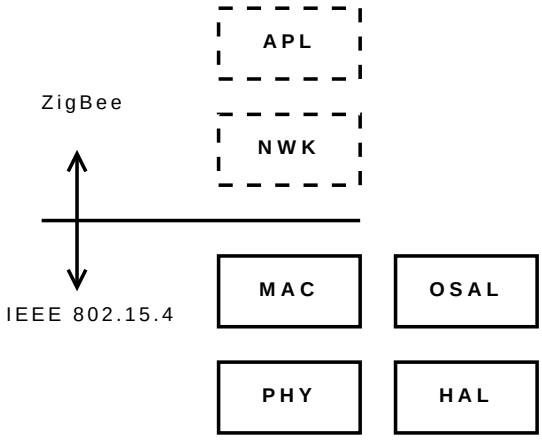

Figure 3.4.: ZigBee Stack Overview

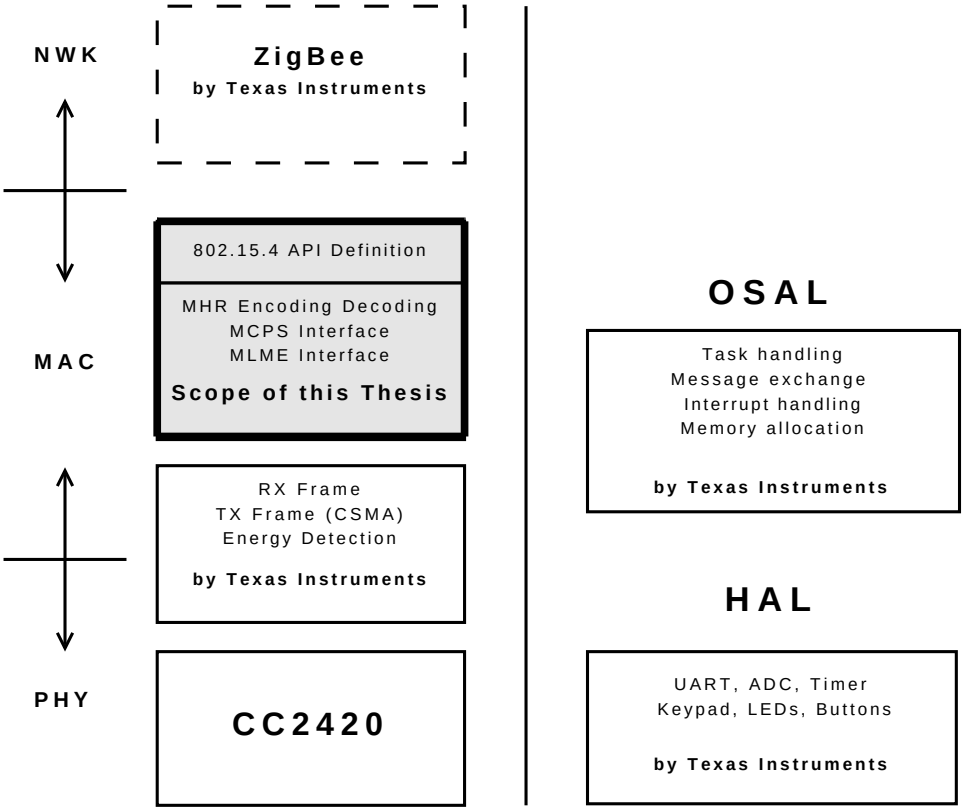

Figure 3.5.: Implementation Overview of the ZigBee Stack

3.3.1. Provided Functions of the TI Stack

In this section, the essential functions are listed, which are used in the current implementation of the API.

3.3.1.1. Basic Radio Functions

The basic radio functions, which are implemented in the file `mac_radio.c` include functions to set the following parameters of the RF device.

- PAN coordinator address
- PAN ID
- Device short address
- Device extended IEEE address
- Transmit power
- Physical channel

This is necessary because the RF chip is able to perform message filtering and address decoding. There are also functions for channel scanning implemented. These functions only edit parameters in the RX state machine, which is described in Section 3.3.1.2, so that only beacon frames are received. Moreover there is a function for the energy detection implemented which calculates the received signal strength indication (RSSI) value and the link quality (LQ).

3.3.1.2. Receive and Transmit Functions

The receive primitive is implemented as an event driven state machine at the interrupt level. If a frame is received, a callback function is executed. The receive state machine decodes the frame header and delivers the header information to the callback function through parameters. The callback function gets a pointer to the buffer of the payload and the length of the frame. This buffer is allocated by using OSAL functions for dynamic memory allocation. It has to be deallocated immediately after its use, otherwise the memory gets an overflow and the system crashes.

The `macTxFrame()` function is designed to transmit all three types of messages, the unslotted CSMA-CA, the slotted CSMA-CA and the slotted (immediate) transmission. A closer loop inside the function revealed that only the unslotted CSMA-CA transmission is originally supported by Texas Instruments.
The reason why the slotted CSMA-CA mode is not originally supported is, that the CC2420 chip is not completely compliant to the 802.15.4 standard and so it is not possible to implement a compliant version of the slotted CSMA-CA mode.
If another mode is chosen, the function forces an assert (a complete stop of the MCU). Therefore this function was extended to support also the slotted CSMA-CA and the slotted immediate transmission.
As mentioned in Section 2.1.3.4 the algorithm has to perform the CCA check twice, but as described in [7], the CC2420 has only one internal operation, called `STXONCCA`, which performs the CCA only once. If the channel is free, the frame is immediately transmitted. This fact is ignored by the modified implementation, but the remaining parts of the TX function comply to the IEEE 802.15.4 standard.

3.3.1.3. Timer Functions

Since there are only two timers in the MSP430 MCU and a lot of time triggered events and much delays have to be handled, the timer functions

are very complex. The whole MAC stack needs only one hardware timer, as shown in Figure 3.6 to handle all timing tasks. Therefore a timer is started at the initialization phase of the device and a rollover interrupt is registered for a period of one backoff interval, which is the primary timebase for the superframe structure and CSMA-CA algorithm.

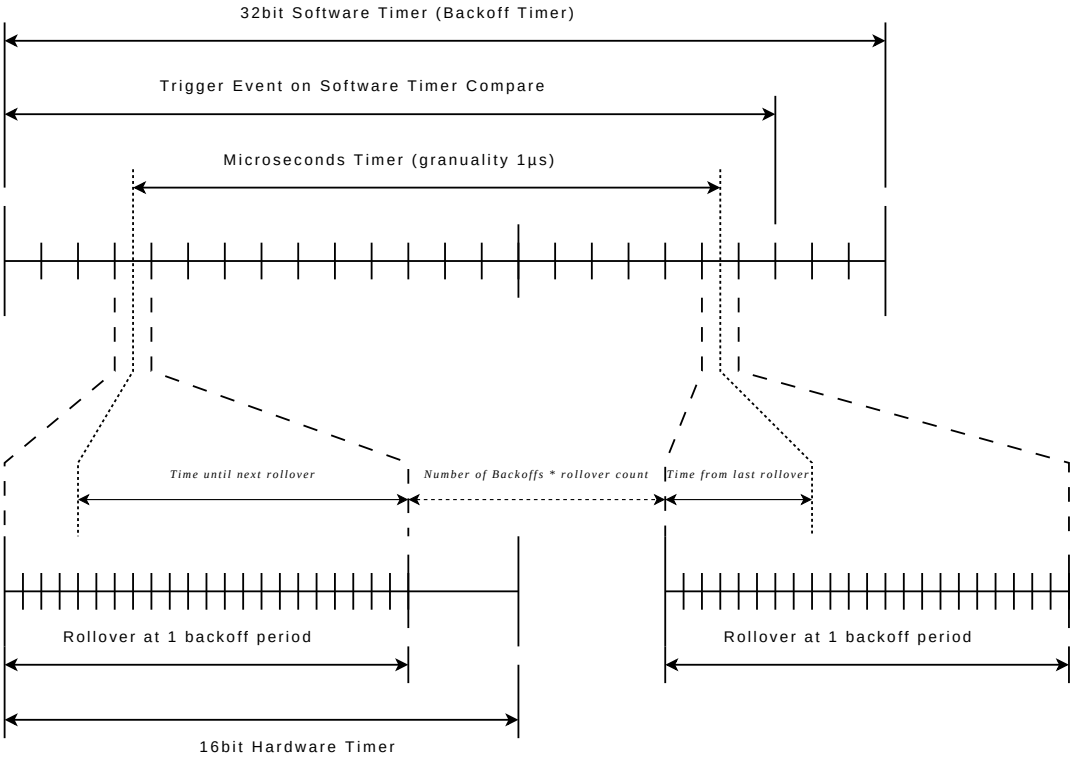

Figure 3.6.: Schematic view of the global timebase

Microseconds Timer A very important function (`macMcuTimerUsecs()`) provides a user defined callback after a given period of microseconds. This function is used to set a timeout, if a device is waiting for an acknowledgment.

Backoff Timer The backoff timer is used for the superframe structure and has a granularity of one backoff period. The backoff timer is a 32 bit variable, which is incremented during each overflow of the hardware timer rollover. In beacon enabled PANs the count should always be reseted, if a beacon frame is received. To adjust the local backoff timer with the timebase of the coordinator, there exists a function, which realigns the local timer. This realignment uses the current value of the timer at reception time and the estimated timer

count of the coordinator to calculate the correction value for the timer rollover. It is also possible to register a rollover callback function which is called periodically after a number of backoff periods. This rollover callback can be used to transmit the beacon frame, if the device is a coordinator.
Another function (`macBackoffTimerSetTrigger()`) allows a device to register a trigger callback, which can be used to send a frame in a specified GTS. After the execution of a trigger callback, the registration is cleared and the trigger has to be registered again, if necessary.

3.3.2. Source File Structure

An overview of the file structure of the stack is given in Figure 3.7. The current implementation of the IEEE 802.15.4 stack is implemented in the API source files, which are described in the following sections. The file `mac_spec.h` provides basic constants of the IEEE 802.15.4 standard, while the `mac_api.h` provides the interfaces of the MAC primitives and the used data structures and defines some useful constants.

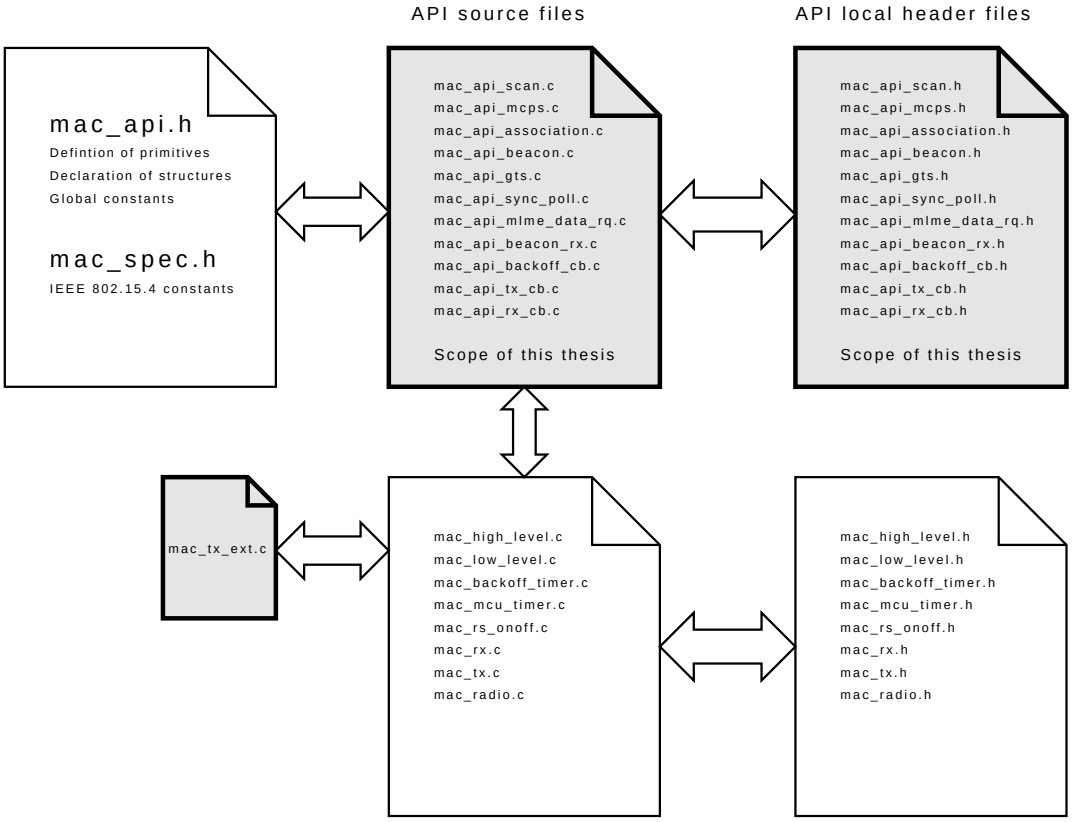

Figure 3.7.: Overview of the file structure

3.4. MAC API Description

As mentioned in the introduction of this thesis, the API is only provided by a static library, which is not compliant to the mspgcc. For compatibility reasons, the API, which was implemented has the same structure and interfaces as the original MAC Stack of Texas Instruments. The function headers and data structures are provided in the file `mac_api.h` and completely described in [6]. The compliance of the syntax between the own stack and the one of Texas Instruments is very important, because in the future it should be possible to set up the NWK and APL layer of the ZigBee stack of Texas Instruments.
Currently only a set of the most important functions is implemented. Moreover there is a self defined API extension implemented which provides GTS handling, because the original Stack of Texas Instruments does not support GTS as mentioned in Section 3.3.
Since the used MCU has only 2KB memory and the one which is originally used by the implementation of Texas Instruments, has 8Kbytes memory, it was not possible to provide the full functionality as defined in the IEEE 802.15.4 standard. There are limitations in the buffer size of the transmit buffer as described in Section 3.4.2. Moreover the dynamically allocated receive buffer is very limited.
These limitations cause some restrictions of the API primitives which are described in Section 3.4.3.1 and the following.

3.4.1. General Event Structure

To make the MAC API very flexible and efficient, it is necessary to implement the API functions event triggered. Such an implementation has the advantage, that the system does not waste expensive CPU capacity and is also easier to understand.

Since some of the API functions are event triggered, an efficient and global method is implemented to handle them. So there exists one single callback function for the whole MAC API, which is called every time an event occurs. To identify the actual event and to deliver the individual parameters, a union data structure (Figure 3.8) exists which includes one individual data structure for each event. The first two values of the union are the identifier of the event and the status variable. There exists also a data structure in the union, which allows direct access to the event and status variable, without using a special event data structure. This is possible because all members of the union are using the same memory area and the status and the event variable are defined

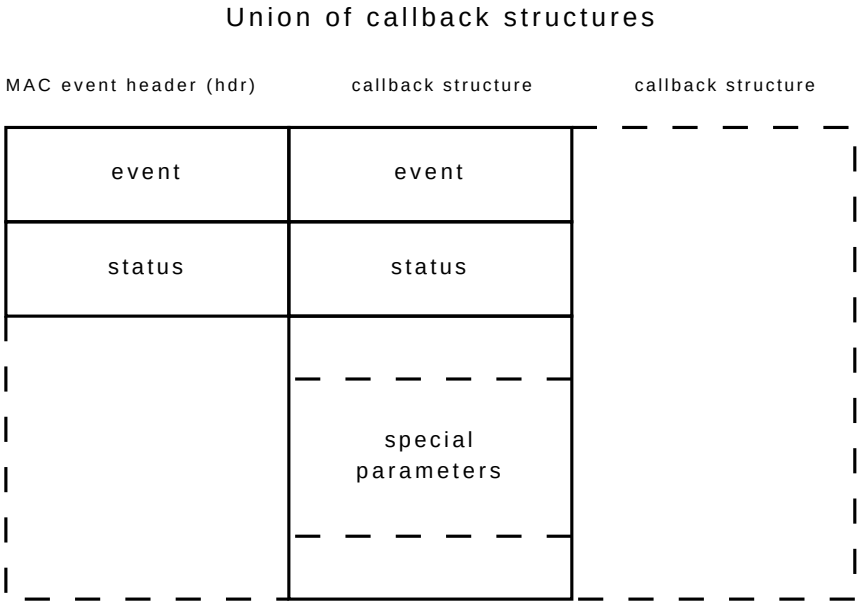

Figure 3.8.: Overview of the union data structure

always at the beginning of each individual data structure.

3.4.2. Memory Management

Since the MCU has only 2KB memory, it is not possible to fully implement the functionality, which is described in the standard. Alternatively two global frame buffers are used, as shown in Figure 3.9, which are allocated statically at the beginning.

- The first buffer is used for beacon frames and is divided into three parts. In the first part of the frame the MAC header (MHR) fields are located. Since these fields have a variable length and the following GTS fields should be updated independently, the GTS management uses an own buffer for GTS fields. Before a beacon is transmitted, the GTS buffer is copied to the main buffer, depending on the length of the MHR fields. The pending addresses field are handled quite similar, they are also updated independently and copied to the main frame before a beacon transmission.
- The second buffer is used for all other frame types including MCPS frames and MLME frames. This buffer has no special structure like the beacon buffer.

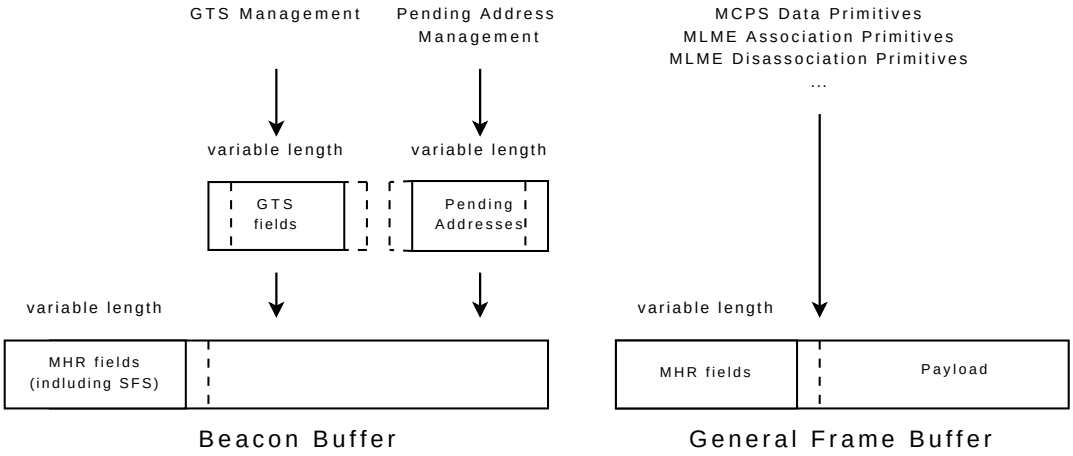

Figure 3.9.: Internal memory management of the stack

3.4.3. MAC Data Service

The Data Interfaces provides primitives (defined in [6]) which are used to exchange data between the MAC and the next higher layer. The MAC itself is able to transmit data frames to the receiver, where the data can be read by the next higher layer of the MAC.
An overview of the primitives of the MCPS is given the following table:

Name
MCPS Data Request
MCPS Purge Request

Table 3.1.: Overview of MCPS primitives

3.4.3.1. MCPS Data Request

Description The `MAC_McpsDataReq()` is used to transmit a data frame as shown in Figure 3.10 to the receiver which is specified by the parameters `dstAddr` and `dstPanId`. The function forms the frame header, considering the input parameters as described below. After the function has formed the frame header it adds the requested data and sends the frame using the function `macTxFrame()` which is originally provided by the Texas Instruments stack.

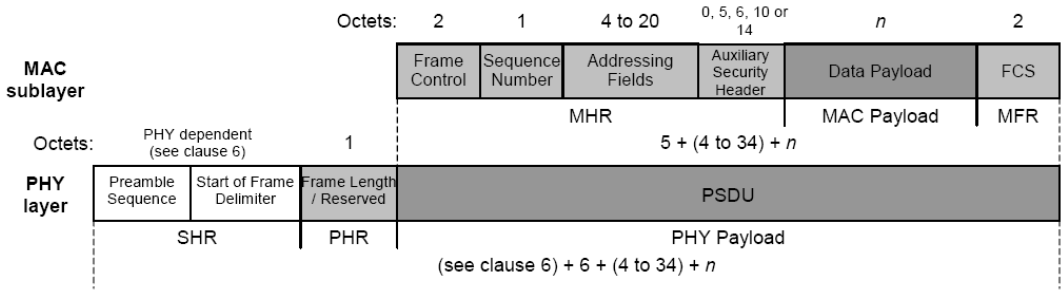

Figure 3.10.: Schematic view of a data frame (taken from [1])

Events If the MAC_McpsDataReq() primitive is called the following events possibly occur:

- `MAC_MCPS_DATA_CNF`
  The `MAC_MCPS_DATA_CNF` event always occurs at the device which has called the `MAC_McpsDataReq()` primitive.
  - If the next higher layer tries to call two data requests consecutively, the `MAC_MCPS_DATA_CNF` event is generated with the status `MAC_TRANSACTION_OVERVFLOW`, because it is only possible to buffer one single frame.
  - If an acknowledgment frame was requested and a number of *macMaxFrameRetries* transmissions of the data request frame fail, the status `MAC_NO_ACK` is delivered.
  - If the message was transmitted successfully, the status variable of the event has the value `MAC_SUCCESS`.
- `MAC_MCPS_DATA_IND`
  The `MAC_MCPS_DATA_IND` is always generated at the destination device of the actual data request.
  - If the frame was successfully transmitted and the destination address of the message is equal to the address of the receiver, a buffer with the length of the frame payload is allocated at the receiving device. Finally the `MAC_MCPS_DATA_IND` event is generated with the status `MAC_SUCCESS` and the payload can be read from the buffer.

Some special cases of these events which occur during indirect transmission and the GTS transmission are described below.

Options Since the MAC_McpsDataReq() can be used in different network modes (beacon-enabled, non beacon-enabled), there exist different transmission modes which are described as follows:

- Normal transmission
  If no special parameter is set, the frame is immediately transmitted using the `macTxFrame(type)` function with the parameter `MAC_TX_TYPE_SLOTTED_CSMA` or `MAC_TX_TYPE_UNSLOTTED_CSMA`, depending on if the device is associated with a beacon-enabled PAN or not.

- GTS transmission
  If the parameter `txOptions` contains the flag `MAC_TXOPTION_GTS`, the function checks, if the device is associated with a beacon-enabled PAN. If a GTS is allocated, a timer will be triggered at the beginning of the GTS. If the trigger is fired, the frame is transmitted using the function `macTxFrame(type)` with the parameter `MAC_TX_TYPE_SLOTTED`.

- Indirect transmission
  If the flag `MAC_TXOPTION_INDIRECT` is set in the parameter field `txOptions` and this function is called by a PAN Coordinator, the frame is transmitted using indirect transmission.
  In this case the address of the destination device is added to the pending addresses field of the beacon frame. Each associated device should check if there is a message pending by examining the pending addresses field of the beacon frame. If the own address is recognized, a MLME data request should be sent to the coordinator which is described in the next sections. If the coordinator receives the expected MLME data request, it should return an acknowledgment frame with `FRAME_PENDING` flag in the frame control field (FCF) set. To determine whether the acknowledgment frame should be set or not, a callback function is called by the provided `macTxFrame()` function. After the acknowledgment frame was transmitted, the actual data frame is transmitted using the `macTxFrame()` function with the parameter `MAC_TX_TYPE_SLOTTED_CSMA` during the CAP period.
  After adding the destination address to the pending addresses field, a counter is started which is increased by each beacon frame transmission. If the coordinator does not receive a MLME data request within *aGTSDescPersistenceTime* beacon transmissions, the address is removed from the pending addresses field and the `MAC_MCPS_DATA_CNF` event occurs with the status `MAC_TRANSACTION_OVERFLOW`.
  If the MLME data request is not acknowledged, it is retransmitted up to a number of *macMaxFrameRetries*. After the last retransmission fails, the `MAC_MCPS_DATA_CNF` event is generated with the status `MAC_CHANNEL_ACCESS_FAILURE`.

Restrictions

- If an indirect transmission in a non beacon-enabled PAN is requested, the message frame normally should be buffered, until the associated device requests the message using the `MAC_Mlme_PollReq()` primitive.
  Since the MCU has limited memory, a buffered frame would block the whole communication of the coordinator, until the data is requested by the other device. In the current implementation the `MAC_TXOPTION_INDIRECT` parameter in a non beacon-enabled PAN is ignored. In the future, this indirect transmission should be included.
- Unlike the API specification [6], this function can not buffer more than one frame, because of the high demand of memory. A new transmission can only be started, if the past transmission was finished, otherwise the `MAC_MCPS_DATA_CNF` event occurs with the status `MAC_TRANSACTION_OVERFLOW`.
- Since the memory of the MCU is limited, it is only possible to receive two to three frames (depending on the length) consecutively. Therefore it is recommended to deallocate the dynamic buffer immediately after the use of the data.

3.4.3.2. MCPS Purge Request

Description The `MAC_McpsPurgeReq()` primitive is used to purge requested data frames from the message queue.

Restrictions Because of limited memory, there exists a buffer for only one data or command frame which is defined statically. It is only possible to start a MCPS data request (or any MLME request), if the past transmission was completed. Therefore the API function `MAC_McpsPurgeReq()` is obsolete and not implemented. In a future version this function should be implemented.

3.4.3.3. MCPS Data Allocate

Description There exists also a function `MAC_McpsDataAlloc()` in the API specification [6] which is not specified in the IEEE standard. This function dynamically allocates a frame buffer of a specified length.

Events The `MAC_McpsDataAlloc()` primitive does not generate any events.

Restrictions Because of the memory limitations, this function returns only the pointer to the single buffer in each case.

3.4.4. Management Services

The Management services are used for association, disassociation, synchronisation and much more. A detailed description is given in [1]. An overview of the primitives which are currently implemented, is given in the following table:

Name	Implemented
MLME Association	yes
MLME Disassociation	yes
MLME Beacon Notify	yes
MLME Get	yes
MLME Set	yes
MLME GTS	yes
MLME Orphan	not yet
MLME Reset	yes
MLME RX Enable	yes
MLME Scan	yes
MLME Start	yes
MLME Comm-Status	yes
MLME Sync	implicit included
MLME Sync-Loss	not yet
MLME Poll	yes

Table 3.2.: Summary of primitives

The primitives which are currently implemented are described in the following sections.

3.4.4.1. MLME Association

Description The association procedure is used to associate an unassociated device to a PAN. To start such an association procedure, the channel and the address of the PAN have to be known by the device. This information could be either statically written to the device memory or discovered through a previous scan procedure, as described in Section 3.4.4.10.

If a device wants to associate with a PAN, the primitive `MAC_Mlme_AssociateReq()` has to be called, which forms the association request frame and checks, if the local PAN is a beacon-enabled PAN or not. This distinction can be made by looking at the MAC PAN information base (PIB) elements *BeaconOrder* and *superframeOrder*. If one of the two values is greater or equal than 15, it is a non beacon-enabled PAN. If both values are smaller than 15, a beacon-enabled PAN is used. The different association procedures are described below.

Events If the `MAC_MlmeAssociationReq()` primitive is called the following events can occur:

- `MAC_MLME_ASSOCIATE_CNF`
  - If there is an active transmission detected at the beginning of the `MAC_Mlme_AssociateReq()` primitive, the association procedure can not be started. The `MAC_MLME_ASSOCIATE_CNF` event is generated with the status `MAC_CHANNEL_ACCESS_FAILURE`.
  - If a MLME data request frame was sent by the associating device and an acknowledgment frame was received from the coordinator with the frame pending subfield in the FCF not set, the `MAC_MLME_ASSOCIATE_CNF` event is generated with status `MAC_NO_DATA`.
  - If the associating device receives an association response frame from the coordinator, it generates a `MAC_MLME_ASSOCIATION_CNF` event and delivers the received address information and actual status to the callback function. Note that the parameters of the MAC PIB are not updated by the API function, because this is not within the scope of the standard and has to be done by the next higher layer.

- `MAC_MLME_ASSOCIATE_IND`
  - If the MLME association request frame is received by the coordinator and the address is valid, the `MAC_MLME_ASSOCIATION_IND` is generated and indicates the association request. After the indication the coordinator has time to decide about the association and should call the `MAC_MlmeAssociationRsp()` primitive afterwards.
- `MAC_MLME_BEACON_NOTIFY_IND`
  The occurance of this event is described in Section 3.4.4.4.

Association in a beacon-enabled PAN If a device wants to associate to a beacon-enabled PAN, it transmits the previously formed frame during the CAP

using the `macTxFrame()` function. If an acknowledgment frame is received, a timer is started which requests a callback after a time of *aResponseWaitTime*. During this time interval the coordinator has time to decide about the association, while the associating device checks each received beacon for a pending message.
After the coordinator made its decision, the `MAC_MlmeAssociationRsp()` is called and the coordinator adds the extended 64 bit address of the device to the pending addresses field of the beacon frame and waits, while the response frame is requested through a MLME data request command by the associating device. If a beacon with pending information (device address of the associating device in the address pending fields) is received by the associating device, the timer is canceled and a MLME data request frame is automatically sent to the coordinator, if the *macAutoRequest* parameter of the MAC PIB is set. However if this parameter is not set, the event `MAC_MLME_BEACON_NOTIFY_IND` is generated as described in Section 3.4.4.4 and the next higher layer decides whether a MLME data frame is sent or not.
If the coordinator receives this data request, an acknowledgment frame is generated in which the frame pending subfield in the FCF is set. After the acknowledgment frame was transmitted, the actual response frame is transmitted to the device using the extended 64bit address as destination address.
If the associating device receives the MLME association response frame the association procedure is finished.

Association in a non beacon-enabled PAN In a non beacon-enabled PAN, the frame forming and the transmission of the association request command is quite similar to the beacon-enabled mode. However if the request frame was successfully sent, a timer is started which generates an interrupt after a time of *aResponseWaitTime*. During this time the coordinator has time to decide whether it allows association or not. Afterwards it should call the `MAC_MlmeAssociationRsp()` primitive which waits for the MLME data request frame from the device.
After the expiration of the timer at the associating device, an interrupt service routine is called and a MLME data request frame is sent to the coordinator. If the coordinator sends an acknowledgment frame, with the frame pending subfield set in the FCF, the associating device waits for an association response of the coordinator.
If the coordinator receives this MLME data request, an acknowledgment frame is generated, with the frame pending subfield set. After the acknowledgment frame was transmitted, the actual response frame is transmitted to the device using the extended 64bit address as destination address.
If the device receives the MLME association response frame the association

procedure is finished.

3.4.4.2. MLME Disassociation

The disassociation is used to disassociate an associated device from a PAN. The disassociation procedure can be started either by the coordinator or by the associated device. When the disassociation is started with the primitive `MAC_MlmeDisassociateReq()`, the function first checks, if there is no active transmission and then the disassociation frame is formed and transmitted as described below.

Events If the MAC_MlmeDisassociationReq() primitive is called the following events can occur.

- `MAC_MLME_DISASSOCIATE_CNF`
  - If there is an active transmission which uses the frame buffer at the same time the MAC_MlmeDisassociationReq() is called, the disassociation is stopped and the `MAC_MLME_DISASSOCIATE_CNF` event is generated with the status `MAC_TRANSACTION_OVERFLOW`.
  - If no acknowledgment frame is received after *aMaxFrameRetries* retries, the `MAC_MLME_DISASSOCIATE_CNF` event is generated with the status `MAC_NO_ACK`.
  - If the device which initiated the disassociation receives an acknowledgment frame, the `MAC_MLME_DISASSOCIATE_CNF` event is generated with the status `MAC_SUCCESS`.
- `MAC_MLME_DISASSOCIATE_IND`
  - If a device receives a disassociation notification frame, it extracts the disassociation reason from the frame and immediately informs the next higher layer by generating the `MAC_MLME_DISASSOCIATE_IND` event. The MAC PIB is not updated by the API function, because this is not within the scope of the IEEE 802.15.4 standard.
- `MAC_MLME_BEACON_NOTIFY_IND`
  The occurance of this event is described in Section 3.4.4.4.

Note that the `MAC_MLME_DISASSOCIATE_CNF` event is generated at the device which initiated the disassociation procedure. This could be either the coordinator or the disassociating device.
The `MAC_MLME_BEACON_NOTIFY_IND` event is always generated at the receiver of the disassociation notification frame.

Disassociation initiated by the device If the disassociation was initiated by the currently associated device, the disassociation notification frame is transmitted in the CAP using slotted CSMA-CA in a beacon-enabled PAN. In a non beacon-enabled PAN the frame is transmitted using unslotted CSMA-CA. Since the acknowledge request parameter in the FCF is set, the device waits for an acknowledgment frame. If the acknowledgment frame is not received within a time of *macAckWaitDuration* symbols the frame is retransmitted up to a number of *aMaxFrameRetries*. If the acknowledgment frame is received, the device is disassociated.

Disassociation initiated by the coordinator If the disassociation was initiated by the coordinator in a beacon-enabled PAN, the disassociation request frame should be transmitted indirectly. After forming the frame, the address of the device which should be disassociated is added to the pending addresses field of the beacon frame. If the currently associated device recognizes that a message is pending, it either automatically transmits a MLME data request or generates a `MAC_MLME_BEACON_NOTIFY_IND` event as described in Section 3.4.4.4. After the coordinator receives the MLME data request frame, it transmits an acknowledgment frame, with the frame pending subfield set in the FCF. Then it transmits the previously formed disassociation notification frame to the device. Since the acknowledge request subfield in the FCF is set, the coordinator retransmits the frame up to a number of *aMaxFrameRetries*. If a device receives a disassociation notification frame, it is disassociated and the `MAC_MLME_DISASSOCIATE_IND` event is generated as described above.

Restrictions Since the disassociation (initiated by the coordinator) requires indirect transmission, it is currently not possible that a coordinator initiates the disassociation in a non beacon-enabled PAN. The reason therefore is that in case of a disassociation request, the whole traffic of the coordinator would be blocked until the associated device calls a `MAC_MlmePollReq()` primitive.

General Assumption In case the disassociation is not successful, a device (either the coordinator or the associated device) can assume that the associated device is disassociated now.

3.4.4.3. MLME Start

Description The `MAC_MlmeStartReq()` is used to initiate the device as PAN coordinator. So if this primitive is called, first the parameters (from the next higher layer) like PAN ID or short address are checked and the MAC PIB

is updated if the values are valid. If the *superframeOrder* and *BeaconOrder* parameters are less than 15, a beacon-enabled PAN should be started and the rollover value of the backoff timer is initiated to generate a callback event every $2^{BeaconOrder}$ backoff periods. Each time this event occurs, the rollover value is updated with the *BeaconOrder* value of the MAC PIB and a beacon frame is formed and sent.

Events

- `MAC_MLME_START_CNF`
  - The `MAC_MLME_START_CNF` event is called with status `MAC_INVALID_PARAMETER`, if one of the parameters (e.g., *superframeOrder* or *BeaconOrder*) is not valid.
  - If the `MAC_MlmeStartReq()` primitive was performed successfully the `MAC_MLME_START_CNF` event is generated with the status `MAC_SUCCESS`.

Beacon Frame The frame, as shown in Figure 3.11 consists of four different parts, where each part has a variable length. Since that, it is necessary to edit each part independently. The best method is to implement different buffers which are merged before transmitting a frame as shown in Figure 3.9. These different parts are listed as follows:

- The first part consists of the MHR field which is similar in all frames and the superframe specification (SFS) which has a constant length of 2 byte. Since the beacon frame is not a directed message, the destination address should be the broadcast address `0xFFFF`. The acknowledge request subfield should not be set, because it is not possible to receive an acknowledgment frame from each receiving device.
- The second part of the beacon frame is the GTS field which contains a variable number of entries. This part contains the direction field and one three byte record for each GTS entry. Each entry consists of a two byte short address and one byte specification field, which determines the length and the start slot of the actual GTS.

- The third part of the beacon frame represents the pending addresses field which contains a variable number of addresses. The first byte of this part is the pending address specification, which contains the count of short and extended addresses stored in the pending addresses field. In the following bytes the addresses are stored.

- The last part of the beacon frame is optional and consists of the beacon payload.

If a beacon should be transmitted, the buffers which can be edited by the corresponding function (e.g., GTS management, pending addresses management) are merged to one single buffer and the frame is finally transmitted using direct transmission.

Note Since the beacon frame has to be sent periodically, it is the only frame type which has its own buffer as shown in Figure 3.9.

Restrictions

- The length of the GTS list is limited to 11 bytes. It is only possible to store at most three GTS.
- The length of the pending addresses field is limited to 17 bytes in the current implementation.t is only possible to store at most two extended addresses, eight short addresses or any other combination with a size of maximum 16 bytes.
- In this implementation a beacon payload is not needed and would only waste memory. No buffer for the beacon payload is allocated.

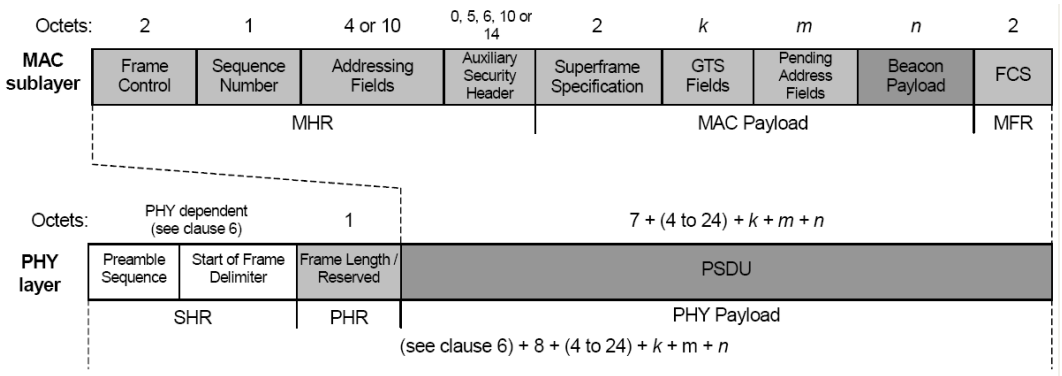

Figure 3.11.: Schematic view of a beacon frame (taken from [1])

3.4.4.4. MLME Beacon Notify

Description The MLME Beacon Notify (`MAC_MLME_BEACON_NOTIFY_IND`) is an event which is possibly generated at an associated device in a beacon-enabled PAN.
The beacon notify indication is used to inform the next higher layer about

the reception of a beacon frame. This event is only generated, if the beacon contains any payload or if the *macAutoRequest* parameter in the MAC PIB is not set.
To check if the beacon contains payload, the whole beacon frame (shown in Figure 3.11) has to be decoded.

- The MHR fields are decoded by the provided RX state machine and available in a structured form. The SFS is the first entry of the beacon frame with a length of two bytes.
- Then the GTS specification has to be decoded to determine the length of the GTS list. If the GTS list contains at least one entry, there also exists a GTS direction field of one byte and 3 bytes for each GTS slot in the list.
- To determine the length of the pending addresses field, the pending address specification has to be decoded. The pending addresses field possibly contains short addresses of 2 bytes and extended addresses of 8 bytes.

Events

- If there are any data in the frame following the pending addresses field, these bytes are the beacon payload and they are extracted and copied to a dynamic buffer. Afterwards the `MAC_MLME_BEACON_NOTIFY_IND` event is generated. Since this function allocates dynamic memory, it is strongly recommended to deallocate the memory immediately after the usage and before the reception of the next beacon frame with any payload.

- The `MAC_MLME_BEACON_NOTIFY_IND` is also generated, if the *macAutoRequest* parameter of the MAC PIB is not set and if the own device address (either short or extended) is found in the pending addresses field of the beacon frame. In this case the coordinator wants to transmit a frame to the associated device and waits for a MLME data request frame. If the parameter *macAutoRequest* is set, the MLME data request is transmitted automatically and otherwise the next higher layer is responsible for that.

Restrictions As a result of the limited memory, the maximum number of address, especially extended addresses is strongly limited to two addresses (described in Section 3.4.4.3) and not as large as defined in the IEEE standard.

3.4.4.5. MLME Get

Description The `MAC_MlmeGetReq()` primitive is used to read parameters of the MAC PIB. In the MAC PIB there are some parameters, which are *read only* and some, which are *read and writable*. To choose the desired parameter, a numerical value is used, which is defined by macros in the `mac_api.h` file.

Return Values Apart from the other MAC primitives, the `MAC_MlmeGetReq()` does not generate the general callback event as described in Section 3.4.1, but it only returns the status value via a simple return value.
If an invalid parameter should be read, the function returns the value `MAC_UNSUPPORTED_ATTRIBUTE`.

Note The requested value is delivered through a so called *void pointer*. As a result of that, the calling function has to know exactly the data type of the desired variable and no error or warning is generated by the compiler in case of a bad type conversion.

3.4.4.6. MLME Set

Description The `MAC_MlmeSetReq()` primitive is quite similar to `MAC_MlmeGetReq()` primitive, but it is used to set parameters in the MAC PIB.

Return Values Apart from the other MAC primitives, the `MAC_MlmeSetReq()` does not generate the general callback event as described in Section 3.4.1. It only returns the status value via a simple return value.

- If a *read only* parameter should be set, the function returns the status `MAC_READ_ONLY`.
- If an unknown parameter should be set, the status `MAC_UNSUPPORTED_ATTRIBUTE` is returned.
- If the requested parameter is valid and if it is a writable parameter, the value domain is checked. If there are restrictions in domain for the actual parameter (e.g., $0 \leq macPib.BeaconOrder \leq 15$) and the value is out of range, it is not updated and the status `MAC_INVALID_PARAMTER` is returned.
- In case of a correct request (attribute name + value) the primitive returns the status `MAC_SUCCESS`.

Note If the requested parameter is also stored in the CC2420 (e.g., *macPib.shortAdress*), the value in the chip has to be updated too, using functions, which are provided by the original stack.

3.4.4.7. MLME GTS

Description The `MAC_MlmeGtsReq()` is called by an associated device in a beacon-enabled PAN and is used to allocate or deallocate a GTS. This primitive checks the parameter value *GTSCharacteristics* (Figure 3.12). If it is valid, a GTS request frame is formed and transmitted to the coordinator during the CAP period. If no acknowledgment frame is received the frame is retransmitted. If the coordinator receives the GTS request frame, it analyzes the GTS characteristics field (shown in Figure 3.12). If the *Characteristics Type* subfield is set, a GTS is requested Finally it is checked, if there is a free slot in the CFP.

bits: 0–3	4	5	6–7
GTS Length	GTS Direction	Characteristics Type	Reserved

Figure 3.12.: GTS Characteristics field (taken from [1])

Adding a GTS Since no fragmentation of GTS is allowed, the new slot can only be allocated only at the beginning of the CFP interval. So it only has to be considered the minimum CAP length, which guarantees a minimum time interval for CSMA-CA transmissions. This value is not defined by a number of slots in the superframe structure, because the length of a slot depends on the *SuperframeOrder* and *BeaconOrder*, but it is defined by the number of symbols, while a symbol interval is constant and defined in the PHY of the IEEE standard. As a result of that, the higher the *SuperframeOrder* is, the more GTS can be allocated.
If there is enough time to allocate a GTS, the address of the requesting device is added to the GTS list of the beacon coordinator.

Removing a GTS Since the GTS slot is only valid for a number of *aMaxDescPersistenceTime* beacon frames, a counter variable has to be introduced for each entry in the GTS list. These variables have to be decreased each time, a beacon is transmitted. If one value reaches zero, the according GTS is deallocated. A GTS is also deallocated if the device wants to deallocate the GTS by calling the `MAC_MlmeGtsReq()`.

The deallocation of a GTS is quite harder, because the whole GTS frame buffer and the CFP itself (shown in Figure 3.13) have to be reorganized.

- First the actual slot has to be found either by searching for the address, in case of the deallocation was requested by a device or by searching for expired persistence variables, every time a beacon is transmitted.
- If a GTS is selected, it first has to be deleted from the GTS list of the beacon frame buffer and the entries above the actual entry (in the memory) have to be moved. Additionally the persistence counter variables have to be moved analogue to the GTS entry.
- Afterwards the start slot value of each slot, which is chronological before the actual frame, is increased by the size (in slots) of the deallocated slot to close the gap in the CFP as shown in step 2 of Figure 3.13.

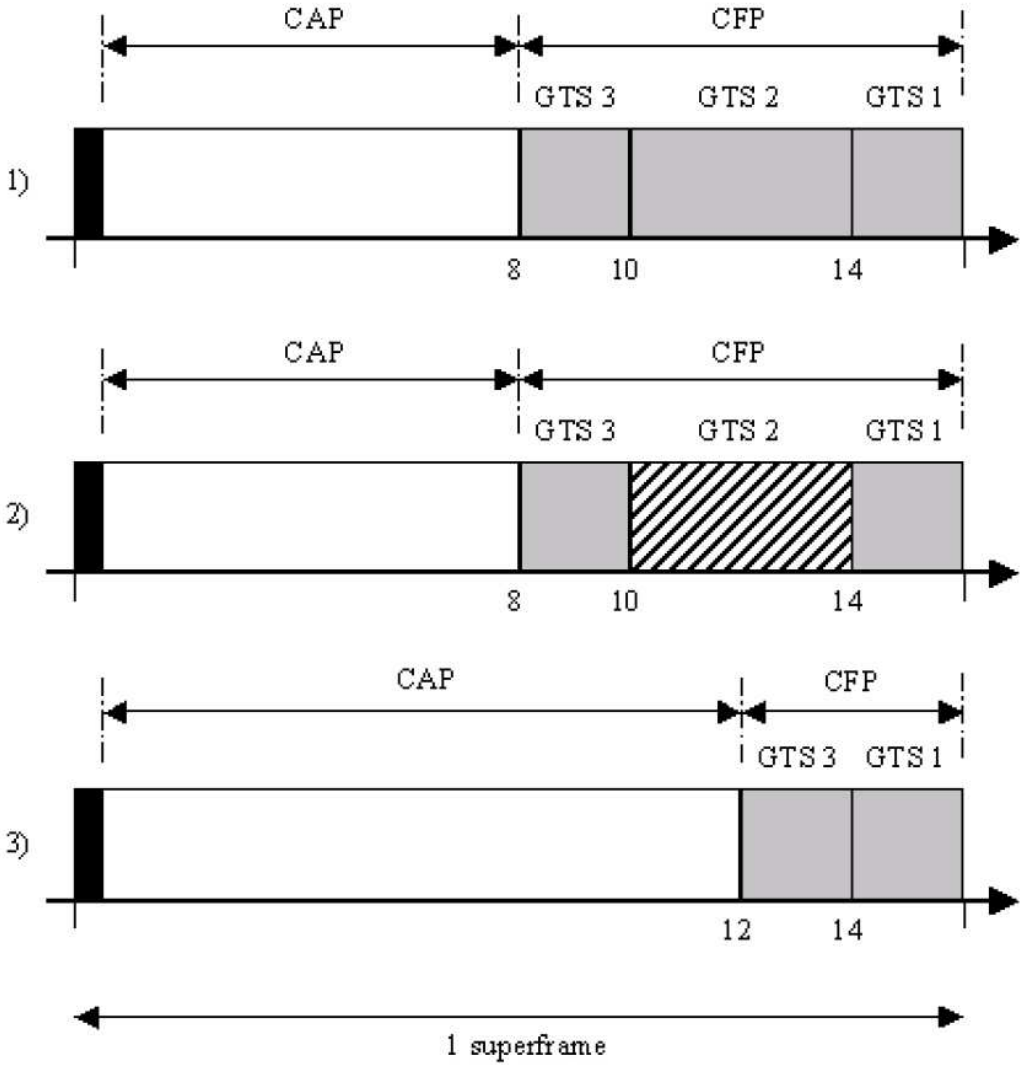

Figure 3.13.: CFP defragmentation on GTS deallocation (taken from [1])

Note

- The GTS direction subfield in the *GTSCharacteristics* is used to determine the direction of the data between the associated device and the coordinator. This subfield should also be checked if a data transmission in a GTS is started(described in Section 3.4.3.1).
- Since the MLME GTS primitives are not implemented in the original MAC Stack of Texas Instruments, an own simple API definition was introduced. However these primitives are very similar to the other predefined primitives and have the full functionality as described in the IEEE standard.

3.4.4.8. MLME Reset

Description The `MAC_MlmeResetReq()` is used to reset the internal state of the MAC and the PHY layer. If the parameter *SetDefaultPIB* is set, the values in the MAC PIB are reset to their initial values.
To reset the internal state of the stack, provided low level primitives are used, which reset the CC2420 and turn off the transmitter and the receiver.

Note This primitive has to be called at least once during the initialization of the device.

3.4.4.9. MLME RX Enable

The MLME RX Enable request is not directly implemented in the API of Texas Instruments. A similar function provides the same functionality and is defined in the `mac_low_level.h` file.

3.4.4.10. MLME Scan

Description The scan primitive is basically used to discover the channel before starting a PAN or to search for an existing PAN to associate with. In general we distinguish between the following four scan modes:

- Energy detection scan
- Passive scan
- Active scan
- Orphan scan

Basic Scan Procedure The basic procedure, as shown in Figure 3.14, is quite similar for each of the scan modes. Basically the backoff timer is set, depending on the parameter *scanDuration* which specifies the time one channel is observed. If the timer event occurs, the actual channel will be set according to the channel list and afterwards the scan is started. The channel list is a 32bit bit field, where each bit specifies whether a channel has to be scanned or not. Since the actual operating mode allows only channel values from 11 to 26, other channels are ignored.
If the next Timer ISR event occurs, the previous scan is stopped and the discovered data (e.g., RSSI, beacon descriptor) are stored in the result vector, afterwards the next channel is set and the scan is started once again, until there is no more channel to scan.

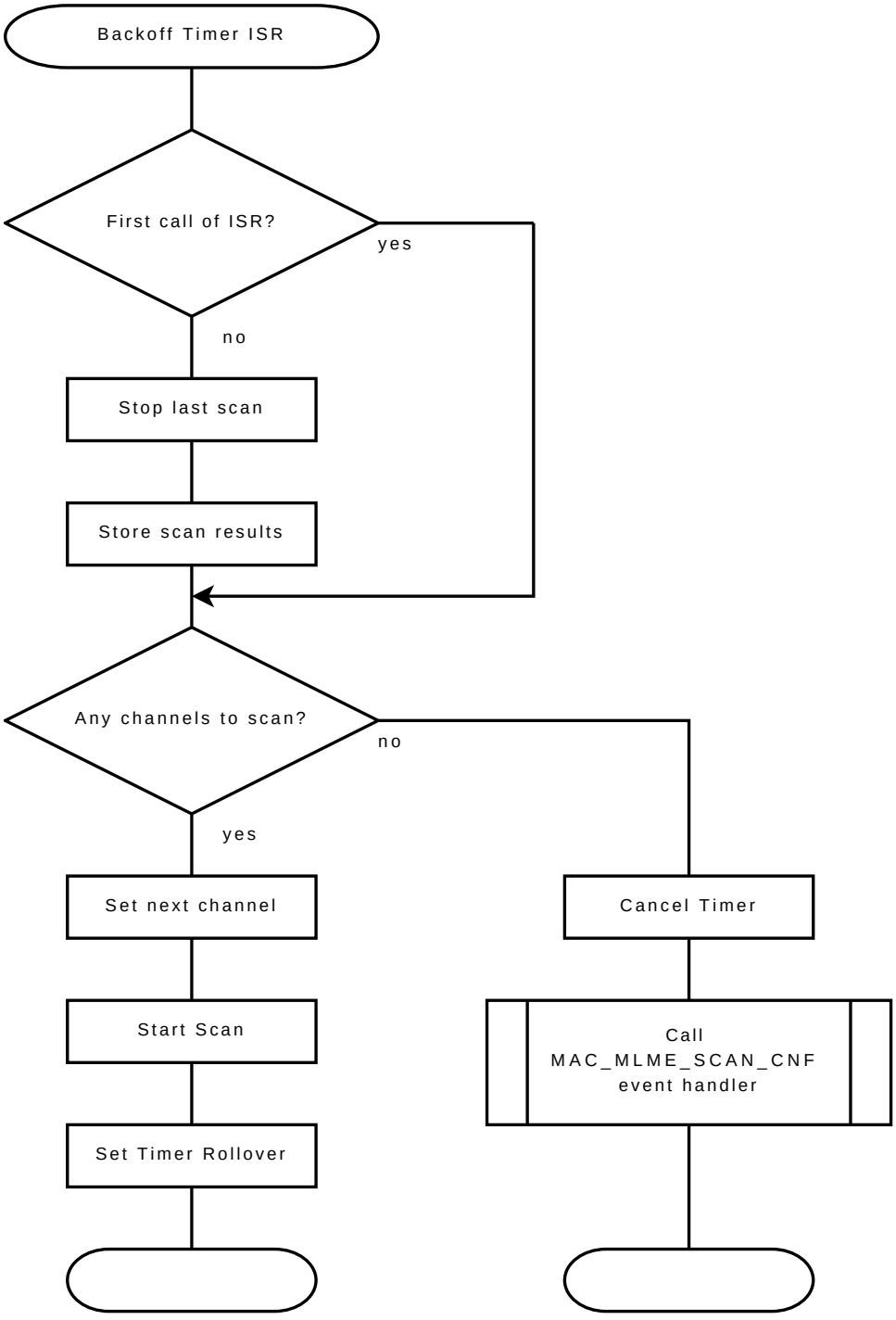

Figure 3.14.: Flow Chart of the basic scan procedure

Scan Modes

- Energy Detection
  In case of an energy detection scan, the internal function starts a timer with an interval of one backoff period. Every time the timer event occurs the RSSI value is read from the CC2420 and the maximum value is stored.
- Passive Scan
  If a passive scan is started, the receiver state machine, as described in Section 3.3.1.2, is advised to receive all beacon frames. If the beacon frame is received, the RX callback function checks, if either the active or passive scan is activated. In such case it decodes the beacon and stores the decoded beacon information in the beacon descriptor result vector. If a beacon is received once again, because of a long scan duration, the beacon is compared, with the stored beacons in the beacon descriptor vector. If the beacon is equal to one of them, it is discard.
- Active Scan
  Before a scan on a channel is started, a beacon request frame is generated (broadcast), which advises all (especially non beacon-enabled) coordinators to send one single beacon. The rest of the scan procedure is similar to the passive scan.

- Orphan Scan
  The orphan is not yet implemented.

Events

- `MAC_MLME_SCAN_CNF`
  If the last channel has been scanned or the maximum value of the result vector is reached, the `MAC_MLME_SCAN_CNF` event is called and the result vector is delivered to the callback function.

Restrictions

- Since the Orphan procedures are not implemented in the current version, the orphan scan is not supported. If an orphan scan is requested, the parameter is ignored and the function returns without any action.
- Because of the limited memory of the MCU and the large size of a beacon descriptor, it is strongly recommended, to limit the result vector to two beacon descriptors, in case of an active or passive scan. In case of the energy detection scan, the result has only one byte.
- Because of an unknown reason, the time interval where the RSSI value is read from the CC2420 is too short, the read operation could not be

finished. Hence the standard value of the timer was adapted by doubling the rollover value of the timer. This changes the timebase of the whole backoff timer which is also used to determine the scan duration. Because of this, the scan duration, which is defined by a number of backoff counts, has to be adapted (divided by 2). After the energy detection scan is finished, the value of the backoff timer rollover has to be reset.

3.4.4.11. MLME Poll

Description The `MAC_MlmePollReq()` primitive is used to request pending data from a coordinator in a non beacon-enabled PAN. This primitive should be called periodically by the next higher layer. If there is no active transmission a data request frame is formed and transmitted to the coordinator using unslotted CSMA-CA.

Events

- `MAC_MLME_POLL_CNF`
  - If the `MAC_MlmePollReq()` primitive is called, it is checked, if there is an active transmission. In this case the `MAC_MLME_POLL_CNF` event is generated with the status `MAC_CHANNEL_ACCESS_FAILURE`.
  - If no acknowledgment frame is received, after a number of *aMaxFrameRetries*, the `MAC_MLME_POLL_CNF` event is generated with the status `MAC_NO_ACK`.
  - If the MLME data request was successfully transmitted and the acknowledgment frame was received by the associated device with the frame pending subfield in the FCF is not set, no data is available at the coordinator. Hence the `MAC_MLME_POLL_CNF` event is generated with the status `MAC_NO_DATA`.
  - If the frame pending subfield in the FCF is set in the received acknowledgment frame, a message frame is available at the coordinator which is transmitted immediately by the coordinator. Hence the `MAC_MLME_POLL_CNF` event is generated with the status `MAC_SUCCESS`.
- Other Events:
  If the `MAC_MLME_POLL_CNF` event was generated with the status `MAC_SUCCESS`, a data frame is available at the coordinator. Depending on the type of the frame, the following types of events are possibly generated:
  - `MAC_MCPS_DATA_IND`
    A detailed description is given in Section 3.4.3.1.

– `MAC_MLME_DISASSOCIATE_IND`
  A detailed description is given in Section 3.4.4.2.

Restrictions As mentioned in Section 3.4.3.1, the indirect data transmission in a non beacon-enabled PAN is not enabled, because the coordinator is blocked until the associated device calls the MLME Poll primitive.

3.5. Demo Application

The demo application is used as prototype of the implemented MAC API. To show a large number of API functions, two different demo applications are implemented. One shows the communication in a non beacon-enabled PAN. The other demo application shows the communication in a beacon-enabled PAN.

3.5.1. Test environment

The demo application shows the communication between two devices, one coordinator and one FFD device, which are represented by two development boards as shown in Figure 3.15.

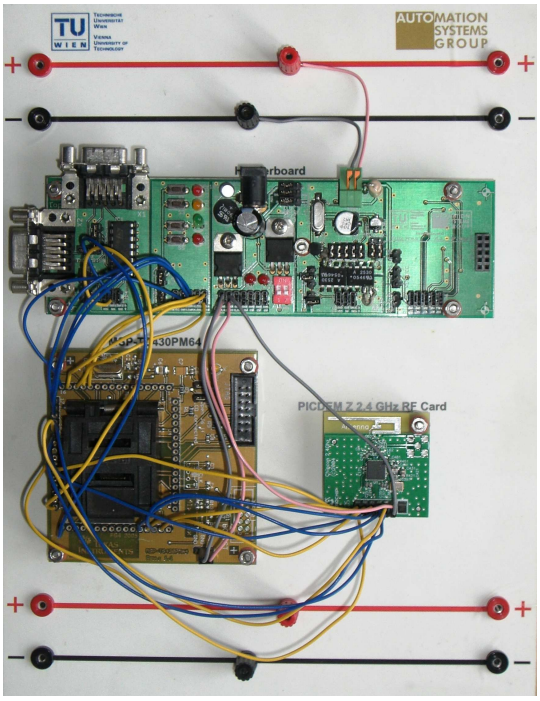

Figure 3.15.: Top view of the demo board

The communication is monitored by a ZigBee USB stick from Integration as shown in Figure 3.16.

Figure 3.16.: ZiBee USB Stick from Integration

The wireless protocol analyzer (WPA)[1] which visualizes all received frames, is able to decode IEEE 802.1.4/ZigBee frames and shows all parts of the frame broken down to each flag and data word as shown in Figure 3.17.

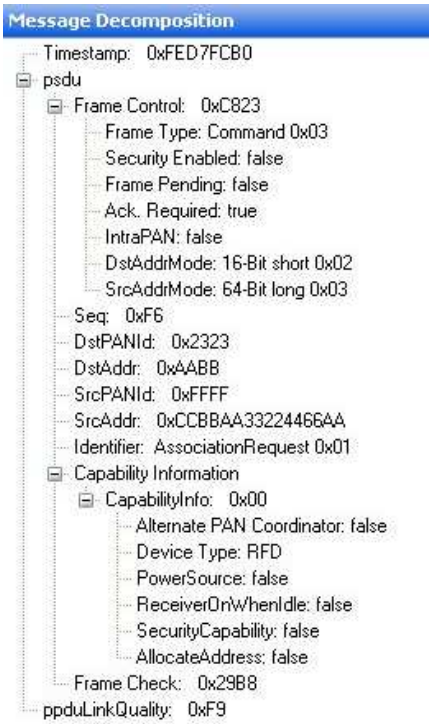

Figure 3.17.: Example of decoded frame of the WPA

Moreover it is possible to enable a filter, which filters only correct frames and discards all others. One of the main advantages of the WPA is timestamping. Since the timestamp of each received frame is recorded, it is possible to prove

[1]The WPA is a commercial product from Integration

the timing of the beacon interval or the data transmission within a GTS. The whole arrangement of the test environment is shown in Figure 3.18.

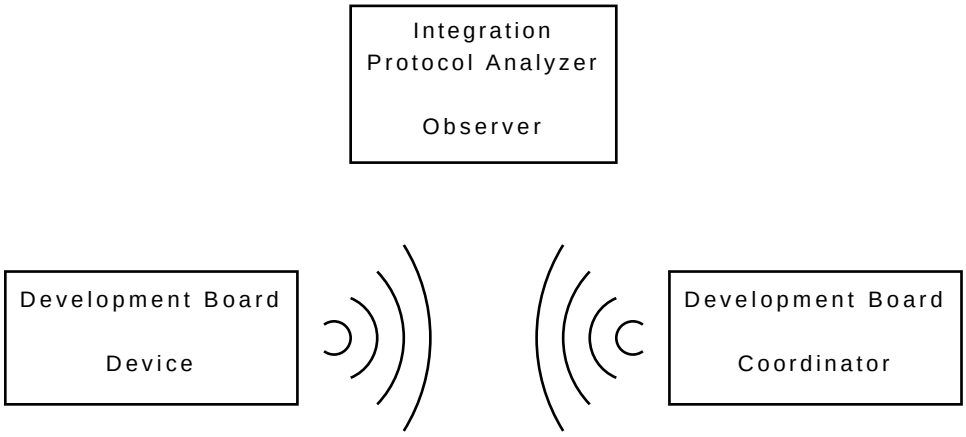

Figure 3.18.: Arrangement of the test environment

3.5.2. Beacon-enabled Demo Application

This demo application shows, how an unassociated device is able to associate with a coordinator and send data in a CFP or in the CAP using indirect transmission.

Scan If a device wants to associate with a coordinator, it first performs an energy detection scan, to detect any traffic on the channel. The energy levels of the selected channels are sent to a text output application over the serial interface to a PC. After the energy detection scan was performed, a passive scan is started to discover a coordinator. Since the corresponding coordinator has a $BeaconOrder = 6$, the *scanDuration* is set to a value of 7. If a beacon is received from the coordinator, the `MAC_MLME_SCAN_CNF` event is generated, because the scan result is limited to one. In this event handler the device reads the coordinator address and its PAN ID from the beacon descriptor and starts the association procedure by calling the primitive `MAC_MlmeAssociationReq()`.

Association The `MAC_MlmeAssociationReq()` primitive sends an association request frame to the coordinator as described in Section 3.4.4.1. At the coordinator the `MAC_MLME_ASSOCIATION_IND` event is generated. The coordinator assigns a predefined address to the device and sends the response to the device by calling the `MAC_MlmeAssociationRsp()` primitive. If the device receives the association response from the coordinator, it updates its MAC PIB.

Indirect MCPS transmission After the association was finished, the coordinator wants to send a data frame to the device using indirect transmission, by calling the `MAC_McpsDataReq()` primitive with the *txOption* `MAC_TXOPTION_INDIRECT`. The coordinator adds the address of the device to the pending addresses field of the beacon frame and waits for a MLME data request frame. Since the *autoRequest* parameter in the MAC PIB of the associated device is set, the device automatically transmits the MLME data request frame. If the coordinator receives this frame, it transmits the MCPS data frame. If the associated device receives this frame the `MAC_MCPS_DATA_IND` event is generated and the payload is written to the UART.

GTS Request After that the associated device initiates a GTS transmission, but before a frame could be sent, a GTS has to be allocated by using the `MAC_MlmeGtsReq()` primitive. If the coordinator receives the GTS request of the device, it allocates a GTS and updates the beacon frame. If the associated device receives the beacon frame, in which a GTS is allocated for the device the `MAC_MLME_GTS_CNF` event is generated and now the `MAC_McpsDataReq()` primitive can be called.

MCPS GTS transmission The `MAC_McpsDataReq()` primitive calculates the backoff count of the allocated GTS and triggers the backoff timer. If the trigger is fired, the requested data frame is transmitted and the `MAC_MCPS_DATA_IND` event is generated at the coordinator. In this event handler the transmitted data are written to the UART.
After that transmission the coordinator tries to disassociate the device by calling the `MAC_MlmeDisassociationReq()` primitive.

Disassociation The `MAC_MlmeDisassociationReq()` primitive adds the address of the device to the pending addresses field of the beacon frame and waits for a MLME data request. If the coordinator receives the MLME data request, the disassociation notification frame is transmitted to the device.
If the device receives the disassociation notification frame, the `MAC_MLME_DISASSOCIATION_IND` event is generated and the demo applications is stopped. On the other side the coordinator transmits its beacons anyway and waits for another association request.
A screenshot of the WPA shows the sequence of message frames which are transmitted in the demo application (Figure 3.19).

Summary View of Received Packets

No	Timestamp	Dest PAN	Dest Addr	Src PAN	Src Addr	Detail	Additional Info
11	212.229.589,60			0x2323	0xAABB	Beacon	
12	213.216.389,60			0x2323	0xAABB	Beacon	
13	213.778.389,60	0x2323	0xAABB	0xFFFF	0xCCBBAA3322...	Command	AssociationRequest
14	213.780.689,60					Acknowledgement	Seq: 0xF6
15	214.219.989,60			0x2323	0xAABB	Beacon	1 pending messages
16	214.239.489,60	0x2323	0xAABB	0x2323	0xCCBBAA3322...	Command	DataRequest
17	214.241.989,60					Acknowledgement	Seq: 0x00
18	214.261.689,60	0x2323	0xCCBBAA3322...	0x2323	0x6565426F6C6...	Command	AssociationResponse
19	214.263.789,60					Acknowledgement	Seq: 0x67
20	000.486.324,80			0x2323	0xAABB	Beacon	1 pending messages
21	000.504.424,80	0x2323	0xAABB	0x2323	0xEAEA	Command	DataRequest
22	000.506.824,80					Acknowledgement	Seq: 0x00
23	000.520.324,80	0x2323	0xEAEA		0xAABB	Data	Seq: 0x68
24	000.522.724,80					Acknowledgement	Seq: 0x68
25	000.537.024,80			0x2323	0xEAEA	Command	GTSRequest
26	000.539.424,80					Acknowledgement	Seq: 0x01
27	001.502.624,80			0x2323	0xAABB	Beacon	
28	002.441.224,80	0x2323	0xAABB		0xEAEA	Data	Seq: 0xF7
29	002.443.524,80					Acknowledgement	Seq: 0xF7
30	002.501.024,80			0x2323	0xAABB	Beacon	1 pending messages
31	002.519.624,80	0x2323	0xAABB	0x2323	0xEAEA	Command	DataRequest
32	002.522.124,80					Acknowledgement	Seq: 0x00
33	002.535.924,80	0x2323	0xEAEA		0xAABB	Command	DisassociationNotification
34	002.538.424,80					Acknowledgement	Seq: 0x00
35	003.508.124,80			0x2323	0xAABB	Beacon	
36	004.494.624,80			0x2323	0xAABB	Beacon	
37	005.481.324,80			0x2323	0xAABB	Beacon	
38	006.468.024,80			0x2323	0xAABB	Beacon	
39	007.454.824,80			0x2323	0xAABB	Beacon	
40	008.441.524,80			0x2323	0xAABB	Beacon	
41	009.428.324,80			0x2323	0xAABB	Beacon	
42	010.415.024,80			0x2323	0xAABB	Beacon	

Figure 3.19.: Screenshot of the WPA, showing the packets of the demo application

3.5.3. GTS verification

To check the correctness of the calculation of the GTS, the timestamps of the WPA are used as shown in Figure 3.20. A schematic view of the analyzed beacon interval is shown in a timeline diagram (Figure 3.21). The time difference between two beacon transmissions is

$$z = 2501024.8 - 1502624.8 = 998400$$

and the start time of the GTS transmission within the superframe is

$$x = 2441224.8 - 1502624.8 = 938600$$

If assumed that the normalized beacon interval is

$$\bar{z} = 16$$

the normalized time of data transmission in the superframe is

$$\bar{x} = \frac{x}{z} \cdot \bar{z} = 15,041$$

and this value is correct.

27	001.502.624,80			0x2323	0xAABB	Beacon	
28	002.441.224,80	0x2323	0xAABB		0xEAEA	Data	Seq: 0xF7
29	002.443.524,80					Acknowledgement	Seq: 0xF7
30	002.501.024,80			0x2323	0xAABB	Beacon	1 pending messages

Figure 3.20.: GTS timing sequence

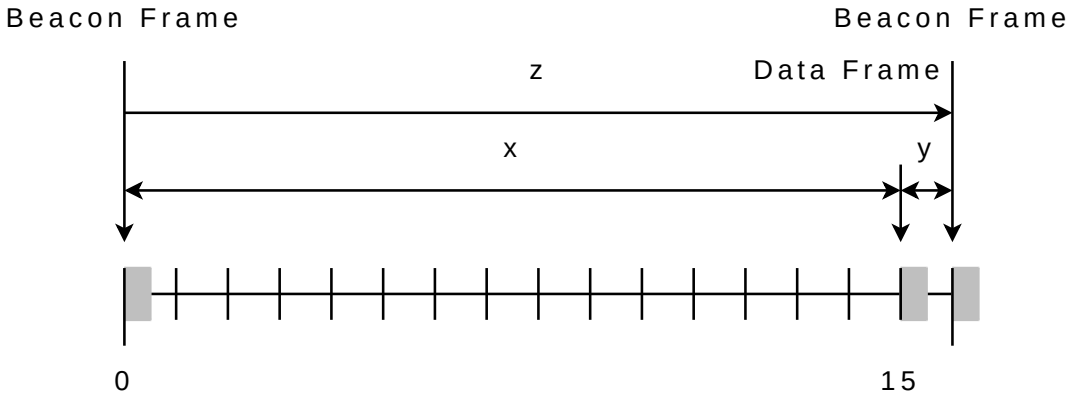

Figure 3.21.: Superframe timeline diagram

3.5.4. Non beacon-enabled Demo Application

The demo application in a non beacon-enabled PAN is quite shorter than in a beacon-enabled PAN because only unslotted CSMA-CA transmissions are possible.

Initialization After the initialization of the devices, the coordinator is started with the `MAC_MlmeStartReq()` primitive, which updates all necessary parameters in the MAC PIB and listens on the channel.

Active scan The device which should be associated with the coordinator starts an active scan after the initialization, using the `MAC_MlmeScanReq()` primitive, which is described in Section 3.4.4.10. In this scan mode the device first sends a beacon request frame on each channel, before the scan is started.

If the coordinator receives a beacon frame request, it transmits only one single beacon. If the scanning device receives this beacon, the scan is terminated by generating the `MAC_MLME_SCAN_CNF`, because the result list is limited to one.

Association After the active scan was finished, the coordinator address and the PAN ID are updated in the *macPib* and then the association procedure is started by calling the `MAC_MlmeAssociationReq()`. In this primitive the device transmits an association request frame to the coordinator. If the coordinator receives the association request frame, it generates the `MAC_MLME_ASSOCIATION_IND` event. The coordinator now assigns a static short address to the device and calls the `MAC_MlmeAssociationRsp()`. This primitive forms the association response frame and waits until a MLME data request frame is received from the device. After a specified time, which is described in Section 3.4.4.1, the devices transmits a MLME data request frame. If the coordinator receives that frame, it transmits the actual association response frame to the device and if this frame is received by the device the `MAC_MLME_ASSOCIATION_CNF` event is generated and the device is successfully associated.

MCPS transmission After updating the short address with the assigned address from the coordinator, a normal data transmission is started by calling the `MAC_McpsDataReq()`. This primitive transmits a data frame to the coordinator. If this frame is received by the coordinator the `MAC_MCPS_DATA_IND` event is generated and the payload of the frame is written to the UART.

Disassociation After the `MAC_MCPS_DATA_CNF` event occurs at the device, the device is going to disassociate itself from the coordinator by calling the `MAC_MlmeDisassociationReq()`. This primitive generates a disassociation notification frame and transmits it to the coordinator. After the acknowledgment frame was returned by the coordinator the `MAC_MLME_DISASSOCIATION_IND` is generated and the coordinator waits for another association request. If the device receives the acknowledgment frame it generates the `MAC_MLME_DISASSOCIATION_CNF` event and stops its operation.

4. Conclusion

This thesis focused on the implementation of an IEEE 802.15.4 protocol stack. Since the MAC sublayer of the IEEE 802.15.4 stack was not published by Texas Instruments, it has to be reimplemented. The main goal of this implementation was to make the MAC sublayer compatible to the original implementation of Texas Instruments. The realization is according to the API interface description of Texas Instruments and the IEEE 802.15.4 standard.

The current implementation of the IEEE 802.15.4 MAC API is the cornerstone for a large number of applications and future projects. So the actual implementation could be directly used to realize simple control networks or time triggered applications. It is also possible to realize simple point to point applications. If ZigBee is not used, a simple routing protocol can be implemented, to make mesh networks possible.

Since the Chipcon CC2420 is not fully compliant to the IEEE 802.15.4 standard and the MCU has limited memory capabilities, neither large applications nor real IEEE 802.15.4 compliant applications can be implemented. In future projects the successor of the Chipcon CC2420, the CC2520 can be used in combination with the MSP430F2618 MCU, which has 8KB memory. Moreover the Chipcon CC2520 is claimed to be fully IEEE 802.15.4 compatible in contrast to the CC2420. If this hardware configuration is used, the provided functions of Texas instruments have to be replaced, and a few compiler specific macros have to be redefined, which are quite similar to the changes in the actual implementation. Finally, it should be possible to make this hardware configuration running and then the actual implementation could be extended by using the larger memory.

It is also possible to set up the ZigBee stack from Texas Instruments because the MAC API should be compliant to the IEEE 802.15.4 stack from Texas Instruments.
Another possibility is to port the whole IEEE 802.15.4 stack to another MCU type (e.g. the Atmel ATMega series). This is even possible, but many changes have to be done (e.g, changing timer functions, USART functions and the memory management).
If security is additionally implemented in future projects, also security applications, like door control systems or alarm systems could be realized with this implementation.

Bibliography

[1] IEEE, 3 Park Avenue, New York, NY 10016-5997, USA, *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, September 2006.

[2] ZigBee Alliance Inc., 2400 Camino Ramon Suite 375, San Ramon CA 94583, USA, *ZigBee Specification*, January 2008.

[3] A. Koubâa, M. Alves, and E. Tovar, "Ieee 802.15.4 for wireless sensor networks: A technical overview," tech. rep., IPP-HURRAY! Polytechnic Institute of Porto (ISEP-IPP), Rua Dr. António Bernardino de Almeida431, 4200-072 Porto, PORTUGAL, July 2005.

[4] S. Y. Shin and S. Choi, "Packet error rate analysis of zigbee under wlan and bluetooth interferences," *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS*, vol. 6, August 2007.

[5] Freescale Semiconductor Inc., 1300 N. Alma School Road, Chandler, Arizona 85224, USA, *IEEE 802.15.4 / ZigBee Software Selector Guide*, September 2007.

[6] Texas Instruments, San Diego, CA USA, *802.15.4 MAC Application Programming Interface*, March 2007. Available at `http://focus.ti.com/docs/toolsw/folders/print/timac.html`.

[7] Texas Instruments Norway AS, Gaustadalléen 21, NO-0349 Oslo, NORWAY, *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, November 2007. Available at `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`.

[8] Microchip Technology Inc., 2355 West Chandler Blvd., Chandler, AZ 85224-6199, USA, *PICDEM Z DEMONSTRATION KIT USER'S GUIDE*, October 2004.

[9] Texas Instruments, San Diego, CA USA, *Texas Instruments MSP-FET430P140 Flash Emulation Tool User's Guide*, April 2001.

A. Acronyms and abbreviations

API application programming interface

APL Application layer

BE beacon exponent

CAP contention access period

CCA clear channel assessment

CFP contention-free period

CSMA-CA carrier sense multiple access with collision avoidance

CW contention window

DLL data link layer

FCF frame control field

FFD full-function device

GTS guaranteed time slot

HAL hardware abstraction layer

HART Highway Addressable Remote Transducer

HVAC heating, ventilating, and air conditioning

IP Internet Protocol

ISO International Organization for Standardization

LAN local area network

LCD liquid crystal display

LLC logical link control

LQ link quality

MAC medium access control

MCPS MAC common part sublayer

MCU microcontroller unit

MHR MAC header

MLME MAC sublayer management entity

mspgcc GNU C compiler for MSP430 MCUs

NB number of backoffs

NWK Network layer

OSAL operating system abstraction layer

OSI Open Systems Interconnection

PAN personal area network

PCB printed circuit board

PIB PAN information base

PHR PHY header

PHY physical layer

PPDU PHY protocol data unit

RFD reduced-function device

RSSI received signal strength indication

SFD start frame delimiter

SFS superframe specification

SHR synchronisation header

SPI serial peripheral interface

SSCS service specific convergence sublayer

UART universal asynchronous receiver transmitter

WPA wireless protocol analyzer

WLAN wireless local area network

B. Board Documentation

B.1. Documentation of the RF Board

Figure B.1.: Top view of the RF Board from Microchip (take from [8])

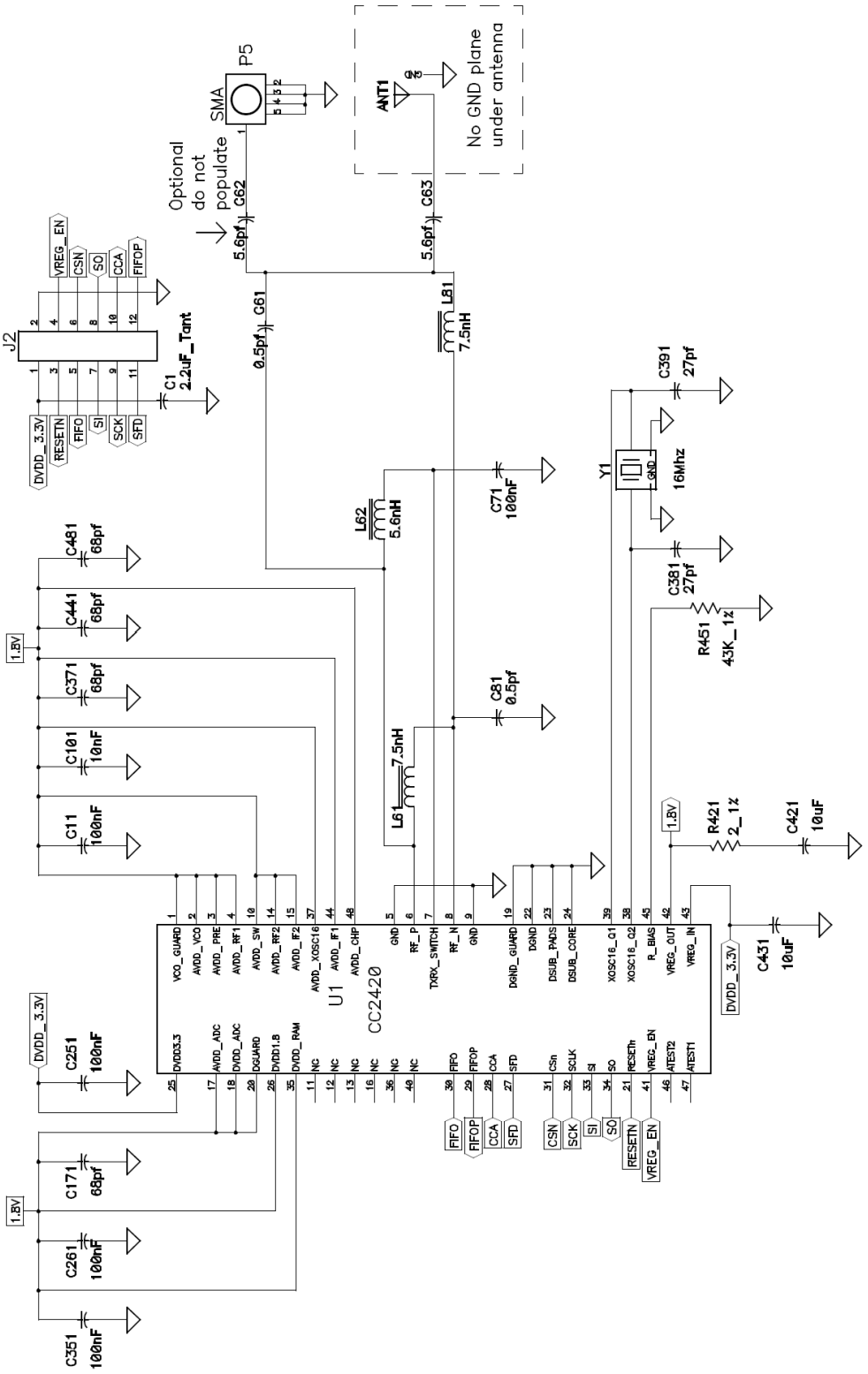

Figure B.2.: Schematic of the RF Board from Microchip (take from [8])

B.2. Schematic of the MCU Board

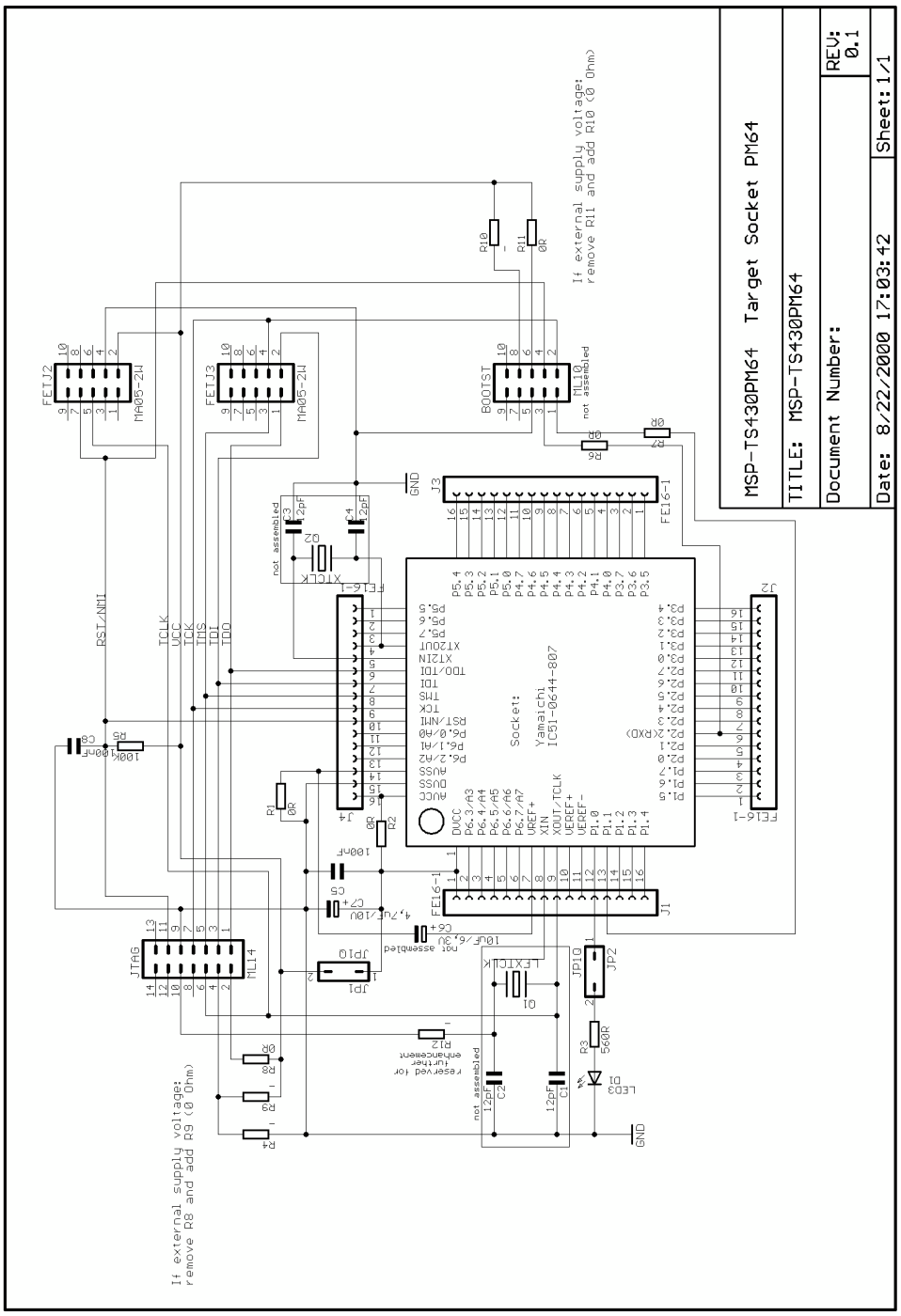

Figure B.3.: Schematic of the MCU Board (take from [9])