CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

BEng Thesis

Towards an implementation of the IEEE 802.15.4 time critical MAC extensions over a real-time OS

Pedro Neto

CISTER-TR-181130

Towards an implementation of the IEEE 802.15.4 time critical MAC extensions over a real-time OS

Pedro Neto

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Growing at a fast pace, the adoption of embedded computing systems, capableof monitoring and controlling the physical environment around them, is spreadingacross different environments, from our houses to the industrial setting. Thislatest trend, induced by recent advancements in the information and communicationfields, craves for the new Wireless Sensor Network (WSN) concept, whichaims to enable an infrastructure that interconnects the set of widespread 1cintelligent 1ddevices, capable of wireless communications, constituting a network ofsensor nodes.On an Industrial context, the new Cyber-Physical Systems (CPS), promptedby the Industry 4.0 revolution, aim to correlate the present automated systemsto the new information technologies, such as cloud and cognitive computing,to compose a group of collaborative computing systems that enact the 1csmartfactory 1d. This new targeted model however, relies on certain time assurances andother QoS (Quality of Service) properties such as scalability, energy efficiencyand robustness, which WSN technologies intent to grant. Although a tenderparadigm, propositions such as the IEEE std. 802.15.4 protocol ambition toenable the WSN infrastructure and satisfy the QoS requirements.The IEEE std. 802.15.4 protocol provides several MAC (Medium Accesscontrol) behaviours to frame the communications stack, each aiming to meetthe set of requirements of distinct applications. For deterministic latency, highreliability and scalability QoS requirements, IEEE 802.15.4 standard provides theDSME(Deterministic Synchronous Multichannel Extension) MAC behaviour.Parallel to the phenomenon of the WSN technologies, real-time operatingsystems (RTOS) are emerging among the IoT(Internet of Things) community tohelp tackle QoS specifications for determinism and time-critical constraints.The use of a real-time OS in conjunction with a time reliable protocol suchas DSME is the key to enable a truly deterministic and time critical WSN.However, besides these settings, QoS at the computing platform must beguaranteed as well if these network infrastructures are to become a reality. Computationsmust be performed in a predictable way, as to support the QoS demandsin terms of latency these networks present. Hence, in this Thesis we propose torely on the FreeRTOS for a real-time operating system and a well known WSNplatform, such as the TelosB to implement the DSME time critical MAC behaviour.To achieve this defined goal, this Thesis presents a port of FreeRTOS tothe TelosB platform, wich includes an IEEE 802.15.4 compliant radio, as well as apreliminary study of the future implementation strategy of the network protocol.Hereby, this Thesis concludes with a successful implementation of the RTOS,FreeRTOS, for the TelosB platform, along with the necessary groundwork andtime requirements support for the DSME extension. Additionally the Thesisprovides a suggested model for the protocol stack to fit the FreeRTOS task system.

Towards an implementation of the IEEE 802.15.4 time critical MAC extensions over a real-time OS

Pedro Alexandre Afonso Neto

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2018

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Projeto/Estágio, do 3º ano, da Licenciatura em Engenharia Eletrotécnica e de Computadores

Candidato: Pedro Alexandre Afonso Neto, Nº 1150386, 1150386@isep.ipp.pt

Orientação científica: Dr. Ricardo Augusto Rodrigues da Silva Severino (PhD), rar@isep.ipp.pt
Co-orientação: John Harrison Kurunathan (MSc), hhkur@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2018

Acknowledgements

I would like to begin by thanking my family for all the support that helped achieve this milestone.

I also want to thank professor Dr. Ricardo Severino for all the guidance, dedication and moral support required for the completion of this project. Furthermore, I want to thank my co-supervisor Harrison Kurunathan and all my colleagues at CISTER for all the provided assistance and enjoyable fun moments.

Last but not least, I want to thank my friends and colleagues at ISEP for helping me out when I most needed it, as well as professor Benedita Malheiro for her insightfull feedback on the writing of this thesis.

Abstract

Growing at a fast pace, the adoption of embedded computing systems, capable of monitoring and controlling the physical environment around them, is spreading across different environments, from our houses to the industrial setting. This latest trend, induced by recent advancements in the information and communication fields, craves for the new Wireless Sensor Network (WSN) concept, which aims to enable an infrastructure that interconnects the set of widespread "intelligent" devices, capable of wireless communications, constituting a network of sensor nodes.

On an Industrial context, the new Cyber-Physical Systems (CPS), prompted by the Industry 4.0 revolution, aim to correlate the present automated systems to the new information technologies, such as cloud and cognitive computing, to compose a group of collaborative computing systems that enact the "smart factory". This new targeted model however, relies on certain time assurances and other QoS (Quality of Service) properties such as scalability, energy efficiency and robustness, which WSN technologies intent to grant. Although a tender paradigm, propositions such as the IEEE std. 802.15.4 protocol ambition to enable the WSN infrastructure and satisfy the QoS requirements.

The IEEE std. 802.15.4 protocol provides several MAC (Medium Access control) behaviours to frame the communications stack, each aiming to meet the set of requirements of distinct applications. For deterministic latency, high reliability and scalability QoS requirements, IEEE 802.15.4 standard provides the DSME(Deterministic Synchronous Multichannel Extension) MAC behaviour.

Parallel to the phenomenon of the WSN technologies, real-time operating systems (RTOS) are emerging among the IoT(Internet of Things) community to help tackle QoS specifications for determinism and time-critical constraints.

The use of a real-time OS in conjunction with a time reliable protocol such as DSME is the key to enable a truly deterministic and time critical WSN.

However, besides these settings, QoS at the computing platform must be guaranteed as well if these network infrastructures are to become a reality. Computations must be performed in a predictable way, as to support the QoS demands in terms of latency these networks present. Hence, in this Thesis we propose to rely on the FreeRTOS for a real-time operating system and a well known WSN platform, such as the TelosB to implement the DSME time critical MAC behaviour. To achieve this defined goal, this Thesis presents a port of FreeRTOS to the TelosB platform, wich includes an IEEE 802.15.4 compliant radio, as well as a preliminary study of the future implementation strategy of the network protocol.

Hereby, this Thesis concludes with a successful implementation of the RTOS, FreeRTOS, for the TelosB platform, along with the necessary groundwork and time requirements support for the DSME extension. Additionally the Thesis provides a suggested model for the protocol stack to fit the FreeRTOS task system.

Keywords

Wireless Sensor Networks; IEEE std. 802.15.4-2015; DSME MAC behaviour; TelosB mote platform; FreeRTOS.

Contents

List of Figures

List of Tables

List of Acronyms

BI	Beacon Interval
BO	Beacon Order
CAP	Contention Access Period
CCA	Clear Channel Assessment
CFP	Contention Free Period
CSMA	Carrier Sense Multiple Access
CSMA-CA	CSMA - Collision Avoidance
CPS	Cyber-Physical Systems
CPU	Central processing unit
DSME	Deterministic Synchronous Multichannel Extension
ED	Energy Detection
FCS	Frame Check Sequence
FFD	Full Function Device
FIFO	First in First out
GTS	Guaranteed Time Slot
GUI	Graphical User Interface
HVAC	Heating, ventilation and air conditioning
IE	Information Element
IEEE	Institute of Electrical and Electronics Engineers
IEEE-SA	IEEE Standards Association
IETF	Internet Engineering Task Force
IoT	Internet of Things
LED	Light-emitting Diode
LQI	link quality indication
LR-WPAN	Low Rate-Wireless Personal Area Network
MAC	Medium Access Control
MAC-PIB	MAC PAN information Base
MCPS	MAC Common part sublayer
MCU	Microcontroller Unit
MLME	MAC layer management entity
MO	Multi-superframe Order
NWK	Network Layer
OSI	Open Systems Interconnection

PAN	Personal Area Network
PHY	Physical Layer
PHY-PIB	PHY PAN information Base
PLME	Physical Layer Management Entity
QoS	Quality of Service
LR-WPAN	Low Rate - Wireless Personal Area Network
RAM	Random-access memory
RF	Radio Frequency
RFD	Reduced Function Device
RSSI	Received Signal Strength Indication
RTOS	Real-Time Operating System
RX	Receive or Receiver
SAP	Service Access Point
SD	Superframe Duration
SFD	Start-of-Frame Delimiter
SO	Superframe Order
SPI	Serial Peripheral Interface
TX	Transmit or Transmitter
USART	Universal Synchronous/Asynchronous Rx/Tx
USB	Universal Serial Bus
WPAN	Wireless Personal Area Network

Chapter 1

Introduction

1.1 Overview

Arguably the greatest technological leap of the modern society, the Internet connected people and allowed for an unprecedented share of information worldwide. As a consequence, the concept of Internet of Things (IoT), grew from the drive to interconnect the available sensors and actuators, enabling "smart" devices, with a growing presence in everyday items, demanded for the Wireless Sensor Network (WSN) paradigm, which takes advantage of the recent advancements in information and communication technology [1], to serve as the underlying communications infrastructure.

Inspired by the new advents, we become ever eager to automate the world around us, by monitoring and controlling everything, everywhere. The trend spread to the industry context, triggering a revolution that lead to the fourth iteration of the manufacturing world, the Industry 4.0 [2].

The new tendency to interconnect computational and physical entities from the third industrial rendition, leading to the Cyber-Physical System (CPS) conception. Coined from the newly bridged interaction between the physical surroundings and accessible data for monitoring and control, the emerging CPS can be applied throughout a variety of fields such as building automation (e.g., security, climate and lighting control, access control), industrial automation (e.g. asset management, process control, environmental control, energy management, preventive maintenance) and personal health care (e.g., body sensor networks).

These applications however, must be conceived in a way that the quality of the service (QoS), appraised by either monitoring systems or through a direct user interaction, meet the necessary requirements for either practicality or even

human safety."QoS is thus usually associated with bit rate, network throughput, message end-to-end delay and bit error rate" [1].

WSN technology aims to be a suitable solution to supply the CPS domain with the support to cope with the stringent QoS properties.

To fulfil the QoS needs of industrial communication over the last decade, the Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA) are continuously working on the IEEE 802.15.4 standard[7], for Low Rate Wireless Personal Area Networks (LR-WPAN). The growing popularity of the Industry 4.0 concept, which converges IoT, Cyber Physical Systems (CPS) and Cloud technologies at an industry level, generated overgrowing demands by the industrial domain and emerging CPS systems for low-power, low-range, and robust wireless communication. The IEEE 802.15.4 therefore provides several MAC behaviours, such as deterministic communication and multi-channel frequency hopping mechanisms, improving the predictability, reliability and latency of this protocol. Targeting deterministic latency, high reliability and scalability QoS requirements, IEEE 802.15.4 standard includes the DSME MAC behaviour.

To help conform these requirements, the IoT community has also pointed at the emerging real-time operating systems (RTOS), with specialized schedulers to meet the set time deadlines and provide determinism to demanding applications, such as CPS derived. The use of a real-time OS along with time reliable protocol such as the DSME will enable a truly deterministic WSN infrastructure.

However the required QoS at the computing platform must be met as well to guarantee determinism for applications. Computations must be performed in a predictable way, as to support the QoS demands in terms of latency these networks present.

To tackle this issue, in this Thesis we propose to entrust FreeRTOS for the real-time operating system as well as the TelosB for the target hardware platform in order to implement the DSME time critical MAC behaviour. In order to carry out the proposed task, this Thesis presents a port of FreeRTOS to the TelosB platform, wich includes an IEEE 802.15.4 compliant radio. Additionally, the envisaged architecture for the implementation of the IEEE 802.15.4 is provided, detailing strategy to achieve the set goals.

1.2 Research Context

This Thesis is being carried out at the CISTER Research Centre in Real-Time Embedded Computing Systems, aligned within several research efforts which aim at improving the Quality of Service of the supporting communication infrastructures, to enable future Cyber-Physical systems, by relying as much as possible on commercial off-the-shelf technology.

1.3 Research Objectives

This Thesis aims at providing support for a reliable implementation of the IEEE 802.15.4 standard, namely the DSME MAC behaviour, on top of a real time operating system, taking the first steps towards building a truly deterministic WSN and enabling support for applications with time-critical, reliable and scalable QoS requirements.

1.4 Research Contributions

The main contributions from this Thesis are the provided support grounds for the implementation of the IEEE std. 802.15.4 over FreeRTOS; the provision TelosB mote support and drivers for the FreeRTOS; additionally, this Thesis provides the design plans for a modular, efficient and accessible implementation of the IEEE 802.15.4 on FreeRTOS.

1.5 Structure of this thesis

The Thesis is composed of five chapters apart from the current, which are organized as follows: The second chapter introduces the IEEE std. 802.15.4 protocol, detailing its most relevant features, taking a particular look into the DSME MAC behaviour. Then, Chapter three presents the technologies required for the implementation, apart from the standard, including the TelosB hardware platform and the real-time system FreeRTOS. Additionally, some support tools used during this Thesis are described. Chapter 4 details the full implementation of the FreeRTOS over TelosB including the code development structure built using GNU make for developing applications with the OS for the TelosB platform, concluding with a experimental validation in order to legitimize the implementation. Chapter 5 presents the planned approach for implementing the IEEE 802.15.4, including a proposed application stack with all involved modules as well a study on how to build the DSME-Superframe, concluding with expected results regarding the OS scheduler. The Thesis concludes with Chapter 6, which summarizes the results and presents the future plans for the protocol implementation.

Chapter 2

Overview of IEEE 802.15.4 and DSME behaviour

This chapter presents the most important features of the IEEE 802.15.4-2015 protocol, and in particular of the DSME MAC behaviour. Even though the protocol is not completely implemented in this Thesis, it is crucial to fully understand its features and requirements to carry out the appropriate implementation.

2.1 Overview

There are several wireless communication protocols that accommodate different types of applications like video, voice and general data communications. Each one of these protocols sets a trade-off between properties such as throughput, latency, energy efficiency and radio coverage targeting well defined application scenarios. A wireless sensor network (WSN) consists of wirelessly communicating, spacially distributed, RF (radio frequency) devices, capable of monitoring the physical properties of their set environment. A WSN usually does not impose stringent requirements in terms of bandwidth, but does require minimized energy consumption so that the overall network lifetime is prolonged. Meeting the Quality of Service (QoS) requirements such as energy efficiency and timeliness is amongst the main objectives of WSN protocols and technologies [5].

To fulfil the QoS needs of industrial communication over the last decade, the Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA), an organization within IEEE, tasked with developing global standards, published in 2003, the IEEE 802.15.4 standard [8], for Low Rate - Wireless Personal Area Networks (LR-WPAN).

Since then, the booming popularity of the Industry 4.0 concept, that converges IoT (Internet of Things), Cyber Physical Systems (CPS) and Cloud technologies at an industry level, generated overgrowing demands by the industrial domain and emerging CPS systems for low-power, low-range, and robust wireless communication. In 2012, following the second revision of the standard (IEEE Std 802.15.4-2011), the IEEE-SA published the IEEE 802.15.4e amendment[9], aiming at enhancing and extending the functionalities of the original protocol. It added various innovative features, grouped into different MAC behaviours, such as deterministic communication and multi-channel frequency hopping mechanisms, improving the predictability, reliability and latency of this protocol. In 2016, the amendment was merged into the standard, bringing the new MAC behaviours and their features into the IEEE Std 802.15.4-2015.

2.2 General Aspects

The standard [7] specifies the physical layer (PHY) and the medium access control (MAC) sublayer, respective to the Physical and Data Link layers of the Open Systems Interconnection (OSI) model, aiming at achieving wireless connectivity with low complexity, low power consumption and low data rate, for portable and moving devices.

Depending on the application requirements, IEEE 802.15.4 LR-WPAN devices are inter-connected following one of two topologies, the star topology or a peer-to-peer topology, represented in Figure 2.1.

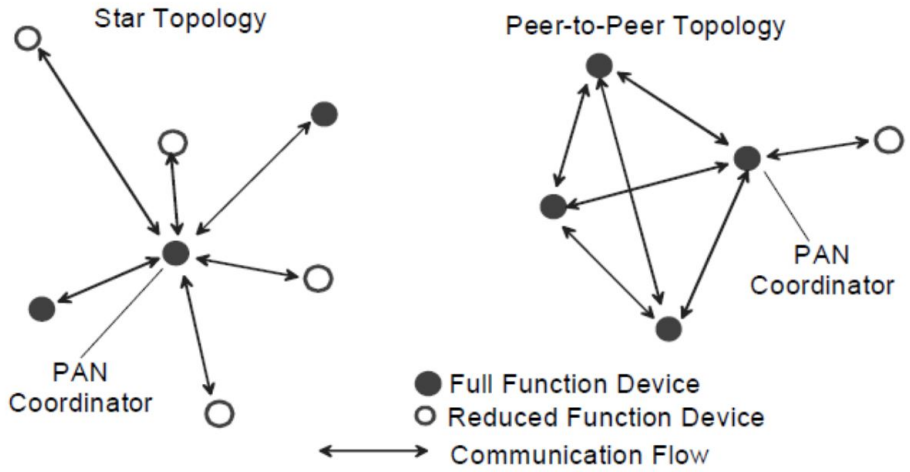


Figure 2.1: Star and peer-to-peer topology examples[7]

In a star LR-WPAN, the communication is established between devices and a single central a unique node(RF device), called the personal area network (PAN) coordinator which is the primary controller of the LR-WPAN. Applications that

benefit from a star topology include home automation, personal computer (PC) peripherals, games, and personal health care, which typically take advantage of the centralized formation.

The peer-to-peer topology also has a PAN coordinator, however, it differs from the star topology in that any device is able to communicate with any other device as long as they are in range of one another. Peer-to-peer topology allows more complex network formations to be implemented, such as mesh networking topology that benefit applications such as industrial control and monitoring WSN, asset and inventory tracking, intelligent agriculture, and security would benefit from such a network topology [7].

Devices on a LR-WPAN can be classified into Fully Function Devices (FFD) and Reduced Function Devices (RFD).

The FFDs encompass all the capabilities such as routing, association and formation of a network. The PAN coordinator is an FFD that acts as the main controller to which other devices may be associated. It is responsible for the time synchronization of the entire network. Any FFD can also act as a Coordinator providing local synchronization services and routing to its neighbours. Every coordinator must be associated to a PAN Coordinator and will form its own network if it does not find one in its vicinity.

The Reduced Function Device (RFD) is typically the end node of an IEEE 802.15.4 network. A RFD is intended for applications that are extremely simple, such as actuators or sensing autonomous devices, typically synced with a coordinator and not capable of routing functionalities [5].

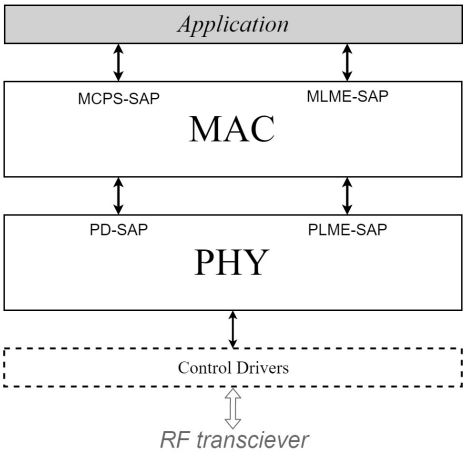


Figure 2.2: LR-WPAN layered architecture

A LR-WPAN device requires a RF transceiver and the respective control drivers for interaction with the PHY layer, along with a MAC sublayer that

provides controlled access to the wireless medium via a layered architecture, such as presented in the block diagram of 2.2.

An intermediate network layer (NWK) may allow support for a network layer protocol, which provides network configuration, manipulation and a message routing, else, the application may have direct access to the MAC sublayer. The SAP (Service access provider) derivatives, function as interfaces to the respective upper layers, allowing for both standardized control and data transfer.

On a IEEE 802.15.4 compliant network stack, the packet transfer between both layers and devices, follows a encapsulating mechanism, where, depending if "travelling" down or up the stack, the layer transporting the data unit, will respectively add or remove the corresponding encapsulation, with elements like headers and footers, designed to support each layer's tasks. This is accomplished through the data frame structures, defined in the standard and represented in Figure 2.3, designed with a minimum complexity while maintaining essential elements for the protocol operations.

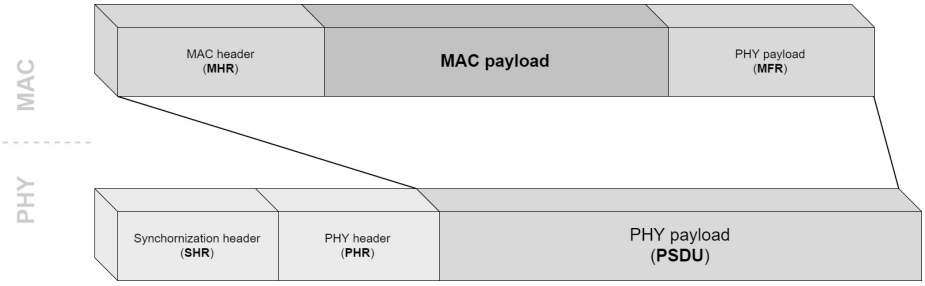


Figure 2.3: 802.15.4 generic data unit encapsulation

A MAC frame, the MAC protocol data unit (MPDU), carrying the application data, is passed to the PHY as the PHY service data unit (PSDU) and encapsulated, forming the PHY protocol data unit (PPDU), as seen in Figure 2.3.

In what follows, we provide a description of the most prominent features of each layer.

2.2.1 PHY

The required features from the physical layer are the activation and deactivation control over the radio transceiver, measuring link quality through the Link Quality Indicator (LQI), energy detection (ED), receiving and transmitting packets across the medium, perform a clear channel assessment (CCA) and channel selection (). As represented in Figure 2.4, the IEEE 802.15.4 operates across three frequency bands: 2.4 GHz (with 16 channels), 915 MHz (with 16 channels) and 868 MHz (single channel). The data rate also varies depending on the used bands.

The 2.4 GHz band operates with a data rate of 250 kbps, the 915 MHz and the 868 MHz bands operate at 40 and 20 kbps, respectively.[6]

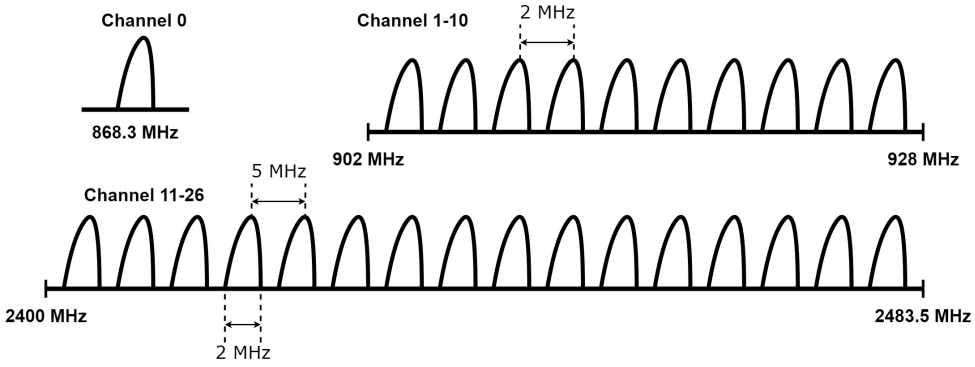


Figure 2.4: Physical layer reference model

The PHY provides an interface between the MAC sublayer and the physical wireless channel, via the RF firmware and the RF hardware. As represented in Figure 2.5, its design model includes a physical layer management entity (PLME) that is responsible for maintaining a database of managed objects pertaining to the PHY referred to as PHY PAN information base (PIB) [7].

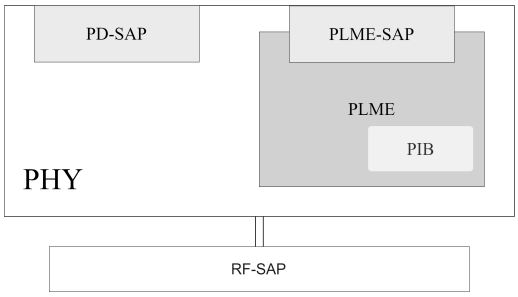


Figure 2.5: Physical layer reference model

The PLME provides management service interfaces through which the PHY data service, accessed via the PHY data service access point (PD-SAP), and the PHY management service, accessed through the PLME-SAP.

However, neither of the interfaces is defined in the standard as they are not expected to be exposed in a typical implementation. Instead the PHY PIB attributes are accessed by the MAC sublayer management entity SAP (MLME-SAP) through MLME-GET and MLME-SET primitives.

2.2.2 MAC

The MAC sublayer is responsible for the beacon processing, channel access, guaranteed time slot (GTS) management, frame validation, acknowledged frame delivery, as well as device association and disassociation from the WSN. In addition,

the MAC sublayer provides support for implementing application-appropriate security mechanisms.

The MAC sublayer, as depicted in Figure 2.6, provides an interface between the next higher layer and the PHY, built around a management entity called the MLME. This manager provides the service interfaces through which layer management may be invoked. The MLME is also responsible for maintaining a database of managed objects pertaining to the MAC sublayer (MAC-PIB).

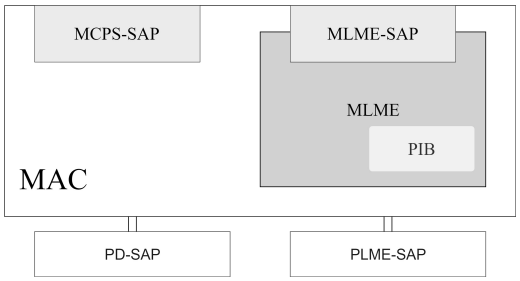


Figure 2.6: MAC sublayer reference model

The MAC sublayer, data and management services, can be respectively accessed through the MAC common part sublayer (MCPS) data SAP (MCPS-SAP) the MLME-SAP. In addition to these external interfaces, an implicit interface also exists between the MLME and the MCPS that allows the MLME to use the MAC data service.

Additionally, "MAC performance metrics" [7] provide feedback to the upper layers on the network performance, in particular, the link quality of the channel, which may help a network layer to take efficient routing decisions, thereby reducing the overall power consumption and latency of the network and the MAC to better assign transmissions into available channels. The feedback information includes: (1) the number of transmitted frames that required one or more retries before acknowledgement, (2) the number of transmitted frames that did not result in an acknowledgement, (3) the number of transmitted frames that were acknowledged properly within the initial data frame transmission and (4) the number of received frames that were discarded due to security concerns.

2.2.3 The Superframe

As defined in the IEEE Std 802.15.4-2015, in order to provide backward compatibility with the previous editions of the standard, the channel access of a LR-WPAN may follow the model of the legacy Superframe wich is delimited by beacons, hence referred as beacon-enabled mode WPAN. Else, the WPAN can instead use a simpler approach based upon Carrier-sense multiple access with collision avoidance (CSMA-CA).

On a beacon-enabled PAN however, the present Superframe is bounded by the transmission of a Beacon frame and, as represented in Figure 2.7, will have an active and an optional inactive portion on which a device may enter a sleep (low power) mode during the inactive period for better energy efficiency.

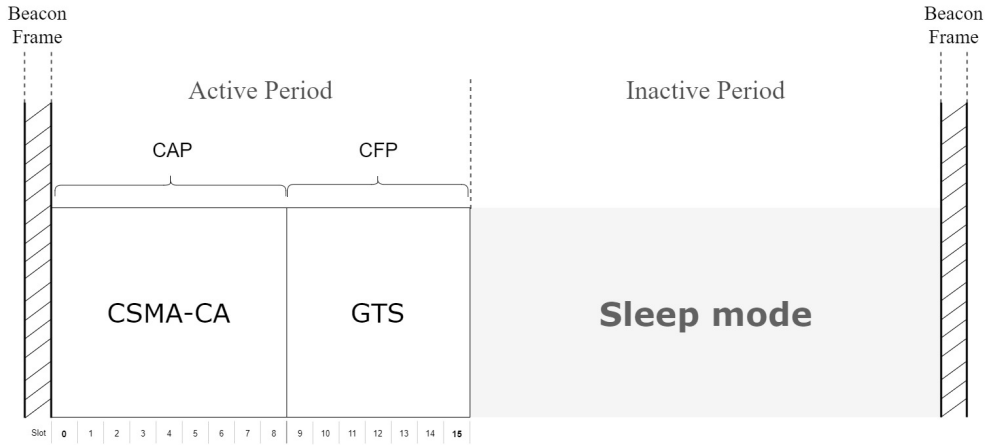


Figure 2.7: IEEE 802.15.4 legacy Superframe Structure example

The active period is divided into the Contention Access Period (CAP) and the Contention Free Period (CFP). During the CAP, the nodes in the network contend to access the channel using slotted CSMA-CA. Whereas, the CFP is composed by Guaranteed Time Slots (GTS), which are used by nodes that require guaranteed bandwidth, resulting on maximum reliability and bounded latency [5].

The structure of the Superframe can be designed to better suit the application and network context and is constructed based on the Beacon Interval (BI), which is the time between two consecutive beacon frames, (2) the Superframe Duration (SD), defining the active portion of the BI, being divided into 16 equal-sized time slots, during which frame transmissions are allowed. The inactive period can be defined, if BI > SD. The BI and SD are determined by two parameters, the Beacon Order (BO) and the Superframe Order (SO), respectively, as defined by the standard and presented in Equation 2.1:

$$\left.\begin{aligned} BI &= aBaseSuperframeDuration * 2^{BO} Symbols \\ SD &= aBaseSuperframeDuration * 2^{SO} Symbols \end{aligned}\right\} \text{for} \quad 0 \leq SO \leq BO \leq 14 \tag{2.1}$$

The *aBaseSuperframeDuration* is defined by the Standard [7] in Table 8-80, as the $aBaseSlotDuration \times numberofslots$, whereas the *aBaseSlotDuration* it

self, is defined as 60 *Symbols*. Some of the timing parameters in definition of the MAC are in units of PHY *symbols*, which is a unit of time based on the transmission duration of 4 bits. The base time varies with the *bitrate*, making the unit universal for all bandwidths. Assuming 250 kbps in the 2.4 GHz frequency band, a *symbol* represents 16 μs [5][10][12].

The beacon interval is delimited by beacon frames, which are standard defined MPDU frames on [7] 7.3.1, that besides "drawing" the structure of the Superframe, also serve as a syncing mechanism for the PAN elements, essential to meet the QoS requirements of determinism and low-latency. On a beacon-enabled PAN, a coordinator, that is not the PAN coordinator, shall maintain the timing of both the Superframe of a received beacon (the incoming Superframe) and the Superframe in which it transmits its own beacon (the outgoing Superframe), ensuring a time synchronism throughout the network. The standard brought an update to the beacon with the Enhanced Beacon (EB) frame format, represented in Figure 2.8, which may now contain additional information elements (IE) fields, used to transfer important management data to the devices.

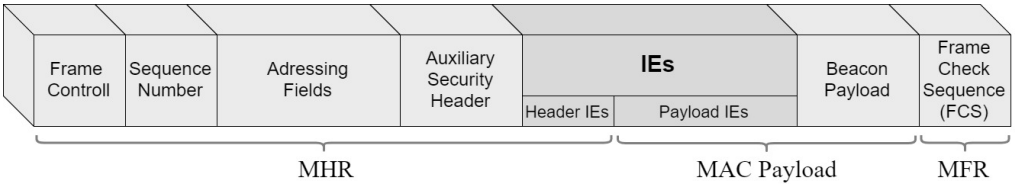


Figure 2.8: Enhanced Beacon MPDU frame format

When present, the IE field length is variable, comprised of a Header and Payload sub-fields containing one or more IE. Apart from the header and management layer based Information Elements, unique IE can support an extended variety of Superframe options.

Different approaches to the Superframe structure and the 802.15.4 protocol are categorized as MAC behaviours. From the 802.15.4e, five different MAC behaviours were added, two being non time-critical namely the Radio Frequency Identification (RFID) and Asynchronous Multi Channel Adaptation (AMCA).

RFID is commonly used for location tracking and "item and people" identification. Using a Blink mode that allows the device to communicate its ID while using a multi-purpose "minimal" frame that consists only of the header fields necessary for their operation. This helps in reducing the overall power consumption in the network.

On a non-beacon enabled PAN, the **AMCA** behaviour can be used when a single channel approach does not have the capability to handle densely populated networks, where channel quality may decrease. So, the AMCA behaviour

makes use of the new Multichannel Access functionality,the device selects a *MAC Designated Channel* based on the channel link quality.

In contrast to AMCA and RFID, the three other MAC behaviours are designed for time critical applications which provide deterministic guarantees and improved robustness.

The Low Latency Deterministic Network (**LLDN**) uses TDMA to provide timing guarantees and is specifically devised for industrial applications requiring low latency such as those found in manufacturing, robotics and other applications that demand robustness because of the critical nature of the data [13]. LLDN is a star topology exclusive MAC behaviour making it suitable for applications that demand a centralized control, allowing for small round-trip time and communication that has to be carried out in a periodic basis.

The Deterministic and Synchronous Multichannel Extension (DSME) and Time Slotted Channel Hopping (TSCH) MAC behaviours, besides providing deterministic communication, are also designed to support multi-channel frequency hopping mechanism.

TSCH provides very high reliability and time critical assurances. In order to achieve collision-free transmissions, seeking reliability, TSCH uses TDMA based *slotframes*, depicted in Figure 2.9, that replaces the traditional IEEE 802.15.4 superframe concept.

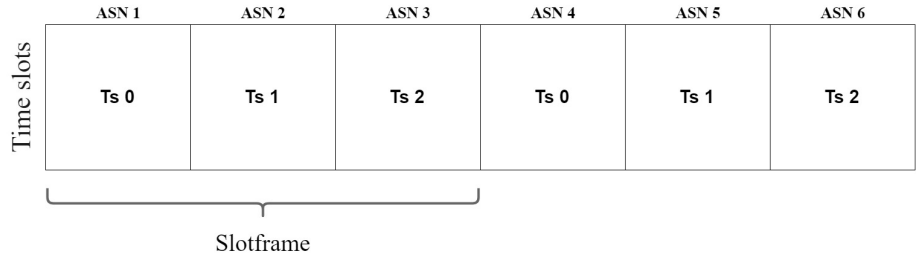


Figure 2.9: TSCH slotframes

Every slotframe is a collection of timeslots, where each has an Absolute Slot Number (ANS) that increases globally and is used to compute the channel in which a pair of nodes communicate, each accommodating a transmission and an eventual acknowledgement. The slotframe size is defined by the number of timeslots in the slotframe that is repeated in cyclic periods, forming a communication schedule.

Communication in each timeslot can be either contention (i.e., using CSMA-CA) or non contention based. Time-slotted communication greatly reduces the unwanted collisions that would otherwise compromise reliability. TSCH can therefore accommodate a dense network, and at the same time, maintain stringent time constraints using fixed length timeslots and multichannel access.

TSCH, along with DSME, supports the Frequency Hopping mechanism, represented in Figure 2.10, which improves the reliability of the network by effectively mitigating the effects of interference at a considerable scale.

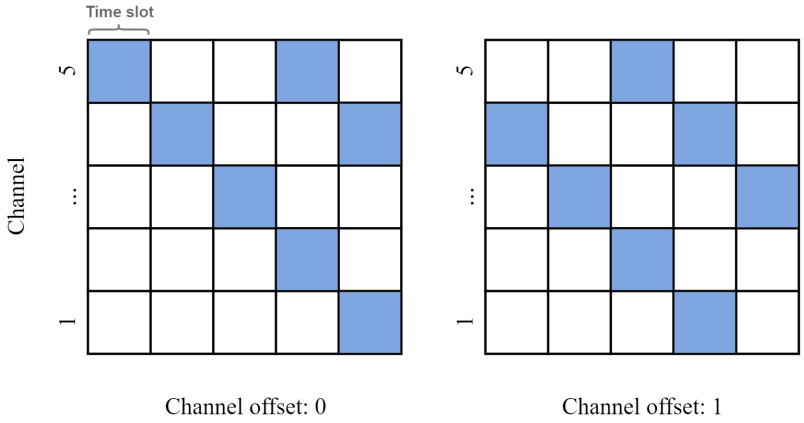


Figure 2.10: Channel hopping example

Channel hopping, used in radio communication systems for decades, is a methodology by which, several devices hop over different channels in a predefined channel order. TSCH can utilize 16 channels which are defined by a channel offset. The link between two nodes is defined by [n, channel offset], a pairwise assignment where the two nodes communicate and their respective offset. However, unlike DSME which has channel diversity mechanisms, the multichannel communication of TSCH completely relies on channel hopping.

In what follows we focus on the DSME MAC behaviour in more detail, given its importance in the context of this Thesis.

2.3 DSME - A time critical MAC behaviour

The Deterministic Synchronous Multichannel Extension MAC behaviour targets applications with QoS requirements such as deterministic latency, high reliability and scalability. To fulfil these requirements, DSME provides several enhancing features to the native IEEE 802.15.4, namely: (1) multi-superframe; (2) CAP reduction; (3) Group Acknowledgement; (4) distributed beacon scheduling and (5) Channel diversity modes.

2.3.1 Multi-superframe

Devices in a DSME-enabled PAN are time synchronized and transmit frames based on specific time structure called *multi-superframe* structure. The PAN coordinator of a DSME network defines a cycle of repeated superframes that will form the multi-superframe, as represented in the structure in Figure 2.11.

Similar to IEEE 802.15.4, a superframe in the multi-superframe structure will have a Contention Access Period and a Contention Free Period. However, in a DSME-enabled PAN, the beacon interval consists only of multi-superframes with no inactive period.

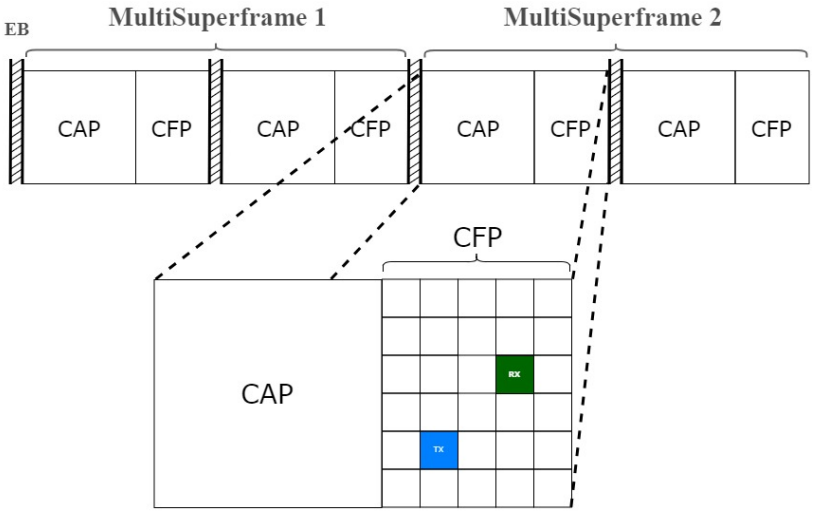


Figure 2.11: DSME Multi-superframe example

In a Multi-superframe, a single common channel is utilized for a successful association, transmitting the EB frames and any frames transmitted during the CAP. The number of superframes that a Multi-superframe accommodates is determined by the PAN coordinator based on the number of data packets meant to be transmitted within the time interval, and is conveyed to the nodes through an Enhanced Beacon (EB).

During the CFP, any available GTS in a DSME-enabled PAN can be allocated by a pair of node devices located within transmission distance to exchange a data frame and a acknowledge frame. Thus, the network is not limited to an hierarchical network topology, but can be also follow a mesh grid network topology. This feature substantially decreases end-to-end latency in multi-hop environments, since the number of hops can be minimized by selecting proper neighbouring devices and scheduling DSME-GTS. Besides periodic data traffic, urgent data or non-periodic data can be sent during the CAP [11].

The standard defines the structure of the Beacon Interval, following the legacy IEEE 802.15.4 superframe, by the values of SO and BO, but now adding the new variable introduced in DSME, the multi-superframe order (MO), integrated on the equations in 2.2. Through MO the Multi-superframe Duration (MD) is

determined.

$$\left.\begin{aligned} SD &= aBaseSuperframeDuration * 2^{SO} Symbols \\ MD &= aBaseSuperframeDuration * 2^{MO} Symbols \\ BI &= aBaseSuperframeDuration * 2^{BO} Symbols \end{aligned}\right\} \text{for} \quad 0 \leq SO \leq MO \leq BO \leq 14 \tag{2.2}$$

As seen in Figure 2.12, DSME enhances the functionality of the traditional GTS by extending its number using the multi-superframe's multi-channel communication. This enables the protocol to select better channels based on link quality and to accommodate higher number of transmissions, increasing the overall reliability and scalability of the network.

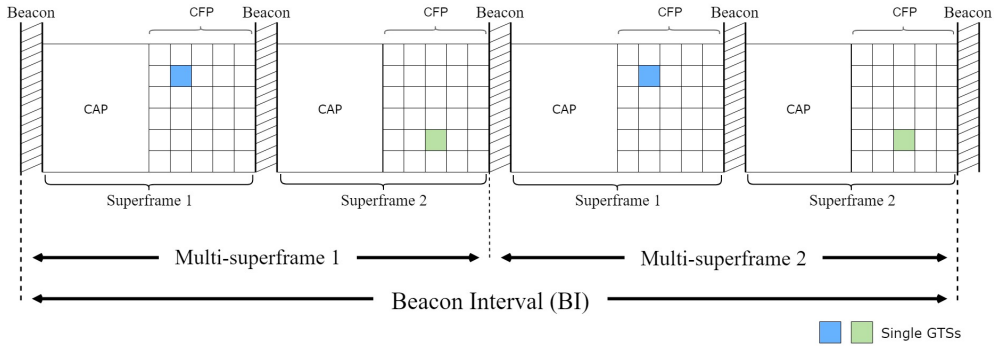


Figure 2.12: DSME Beacon interval structure for BO=5, MO=4 and SO=3

The horizontal axis of the grid represents the time, and the vertical axis of the grid represents the frequency. This means that several GTS can be allocated at a same time but only for different channels.

2.3.2 Beacon Scheduling

In order to build more complex networks topologies such as mesh, it is mandatory to carry out efficient beacon scheduling to avoid interference and collisions. In a DSME network, all devices are time-synchronized, using the values of the *Timestamp* field of the received beacons from the device they are associated with, thus maintaining global time synchronization in the PAN.

When a node wants to join a network, it uses the *MLME SCAN.request* primitive to initiate scanning over all the available channels in the network. During this scanning process, the joining node searches for all coordinators transmitting Enhanced Beacon frames. The neighboring devices send their beacon schedule information to the new joining device by transmitting an Enhanced Beacon. This

beacon schedule is updated as a bitmap sequence to the CFP. The new joining device searches for a vacant beacon slot (GTS), and if available, will claim it for sending its own beacons.

2.3.3 Fast Association

When a device wishes to join a network, *Fast Association* is an option for every MAC behaviour of IEEE 802.15.4e. To carry out a fast association, the higher layer of the device posts a "*MLME ASSOCIATE.request*" primitive, triggering the procedure in the MAC sublayer. The request is sent to the PAN coordinator which acknowledges its reception. Fast Association removes the wait time duration "macResponseWaitTime", which efficiently reduces association delay. The request command contains an acknowledgement request, which the coordinator confirms by sending the acknowledgement frame. If the coordinator has sufficient resources, the higher layer allocates a 16 bit address to the device. The MAC sublayer then generates a status indicating a successful *Fast Association*. The device can then use the allocated "macShortAddress" for its association within the PAN[5].

2.3.4 CAP reduction

It is possible for the PAN Coordinator to reduce the size of the CAP by only enabling it during the first superframe of a multi-superframe. This technique is denominated CAP reduction. This way, the remaining superframes only present a longer CFP (Figure 2.13). It radically increases the number of DSME GTS that are allocated to the neighbouring nodes, while saving energy, since there is no need for a node to stay active during a CAP if no transmissions are expected to occur. The mechanism is flexible, and can be configured along with the parameters in Equation 2.2, to better accommodate a network and its needs.

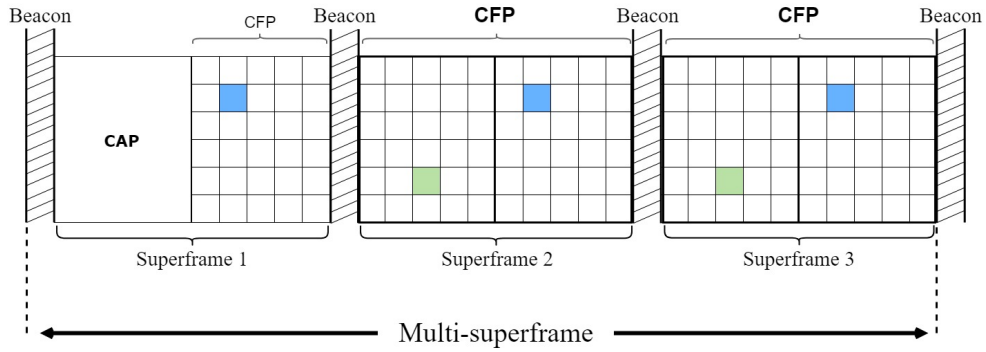


Figure 2.13: CAP Reduction technique in DSME

With CAP reduction, similarly to TSCH, DSME becomes suitable for highly

dense networks with stringent QoS requirements in terms of delay and reliability, whilst providing a higher throughput due to the removal of the CAP.

2.3.5 Group Acknowledgement

Another important functionality of the DSME GTS is its Group Acknowledgement (GACK) feature. This mechanism provides the capability of sending a single acknowledgement for all guaranteed transmissions within the same multi-superframe. The GACK reduces the latency and energy consumption by combining several acknowledgements into a single group acknowledgement. In case of a failed transmission, a new DSME GTS will be assigned to carry out the process.

2.3.6 Channel Diversity

When RF devices operate on the same RF spectrum, sharing a network, interference and failed transmissions can occur, thus affecting the overall reliability of the network. The Channel diversity feature helps overcoming this issue. DSME MAC protocol provides two types of channel diversity mechanisms: (1) Channel Hopping, depicted in Figure 2.10, similar to TSCH behaviour, but also (2) Channel Adaptation.

In channel adaptation, the PAN coordinator has the capability to allocate the DSME guaranteed timeslots either in a single channel or through different channels to an end device. This decision depends on the link quality of the current channel. The link quality of the channel is conveyed to the PAN coordinator through the MAC performance metrics. The PAN coordinator is also responsible for deallocating a specific DSME GTS if the link quality of an allocated DSME GTS becomes degraded.

2.4 Concluding Remarks

This chapter, presents an overview of the IEEE std. 802.15.4-2015, focusing on the features of the standard, but leaving aside the defining instructions and rules. This does not mean, however, that they are of any less relevance. Removing these elements would defeat the purpose of a canon, whose main goal is to standardize the technology for both industry and scientific community, promoting a joint effort on the technology development and use. A goal of accessibility present within this Thesis.

Chapter 3

Tools and Technologies

This chapter presents and describes the technologies used throughout this Thesis. It starts by describing the the TelosB hardware platform, including an evaluation that lead to selecting the device. Then, the Daintree Sniffer and MSPSIM debugging tools are presented before the FreeRTOS analysis as the chosen RTOS.

3.1 TelosB Mote Platform

The implementation target is the TelosB [14] hardware platform (Figure 3.1), which provides the following features:

- TI MSP430F1611 16-bit microcontroller[15];
- CC2420 2.4 GHz RF transceiver, IEEE 802.15.4 Compliant[18];
- 48 kB of Program memory (in-system reprogrammable flash);
- 10 kB of RAM;
- 32 kHz internal crystal;
- Low power consumption;
- Temperature, humidity, visible light and IR sensors;
- UART communication port (USB converter).

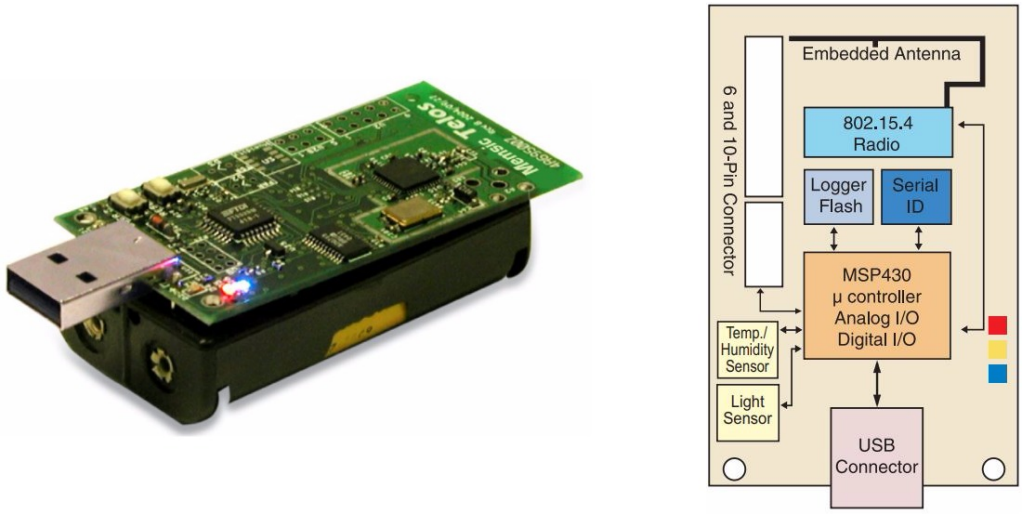


Figure 3.1: TelosB mote and block diagram [14]

3.1.1 Choosing TelosB

The TelosB mote, unlike most WSN motes such as the MicaZ [19], does not need any programmer interface, already having an USB port with drivers that can be used for flashing applications as well as interfacing the mote with other equipment, particularly useful for debugging applications. This facilitates an implementation of the IEEE std. 802.15.4-2015. Although, TelosB will provide challenges, both memory and time wise. With an internal crystal at 32 kHz, capable of a time granularity of about two *symbols* at a 250 kb /s, the platform will require a precise and time efficient implementation of the protocol. Also, as it is an already 15 year old hardware mote, it is less flexible than newer devices when it comes to memory, specially when the running application restricted to 10 kB of random access memory (RAM). However, TelosB is a widespread platform among the IoT community [35][36][37], and being able to implement the IEEE 802.15.4 network stack on TelosB will prove itself a remarkable achievement, with an optimized implementation capable of being ported onto newer WSN devices.

In what follows, the hardware components that constitute the TelosB platform and are relevant for the context of this Thesis, are presented in a better detail.

3.1.2 MSP430F1611

The low power operation of the TelosB module is due to the ultra low power Texas Instruments MSP430F1611 microcontroller featuring 10 kB of RAM, 48 kB of flash, and 128 B of information storage. This 16-bit processor features extremely low active and sleep current consumption that allows TelosB to run for years on a single pair of AA batteries.

The MSP430 has an internal digitally controlled oscillator (DCO) that may operate up to 8 MHz and be turned on from sleep mode in 6 μs. When the DCO is off, the MSP430 operates off a second available clock source referred to as *LFXT1CLK*, running on a 32768Hz watch crystal when on low frequency mode. Although the DCO frequency changes with voltage and temperature, it may be calibrated by using the 32 kHz oscillator.

Additionally, the MSP430 has 8 external ADC ports and 8 internal ADC ports which may be used to read the internal thermistor or monitor the battery voltage.

A variety of peripherals are available including SPI, UART, digital I/O ports, Watchdog timer, and two Timers (A and B) with capture and compare functionalities. The F1611 also includes a 2-port 12-bit DAC module, Supply Voltage Supervisor, and 3-port DMA controller. The features of the MSP430F1611 are presented in detail in the Texas Instruments MSP430x1xx Family User's Guide [16][17].

3.1.3 CHIPCON CC2420

TelosB features the *Chipcon* CC2420 [18] radio for supplying wireless communications. The CC2420 is an IEEE 802.15.4-2006 compliant radio, providing full support for the PHY and MAC functions. With sensitivity exceeding the IEEE 802.15.4-2006 specification and low power operation, the CC2420 provides reliable wireless communication. The CC2420 is highly configurable for many applications with the default radio settings necessary for the IEEE 802.15.4 compliance. The RF transceiver is controlled by the TI MSP430 microcontroller through the serial peripheral interface (SPI) port and a series of digital I/O lines and interrupts (Figure 3.2) and may be shut off by the microcontroller for low power duty cycled operation.

The CC2420 provides extensive hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information, features which reduce the load on the host controller and allow CC2420 to interface low-cost microcontrollers, as is the case of the MSP430F1611.

The configuration interface and transmit/receive First In First Out (FIFO) buffers of CC2420 are accessed via the SPI interface, a 4-wire Serial Configuration and Data Interface [18, chapters 13 and 14], integrated as represented in Figure 3.2.

In order to achieve the best performance for each application, the radio is configurable through a set of programmable configuration registers (listen on [18, chapter 37]), allowing for the following setting tweaks:

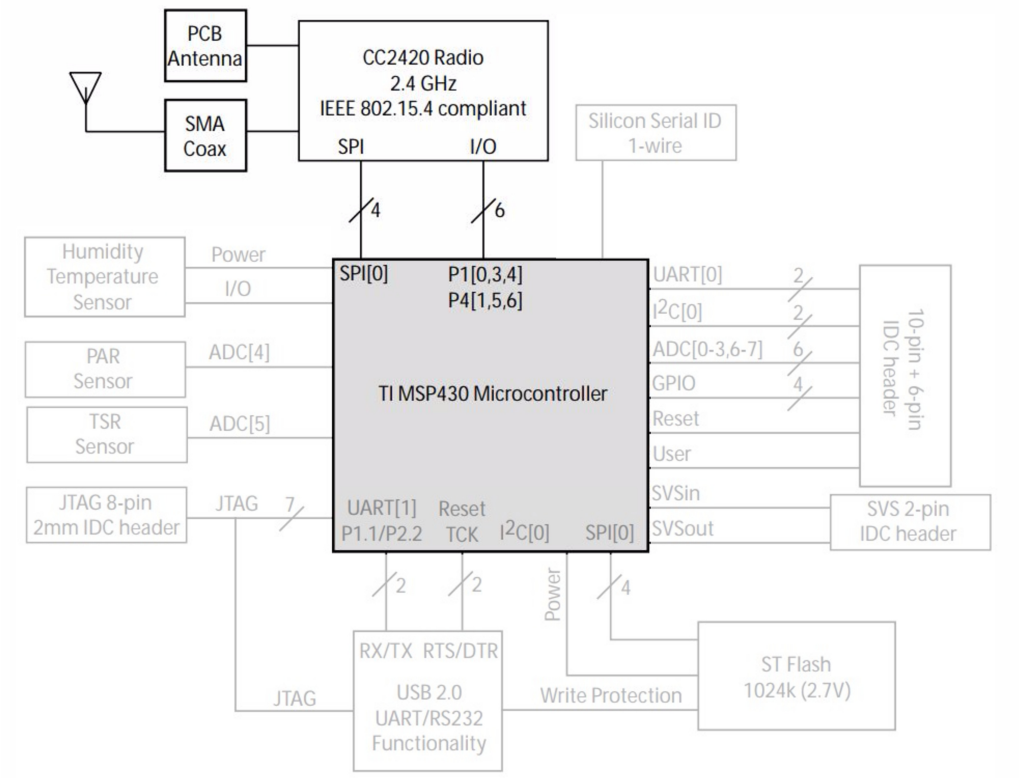


Figure 3.2: Functional Block Diagram of TelosB, MCU-Radio communication [17]

(1) Switching between Receive and Transmit operation mode. In Receive operation mode, the RF device will ”listen” to the physical medium until the chip SFD pin ([18, chapter 7]) goes active, after detecting the start of frame delimiter (SFD) field from the SHR header, staying active only with a successful address recognition, allowing an hardware filter functionality for unwanted source frames. The received frames will be stored on the RXFIFO activating the FIFO pin that remains so until the buffer is empty. The FIFOP pin is active when the number of unread bytes in the RXFIFO exceeds the programmed threshold. When operating under Transmitting mode, the FIFO and FIFOP pins are still only related to the RXFIFO, however the SFD pin will go active when the SFD field has been transmitted, remaining so until the complete MPDU has been sent.

(3) The RF channel options allow selecting between 16 channels (11 to 26), as specified by the IEEE std. 802.15.4-2015 for the 2.4 GHz band, in 5 MHz steps. [18, chapter 26]

(4) The RF output power is programmable with 9 different Output Power [dBm] modes, affecting the overall energy consumption of the device and the residual energy left on the physical medium.

(5) Power-down / power-up mode that allows for controlling the device on/off periods in order to the provide the external microcontroller full energy management capabilities.

(6) The Clear Channel Assessment (CCA) signal, based on the measured Received Signal Strength Indication (RSSI) value and a programmable threshold, allows selecting between 3 modes: CCA when received energy is below threshold; CCA when not receiving valid IEEE 802.15.4 data; CCA when energy is below threshold and not receiving valid IEEE 802.15.4 data [18, chapter 25].

(7) Encryption / Authentication modes for hardware support of the IEEE 802.15.4 MAC security operations.

The CC2420 functionalities are accessed through a built-in state machine (Figure 3.3) that is used to switch between different operational states. The change of state is done either by internal events such as SFD detected in receive mode or by using command strobes, which are single byte instructions sent to the radio via SPI.

In order to turn the radio on, both the voltage regulator, responsible for powering the device, and the crystal oscillator needed for the radio internal clock and other radio operations, must be turned on and become stable. Only then, the radio can be used in either RX or TX mode, as dictated by the state machine, whose active state can be read in the FSMSTATE status register, for test/debug purposes.

CC2420 includes also hardware support for transmitting acknowledge frames, following an IEEE std. 802.15.4 compliant auto-filled frame format, with the timing commanded by the microcontroller by issuing the appropriate command strobes [18, chapter 19]. However, acknowledge frames may be manually transmitted using normal data transmission if desired, future proofing the radio for possible standard updates on the acknowledge frame format.

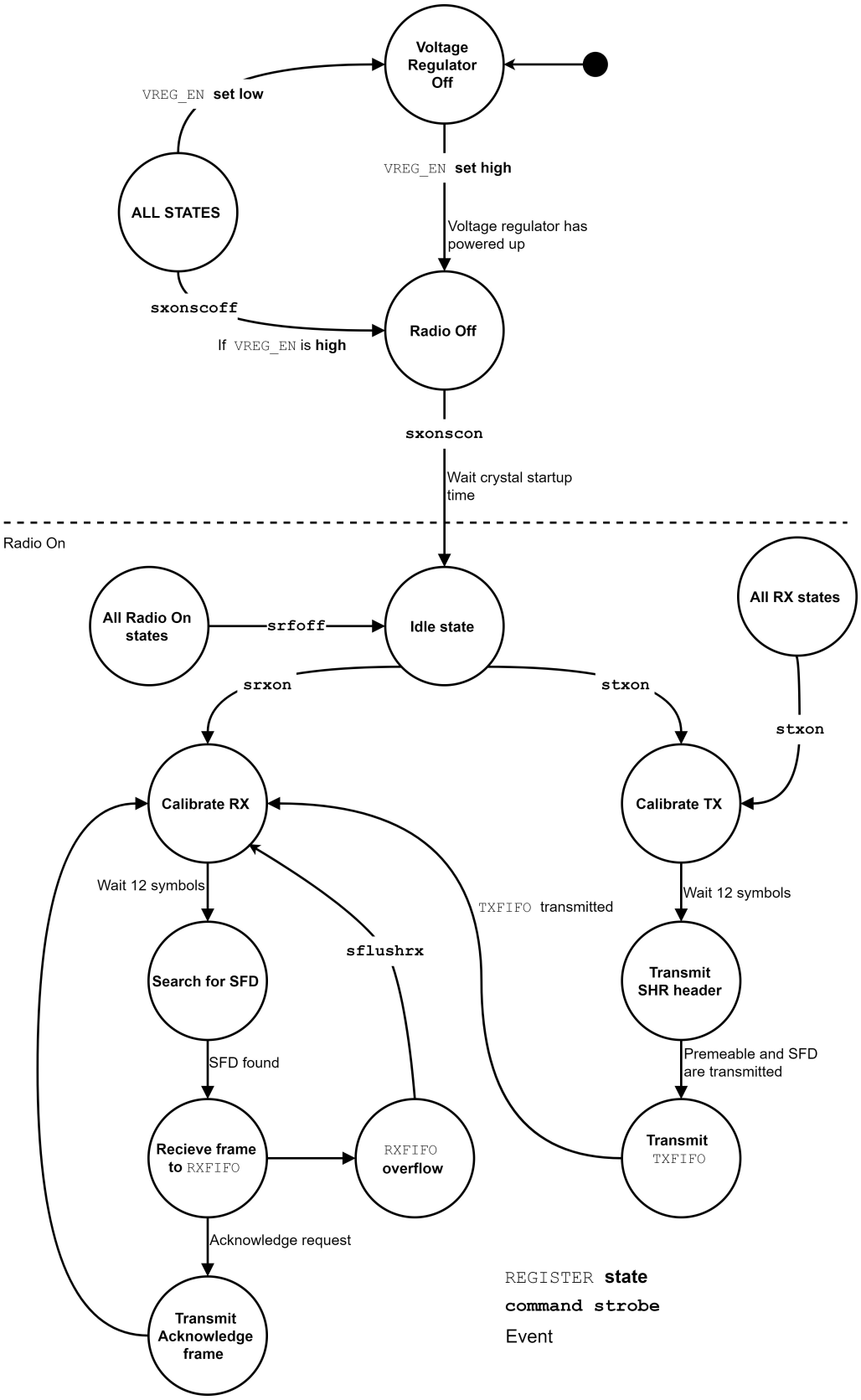


Figure 3.3: Radio control states

3.2 Daintree 2400E Protocol Analyzer

The implementation of IEEE 802.15.4-2011 is supported by the Daintree 2400E packet sniffer. This protocol analyzer, detects and registers any IEEE 802.15.4 transmitting packets, even interpreting frames with versions prior to the 2015 standard, as the sniffer dates before its release. This features allow to debug and to validate the implementation of the IEEE 802.15.4-2015 protocol.

The proprietary Daintree Network Analyzer software, besides providing the received packet list and their field highlighting, also constructs a graphic view of the network topology, including the visualization of routing paths, message flows, device states and link quality of the messages, as depicted in Figure 3.4.

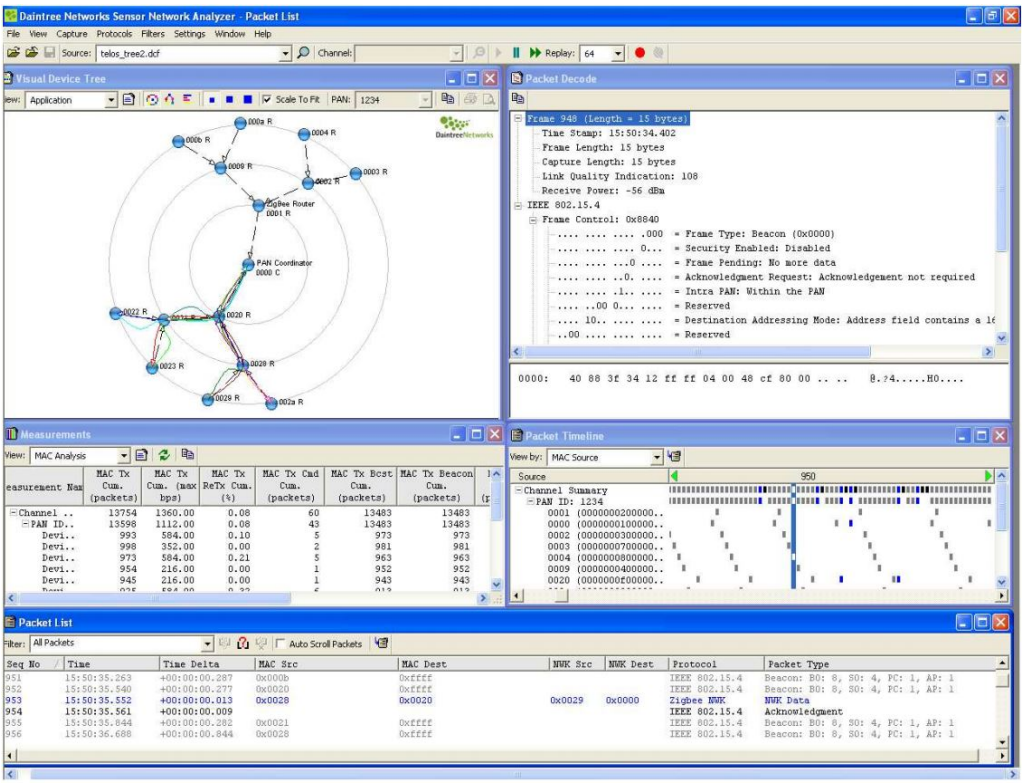


Figure 3.4: Daintree Network Analyzer[38]

The software is also capable of evaluating the network status of the devices by analyzing the messages transmitted, messages received, lost message ratio, bandwidth usage, average link quality indicator among others. Additionally, the Daintree Network Analyzer supports the higher layer ZigBee protocol and the application distinguishes between analysis parameters depending on the selected protocol layers.

The Daintree Analyzer enables the import of a plant layout (office floor, factory floor) and overlay the network topological view over it. This feature allows

dragging and dropping nodes, assigning labels to each node and it can be very useful for monitoring the network. The hardware used in conjunction with this network analyzer is the 2400 Sensor Network Adapter [38]. This adapter includes an Ethernet interface and can be used for a multiple and synchronized node sniffing, meaning that several 2400 can be scattered (connected to an Ethernet network) in a certain geographical area in a way that IEEE 802.15.4/ZigBee traffic can be collected at different locations of a large-scale network into a single application.

3.3 MSP430 simulator

MSPSim is a Java-based instruction level emulator for the MSP430 series microprocessor and emulation of some sensor networking platforms. MSPSim targets cycle accurate emulation of both the MSP430 Central Processing Unit(CPU) core and built-in peripherals such as timers, serial communication and analog to digital converters. Furthermore, MSPSim emulates external components such as the radio chip CC2420, sensors, and flash memories. MSPSim also provides emulation of complete sensor devices such as the Tmote Sky (which has the same board design as TelosB mote) [27] and Scatterweb ESB [26]. The program provides a GUI (Graphical User Interface) with a visual representation of platform (Figure 3.5) that interacts along the emulated instructions, with elements such as sensors, interfaces and Light-emitting Diodes (LED) [28].

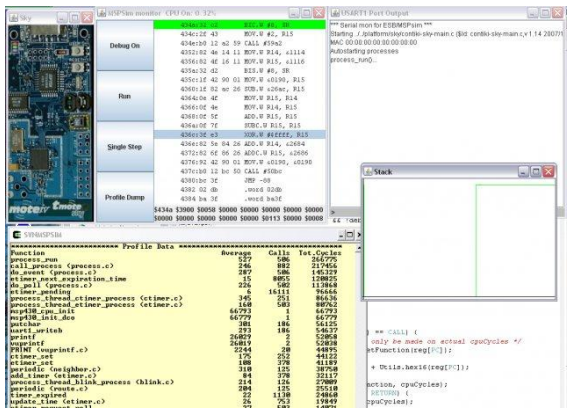


Figure 3.5: Multitaskting on a Real Time OS [39]

The simulator supports IHEX and ELF firmware files, and has some tools for monitoring the stack, setting breakpoints, and profiling [39], making it a powerful tool for debugging applications without the need to acquire the MSP430 Flash Emulation Tool, an expensive MCU Programmer and a direct hardware Debugger.

The usage of an emulator during implementation, that provides an hardware abstraction layer and debugging tools for the application

3.4 FreeRTOS Operating System

3.4.1 Real-time Operation systems

An Operating System (OS) is a computer program that supports a computer's basic functions, and provides services to other programs that run on the computer. Some OS allow multiple programs to execute at the same time. This is called multi-tasking. In reality, each processor core can only be running a single thread of execution at any given point in time and the type of OS is classified by how the processor operation time is distributed between the running programs. The OS has a **scheduler**, responsible for deciding how the processing time is distributed and what task (one of the applications "running") should be executing. For example, the scheduler used in a multi user operating system, such as Unix, will ensure each user gets a fair amount of the processing time whilst the scheduler in a system such as Windows, will try and ensure the computer remains responsive to its user. [3]

In addition to being suspended involuntarily by the kernel, a task can choose to suspend itself. It will do this if it either wants to delay (sleep) for a fixed period, or wait (block) for a resource to become available (e.g., a serial port) or an event to occur (e.g., a key press). A blocked or sleeping task is not able to execute, and will not be allocated any processing time.

In a Real Time Operating System (RTOS), the scheduler is designed to provide a predictable execution pattern, a deterministic behaviour, meeting real time requirements by some applications, common among embedded systems[4]. A real time requirement is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline), only possible with the deterministic scheduler provided by the RTOS.

3.4.2 Choosing FreeRTOS as the IoT OS

Before choosing FreeRTOS as the operating system for the IEEE 802.15.4, a survey was performed, gathering other popular OS among the IoT community. The most critical characteristics under evaluation were (1) Scheduling and Real-time capabilities, (2) Memory footprint, (3) Energy efficiency, (4) Hardware support and (5) programming language, but also, taking into account available documentation and robustness of the OS.

Table 3.1 exposes some of these features to take in consideration for the implementation task.

Table 3.1: A Comparison between OS

OS	Real-time	Schedule	language	Hardware
FreeRTOS	Supported	Preemptive/ Cooperative	C	AVR, MSP430, ARM, x86, 8052, Renesasc
RiotOS	Supported	Preemptive	C	AVR, MSP430, ARM7, x86, ARM Cortex-M
Contiki	No support	Cooperative	C	MSP430, ARM7, x86, ARM Cortex-M, PIC32, 6502, AVR
Tiny OS	No support	Cooperative	NesC	AVR, MSP430, px27ax

In regards to energy efficiency, due to being a recurrent requirement in the IoT world, all the OS are energy efficient and provide most of the same energy management library for the respective hardware [24].

Memory wise, since IoT devices have less resources than conventional connected objects, IoT OS are designed seeking a low memory footprint and need to provide mechanisms in order to abstract the memory management for the programmer. The Internet Engineering Task Force (IETF) standardized a classification devices in three subcategories based on memory capacity [25].

- Class 0 devices have the smallest resources ($\ll$ 10 kB of RAM and $\ll$ 100 kB Flash), e.g., a specialized mote in a wireless sensor network (WSN);
- Class 1 devices have medium-level resources ($\sim$ 10 kB of RAM and $\sim$ 100 kB Flash);
- Class 2 devices have more resources, but are still very constrained compared to high-end IoT devices and traditional Internet hosts.

As for the performance of the compared OS, based on memories needs, FreeRTOS and RIOT are both for devices under class memories 1 and 2. Contiki is for devices classified under classes 1 and 0 and the smallest memory footprint belongs to the TinyOS whose devices fit under class 0 [24]. TelosB, given its aforementioned characteristics, can be classified under class 1.

Only RiotOS and FreeRTOS offer real time capabilities which is a crucial feature for the task in hand, proving the best candidates for the standard implementation. Both support the TI MSP430 and a handful of other IoT hardware, important for the project longevity, and the OS are comparable by the critical characteristics standards mentioned above.

An attempt to implement the protocol grounds on top of RiotOS was made during the early days of the project. However, due to lack of documentation

and coding standards, FreeRTOS was found to be the appropriate OS for the task. The strict quality management of the OS project, better documentation than all the other surveyed OSs, the minimalism and simplicity of the system as well as the robustness, all of this in par with the documented design goals make FreeRTOS look like the perfect partner for the task at hand.

3.4.3 FreeRTOS

FreeRTOS is a real time OS with a compact format, designed with the primary goals of ease of use, small memory and energy footprints, and robustness. The OS targets embedded applications for implementation on microcontrollers, with such size constraints that a "full" RTOS implementation is not possible.

The FreeRTOS kernel was originally created by Richard Barry in 2003, and was later developed and maintained by Richard's company, Real Time Engineers. FreeRTOS was a success, and in 2017 Real Time Engineers, Ltd. passed stewardship of the FreeRTOS project to Amazon Web Services [20].

FreeRTOS provides a priority based scheduling procedure. The OS is well suitable to a small embedded system that has limited, predefined tasks and research should be done on designing FreeRTOS to handle large IoT system tasks with more advanced scheduling mechanisms.

The OS relies on a tick which is a function periodically called, with an interruption like behaviour, and responsible for maintaining several of the OS base functionalities. The main purpose of the tick is to support the scheduler, by driving the context switch between tasks and allowing it to distribute CPU processing time.

The OS selling features are highlighted as the following:

- 6 kB to 12 kB ROM footprint;
- Pre-emptive scheduling option;
- Co-operative scheduling option;
- Round robin with time slicing;
- Mutexes with priority inheritance;
- Easy to use message passing;
- Very efficient software timers;
- Configurable / scalable.

The most relevant characteristics of the OS for this Thesis are described below.

3.4.4 Tasks

In FreeRTOS, tasks can be created and contain a functional component that the user wants to run at specific times or in reaction to other events. When the OS is running, each task exists in one of four states: (1) Running when utilizing processing time, if the processor on which the RTOS is running only has a single core then there can only be one task in the Running state at any given time.(2) Ready whenever able to execute but are not currently executing because a different task of equal or higher priority is already in the Running state. (3) Blocked if in a "sleep" state waiting for any event, i.e. timer interrupt or a semaphore resource to become available, and when in the blocked state, tasks do not use any processing time but cannot be selected to enter the Running state. (4) Suspended tasks, like Blocked ones, cannot be selected to enter the Running state, but when in the Suspended state do not have a time out. Instead, tasks only enter or exit the Suspended state when explicitly commanded. The interaction between the application and tasks state is made through a set of modules [22], that result on a state machine for each task, following the module of Figure 3.6.

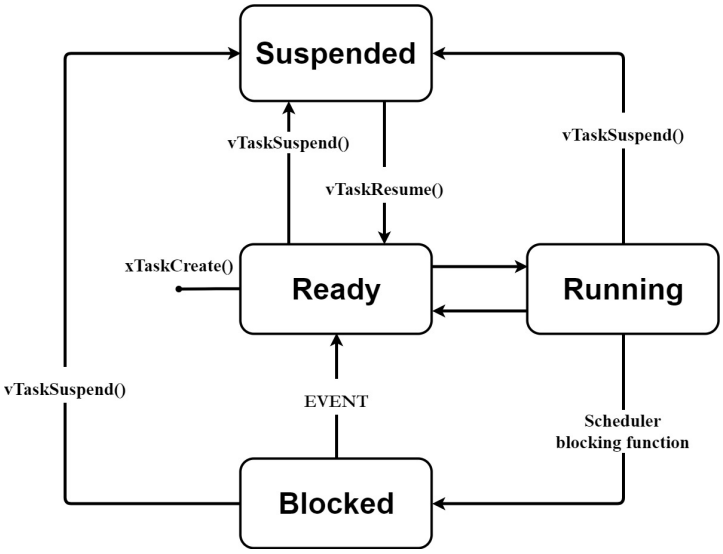


Figure 3.6: Task state transitions

Each task, in FreeRTOS, is assigned a priority from 0 to a configurable $MaxPriorities - 1$ where, low priority numbers denote low priority tasks. The FreeRTOS scheduler ensures that the higher priority tasks will be given processor (CPU) time in preference to tasks with lower priority that are also in the ready state. In other words, the task placed into the Running state is always the highest priority task that is able to run.

An idle task is created automatically when the RTOS scheduler is started to ensure there is always at least one task that is able to run. It is created at

the lowest possible priority (typically 0) to ensure it does not use any CPU time if there are higher priority application tasks in the ready state. This way, it is possible to reduce the power consumed by the microcontroller running FreeRTOS by using the Idle task hook to place the microcontroller into a low power state. The power saving that can be achieved by this simple method is limited by the necessity to periodically exit and then re-enter the low power state to process tick interrupts. Further, if the frequency of the tick interrupt is too high, the energy and time consumed entering and then exiting a low power state for every tick will outweigh any potential power saving gains for all but the lightest power saving modes.

The FreeRTOS tickless idle mode stops the periodic tick interrupt during idle periods (periods when there are no application tasks that are able to execute), then makes a correcting adjustment to the RTOS tick count value when the tick interrupt is restarted. Stopping the tick interrupt allows the microcontroller to remain in a deep power saving state until either an interrupt occurs, or it is time for the RTOS kernel to transition a task into the Ready state.

3.4.5 The scheduler

The scheduler used in FreeRTOS achieves determinism by allowing the user to assign a priority to each task of execution and then chooses, based the priority, which thread of execution to run next. The scheduler follows a preemptive priority method where any running task can be suspended in order to an assigned higher priority one, use CPU processing time instead. This ensures that the application is able to perform any critical task instantly (*t1*, Figure 3.7) and that it has full control of the CPU time distribution, therefore creating a predictable behaviour, or in other words, a real time system.

Figure 3.7 is an example of how the FreeRTOS scheduler handles multitasking. The preemptive behaviour is first represented in *t1*, when the task A is launched by the application, it immediately starts running, interrupting the lower priority task C.

If multiple tasks have the same priority, as is the case of tasks A and B , and are in the Running or Ready state, the kernel creates a Round Robin model, visible between *t2* and *t3*, where each gets a full tick before switching to the next.

Whenever an top priority task ends, the scheduler instantly starts running the next ready higher priority task (*t4*), or, if there are no tasks left to run, the idle task will start running (*t5*) for energy efficiency purposes, until the application launches any task which will immediately start running (*t6*).

FreeRTOS also supports an alternative cooperative scheduler by using Co-routines instead of tasks, which are a different type of process that may be used for a real time application. The main difference from tasks, is that co-routines

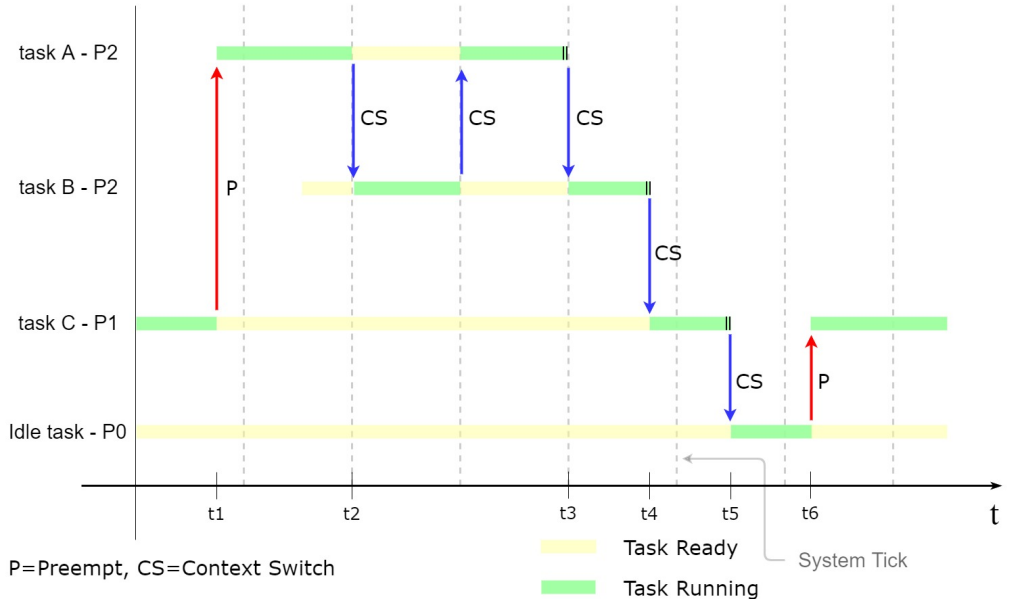


Figure 3.7: Multitaskting on a Real Time OS

use prioritized cooperative scheduling, which means they decide when another co-routine may be executed (task yielding).

3.4.6 Intertask communication

Intertask communication is essential for developing multi-tasked applications. An application will rely on a networking task, a concurrent (parallel) task, when wanting to access a network of devices. This requires some form of communication for both data exchange and resource sharing. FreeRTOS provides the following three mechanisms to fulfil these needs.

Queues are the primary form of intertask communications. They can be used to send messages between tasks or between interrupts and tasks. They use safe FIFO buffers with new data being sent to the back of the queue, although data can also be sent to the front for critical reasons. The items placed in the queues are of fixed size, defined with the maximum number of elements when the queue is created. Tasks can block on queues, waiting for an item to be inserted, or space to be available. It is one way to synchronize different tasks between them, or with interrupts.

Semaphores are directly derived from queues, and contain only one element. The queue can be full or empty and what's contained in the queue does not matter. A task can 'take' a semaphore, that is emptying the queue. If the queue was not full, it will block. An interrupt can 'give' the semaphore, that is filling the queue, hence allowing the task to continue. Semaphores are therefore mainly

designed to offer synchronization between tasks and interrupts.

Mutexes (for MUTual EXclusion) are a form of binary semaphores used to prevent several tasks to access the same resource at the same time. A mutex is created for a shared resource and it's associated queue is full. When a task wants to access the resource, it 'takes' the mutex. When the task is done with the resource, it gives the mutex back, so another task can use it. If a task tries to take a mutex that is already taken, it will block the resource becomes available [31].

3.4.7 Memory management

FreeRTOS offers several heap management schemes, five sample memory allocation implementations [23], providing different mechanisms, with the possibility of implementing a custom memory heap algorithm, so that the user can select the necessary memory management functionalities. The Memory footprint of the OS, the official documentation advertises a RAM usage of about 300 b while some surveys point to a less conservative 4KB of usage. As for the ROM footprint the documented usage is between 5kB to 10 kB, backed by the surveys [34][24][25].

Chapter 4

Implementing FreeRTOS over TelosB

This chapter describes in great detail the port of the FreeRTOS to the TelosB platform, focusing on the microcontroller, radio transceiver and other device peripherals. In addition, an experimental validation is carried out and presented with the purpose to corroborate and validate the implementation.

4.1 Porting FreeRTOS to the MSP430F1611 MCU

FreeRTOS doesn't officially offer a native port for the *TelosB* microcontroller, MSP430F1611. However, as indicated in the official supported devices list[30], FreeRTOS already provides a port for MSP430F449, a variant from the same family as the TelosB MCU. Following a porting guide for the WSM430 hardware platform [31], which also uses the same MCU as TelosB, the port specific source files for MSP430F449, located under the 'Source/portable/GCC/MSP430F449' directory from the official FreeRTOS Kernel [33], can be used for compiling FreeRTOS onto the TelosB, after the appropriate adaptations.

On the *port.c* file, resides the adaptation layer between the operating system and the microcontroller. The hardware timer initialization to generate the RTOS tick, macros to save a task context for task switching and other support functions for the OS scheduler are defined under this file. Fortunately, this port is fully compatible with the TelosB MCU, given all the resources utilized by the OS (registers for context saving and **TIMER A** for the RTOS ticker) are available on both MCU and share the same macros by the *port.c* compiler, *MSP430-gcc*[15][32].

When setting up the ticker ISR timer, in:

```
// Set the compare match value according to the wanted tick rate
TACCR0 = portACLK_FREQUENCY_HZ / configTICK_RATE_HZ;
```

The compare value for **TIMER A**, sourced by the low-frequency auxiliary clock (ACLK), is calculated with the project configured tick rate under *FreeRTOSConfig.h*, and the *port.c* defined:

```
// telosb 32kHz crystal nominal value
#define portACLK_FREQUENCY_HZ ( ( TickType_t ) 32768 )
```

Since the TelosB board, provides an internal 32 kHz oscillator which drives the ACLK [16], no changes are required to the *port.c* file.

Special care needs to be taken if utilizing the **TIMER A** when developing applications given that the resource needs to be shared with the OS.

4.1.1 MCU implementation details

Three clock signals are available from the basic clock module of the MCU. The previously mentioned ACLK, driven by the LFXT1CLK, and other two source selectable clocks, between the LFXT1CLK, the DCO and an optional XT2CLK, which is not present on the TelosB platform [16]. First, the Master Clock (MCLK), used by the CPU and system and a second SMCLK Sub-main clock selectable for individual peripheral.

For this Thesis, it is of interest to have an high frequency clock for faster MCU operations but it is required to have a reliable clock source for accurate time stamping. Therefore, the MCLK is driven by the higher frequency DCO, while the ACLK is available for time critical tasks through the LFXT1CLK low frequency mode.

The MSP430 designed start-up procedure (Figure 4.3), starts with disabling the watchdog timer (WDT). The functionality, enabled by default, will reset the device unless the running application periodically resets the countdown timer ("feeding the dog"). This feature is particularly useful for long term running applications. Should a blocking malfunction occur, the watchdog will automatically reset the device seeking to restore the application without requiring a manual intervention. During the implementation of the IEEE 802.15.4 this functionality would not prove so useful, only blocking the progress of the project, therefore is disabled at upstart.

It is then necessary to setup the clock system, configurable through the two Basic Clock System Control Registers (*BCSCTL*). The MCLK and the SCLK are

by default sourced with the DCOLCK, requiring no further configuration. It is necessary however to disable the optional *XT2* oscillator since its not present on the TelosB board.

In order to be able to setup a timer driven by the digital clock, it is necessary to know the frequency value (*fDCO*) the clock is running at. Given that the fDCO is heavily influence by the MCU's temperature knowing its value becomes a challenge, therefore the MSP430 provides a functionality for adjusting the DCO frequency by configuring a set of register values (DCO and RSEL). One solution is to calibrate the DCO using the more predictable 32 kHz crystal.

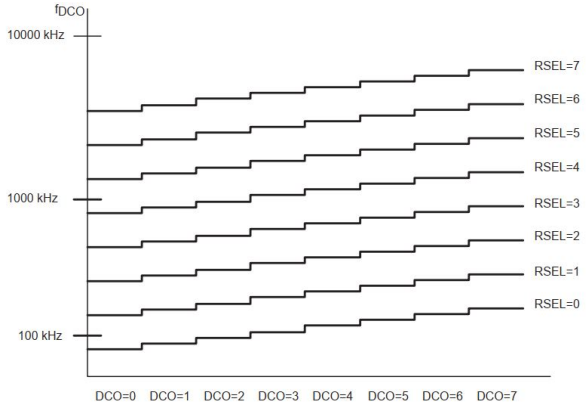


Figure 4.1: Typical DCOx Range and RSELx Steps [16]

Figure 4.1 shows some typical values for the pair of the configurable variables, which represent the DCO frequency behaviour with the adjustment of *DCO* and *RSEL*. Following the indicated nature of the DCO, the algorithm represented on Figure 4.3 performs a clock calibration.

Using a timer sourced by the DCO to capture the input signal of ACLK, the frequency of the DCO can be determined, since the interval between captures of the ACLK corresponds to the steps incremented to the timer counter.

With the configurable reference values as follows:

```
//board.h
#define MSP430_INITIAL_CPU_SPEED     2457600uL
#define F_RC_OSCILLATOR              32768
```

A desirable *"DELTA"* value can be calculated for further comparison, following the Equation 4.1:

$$DELTA = \frac{F\_DCO}{F\_OS} \tag{4.1}$$

Based on the difference between the *"DELTA"* and *"capture"* values, the DCO and *RSEL* are adjusted according to the provided flowchart, where, in a nutshell, the algorithm will range across the set of values represented in Figure 4.1, seeking the desired frequency for the DCO.

4.1.2 Radio implementation details

The CC2420 initialization routine, as represented in Figure 4.4, starts by setting up the SPI communication between the radio and the MCU.

The **USART0** peripheral is set to **SPI** 3-pin Mode with the microcontroller as the master [15]. Additionally the necessary pins (Figure 4.2) are setup accordingly to the board schematic [17], with *P3.1* and *P3.2* working as the master's output and input respectively. The *P3.3* pin is responsible for providing the serial clock necessary for the synchronous protocol while the *P4.2* works as the slave selector.

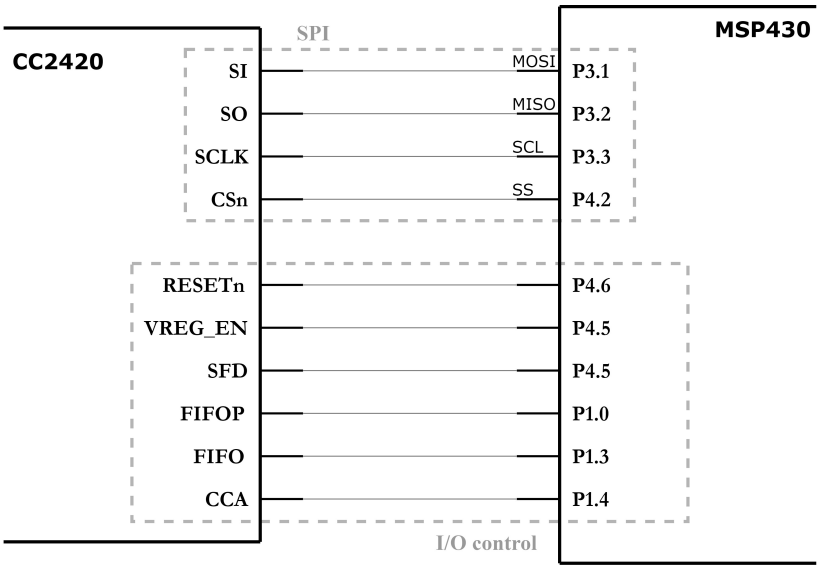


Figure 4.2: TelosB RADIO-MCU logical schematic

Following the serial communication setup, the voltage regulator that powers the radio is enabled and the device is reset, where all the configuration registers of the device will take the default values [18, chapter 37], preparing the chip for post configuration.

Through the now configured SPI connection, the radio controller is then turned on using the corresponding command strobe (SXOSCON), which turns on the internal oscillator that chips the chip's operations. The crystal requires at

most 2 ms to stabilize and when stable, the status register's respective bit will go high, that will be read until either, the crystal stabilization period has passed in case of failure, or when the oscillator has stabilized.

In case the crystal is able to stabilize, some default initial configurations are performed, such as setting the transmission power to the maximum[18, chapter 28], selecting the default channel (23) and a default PAN ID. The security features are also disabled accessing the *Security Control Register*, in order to facilitate the debugging by not requiring to set up the appropriate decryption for the Daintree packet sniffer.

The initialization process will then follow to set the radio on *RX* mode. By issuing the corresponding command strobes, first the radio will be set on the Idle State (Figure 3.3) then clearing any data from the RX buffer, and finally setting on the RX state, concluding the device initialization routine.

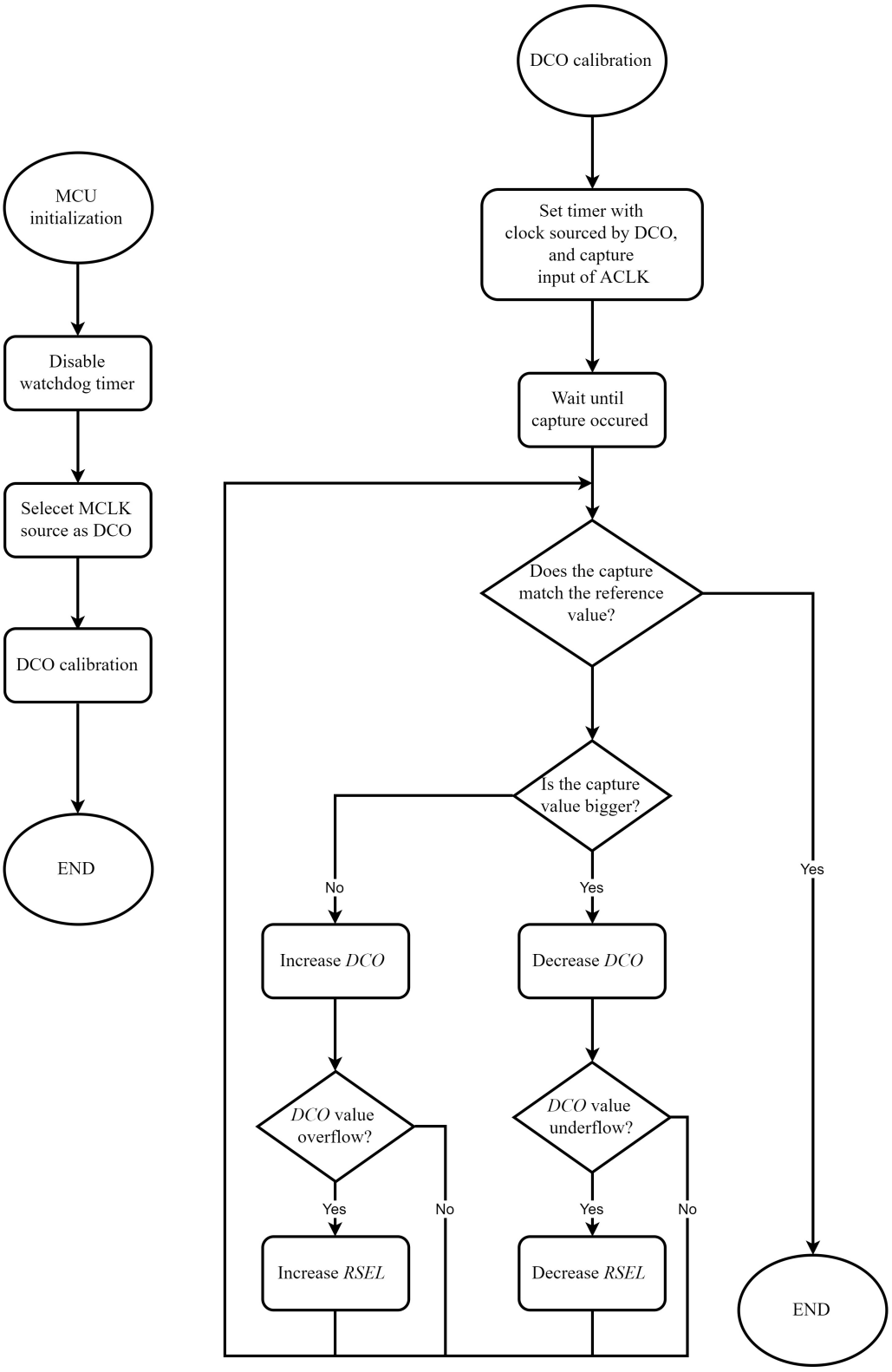


Figure 4.3: MCU initialization with DCO calibration

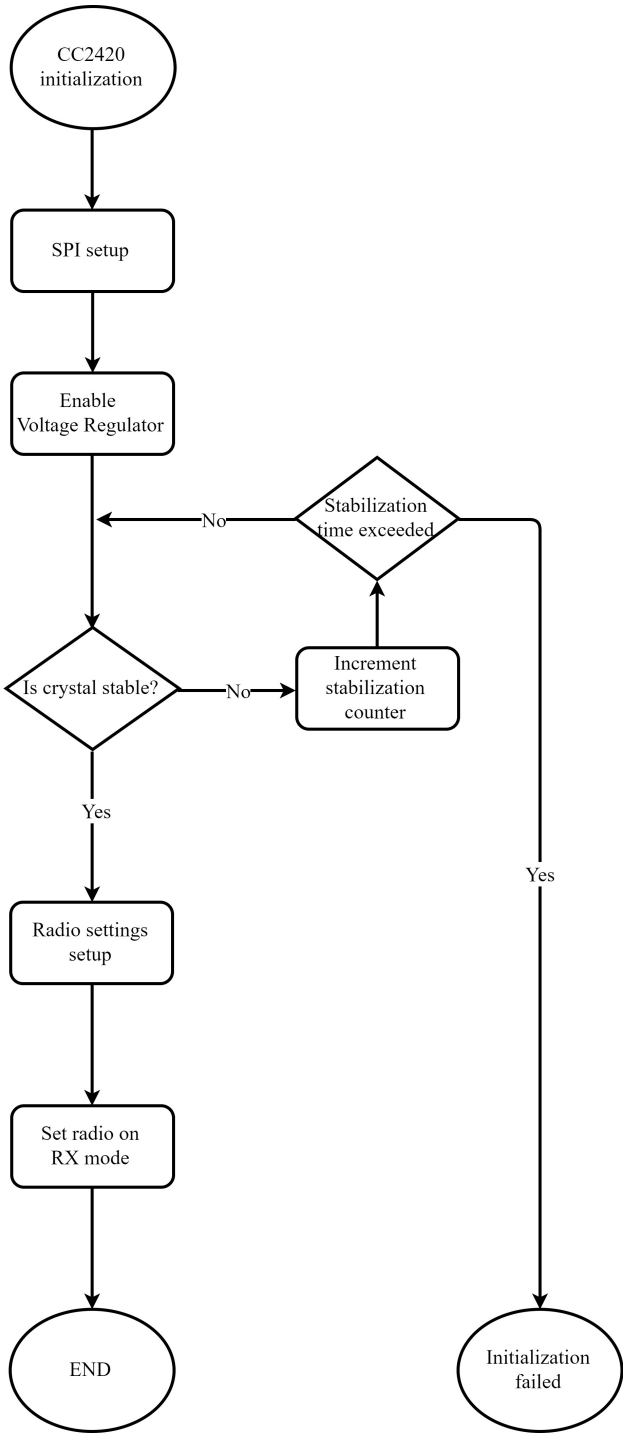


Figure 4.4: Radio initialization flowchart

4.2 An improved development environment for TelosB FreeRTOS applications

An intuitive, modular compilation environment was created using the *GNU Make* tool. With these design goals in mind, and inspired by the *ros2.0*, *riotOS* and *contiki* makefile systems, the building instructions were divided through four different makefile files.

A first one, project specific, were all the customization takes place, following the template on Appendix-B. In this makefile the application developer can define all the compiling requirements, such as the target hardware, and master paths; select any available modules, as is the case of the provided TelosB drivers; choose the appropriate memory allocation implementation by FreeRTOS; and provide any *.c* source files and path to header *.h* directory files. Besides the configuration settings, this makefile is also responsible to invoke the other makefiles that process the configurations and proceed with the compilation.

The other three files, located under the 'Makefiles' directory, were divided by the tasks each one is responsible as follows: (1) The *modules.mk* makefile associates each module to the respective source files and headers, based on the make variable *$MODULE*, where new drivers, network stack elements and layers, or any other modules, can easily be added in the future, bringing the desired modular behaviour to applications; (2) The *makefile_common.mk* takes the project makefile, which provides the source and header files, along with other configurations, flags and compiler information, and builds the *make* command string for the *MSP430-GCC* compiler; (3) The *default_target.mk* makefile, holds the typical instructions of a makefile with the necessary rules to generate the outputs (targets). The three targets can be built by the user through the make command, as follows:

```
$make clean all flash
```

The previous example results on, first, due to the 'clean' target, the removal of any previous executables and object files and then, through the 'all' target, generates the new set of objects and the application executable (*.hex*).

The 'flash' target was designed to setup the flashing tool for TelosB. The flashing tools is the MSP430 serial bootstrap loader, an open source tool [40], that the makefile calls to program the TelosB with the generated *.hex* file. The tool is a python script, that requires the pip package: python-serial.

In order to ease the implementation of the Standard, a minimalistic filesystem was created, leaving only essential modules for the FreeRTOS and removing all unused platform ports, resulting on a simplistic directory system (Figure 4.5).

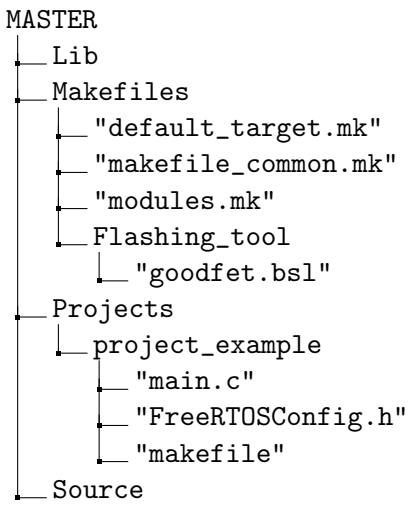


Figure 4.5: Project filesystem

The OS files, including the MCU port, are stored under the 'Source' directory, with the exception to the FreeRTOS configuration file called FreeRTOSConfig.h. Every FreeRTOS application must have a FreeRTOSConfig.h header file in its pre-processor include path. FreeRTOSConfig.h tailors the RTOS kernel to the application being built. It is therefore specific to the application, not the RTOS, and should be located in an application directory.

The TelosB peripheral drivers, network stack and protocol implementation or any other future module should be set under the 'Lib' directory.

4.3 TelosB drivers

The drivers for the mote platform resulted from the adaptation of several other TelosB implementations [37][36][31], mainly the drivers provided by **ROS2 embedded on RiotOS** [41] project, due to its modular design. While following the MSP430F1611 [15] and CC2420 [18] datasheets, the drivers were adapted in order to utilize the FreeRTOS functionalities (such as the software delays) and the module on Appendix-C was added to provide the application with easy access to the drivers.

As visible in the directory tree from Figure 4.6, the drivers were separated under the three folders: (1) 'telosb', (2) 'cc2420' and (3) 'cpu', in order to share the drivers with other platforms that utilize the same RF transceiver or MCU, seeking an organized and efficient filesystem, while storing under 'telosb', board specific instructions.

The platform is implemented following the directory tree represented in Figure 4.7, having the board initialization procedure built under **board.c**, responsible for calibrating the DCO, using the internal 32 kHz crystal, and the initialization of the microcontroller ports, setting up the chips default functionalities.

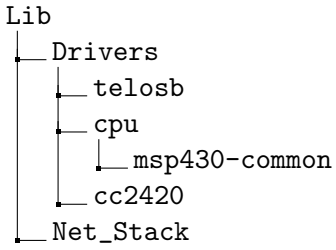


Figure 4.6: Modules library filesystem

The **board.h** header provides the generic *board_init(void)* function to the application, as well as some platform macros for LED control and frequency settings for the calibration process. The **driver_cc2420.c** bridges the CC2420 drivers to the TelosB platform by providing the pin setup and control, which include the *UART0* initialization for the SPI communication with the transceiver as well as the interrupt handlers for the radio pins.

As for the radio driver files, they are organized under the "cc2420" directory, as depicted in Figure 4.7.

The main **cc2420.c** driver file stores main control functions of the RF device, including the functionalities required by the IEEE 802.15.4, which are accessible through **cc2420.h**.

The **cc2420_settings.h** header file provides macros for the command strobes and the necessary register addresses for control and configuration of the device, while **cc2420_spi.c**, using said macros, provides the **cc2420.c** with the control

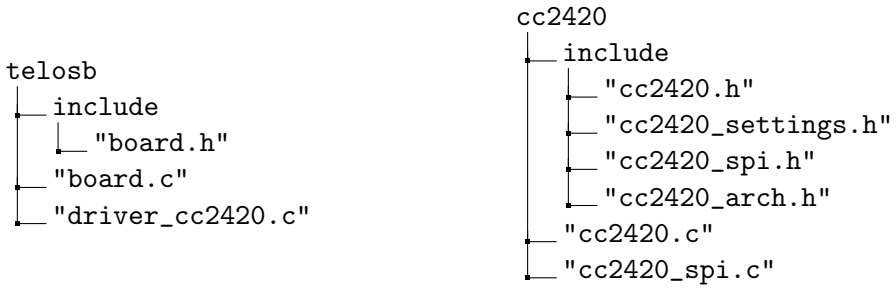


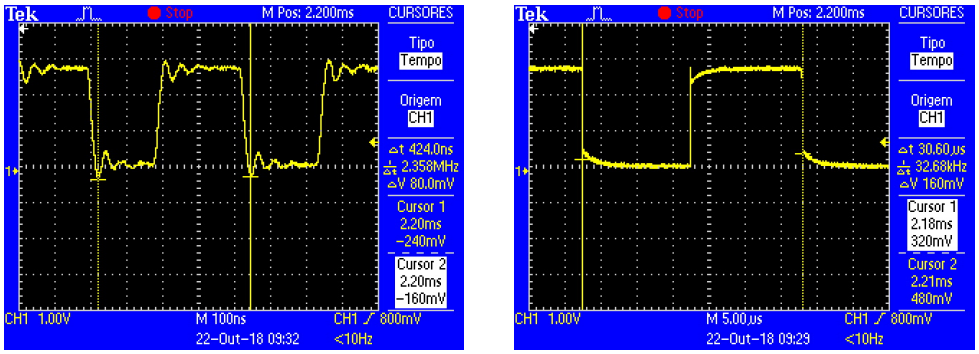
Figure 4.7: TelosB implementation filesystem

for the register content of the CC2420 chip. This functionality relies on the **cc2420_arch.h** which interfaces the singular MCU pin control drivers for the radio (**telosb/driver_cc2420.h**), required to provide the main drivers with access to the correct radio pins for a specific platform.

4.4 Experimental Validation

4.4.1 MCU clocks

The MSP430F1611 provides a mechanism for debugging the MCU clock frequencies through the port selection and enabling the special function on the appropriate pins. In order to corroborate the implementation, first, the platform initialization was tested by verifying the MCLK (main clock) calibration result as well as the ACLK frequency sourced from the 32768 Hz watch crystal. The measurements presented in Figure 4.8 were acquired utilizing the Tektronix oscilloscope [42], with an observed ACLK frequency of 32.68 kHz and 2.358 MHz with the MCLK.



(a) MCLK (b) ACLK

Figure 4.8: MSP430F1611 frequency results

The ACKL frequency (4.8b), as expected, is close to the datasheet[17] value, proving the reliability of the internal crystal as the base time driver for the IEEE 802.15.4 implementation. The MCLK (4.8a), given the volatile nature of the DCO, didn't match the provided reference value on the following macros:

```
//board.h
#define MSP430_INITIAL_CPU_SPEED     2400000uL
#define F_RC_OSCILLATOR              32768
```

This unpredictable behaviour, that would for certain compromise the determinism and, therefore, the reliability of the protocol implementation, makes the DCO unsuitable for this task. However, parallel operations, that don't have such

time critical demands and might even require a time granularity that the internal oscillator cannot provide, can, instead, use the DCO for timer purposes, specially if using the calibration process to mitigate the unsteady performance.

4.4.2 Memory footprint

Using the MSP430-gcc *'-Map'* option [43], the linker will print out the map file which contains diagnostic information along with a memory map for the application. Figure 4.9 is an example of a memory configuration, start of the RAM stack (*0x1100*) and size are available. The RAM length of *0x2800* converts to the characteristic 10 240 B registers of the MSP4301611.

```
Memory Configuration

Name             Origin             Length             Attributes
sfr              0x0000000000000000 0x0000000000000010
peripheral_8bit  0x0000000000000010 0x00000000000000f0
peripheral_16bit 0x0000000000000100 0x0000000000000100
ram_mirror       0x0000000000000200 0x0000000000000800 xw
infomem          0x0000000000001000 0x0000000000000100
infob            0x0000000000001000 0x0000000000000080
infoa            0x0000000000001080 0x0000000000000080
ram              0x0000000000001100 0x0000000000002800 xw
rom              0x0000000000004000 0x000000000000bfe0 xr
vectors          0x000000000000ffe0 0x0000000000000020
bsl              0x0000000000000000 0x0000000000000000
infoc            0x0000000000000000 0x0000000000000000
infod            0x0000000000000000 0x0000000000000000
ram2             0x0000000000000000 0x0000000000000000 xw
usbram           0x0000000000000000 0x0000000000000000 xw
far_rom          0x0000000000000000 0x0000000000000000
*default*        0x0000000000000000 0xffffffffffffffff
```

Figure 4.9: MSP-GCC compiled Mapfile example

The MSPSim emulator provides highest used RAM address of an instant through the command *stack* as seen on the example below.

```
-----------------------------------------------
MSPSim 0.99 starting firmware:
../MASTER/Projects/my_led_timers/main.elf
-----------------------------------------------
MSPSim>stack
Current stack: $14cc
```

Knowing the first and last used addresses of the stack, the total usage of the RAM by an application can be easily calculated.

In order to measure the memory footprint resulting from the platform implementation, a simple application, available on Appendix-D, was developed in order to track the memory usage with the increase of running tasks.

The results of the memory benchmark are presented on Table 4.1 providing the RAM usage of a given instance of the simulation as well as the memory usage increase along the growth of the number of tasks. The application is designed to launch a configurable number of tasks. Each has a small stack size that uses the same *vTask_MylightTask()* function in order to obtain a better representation of the OS memory management.

Table 4.1: Memory Usage Benchmark

Nº of running tasks	**Last used stack address**	**Total RAM Usage** (B)	**Memory scaling** (B)
0*	0x130e	526	
1	0x12ec	492	———
2	0x149c	924	432
3	0x1642	1346	422
4	0x17fa	1786	440
5	0x19c4	2244	458
6	0x1b6a	2666	422
7	0x1d3c	3132	466
8	0x1ee6	3558	426
9	0x209e	3998	440
10	0x2266	4454	456

*Idle task running

From the values presented in Table 4.1, it can be concluded that the memory manager is working correctly with a visible memory usage linearly increasing alongside the number of tasks. In regard to the memory usage of the Idle task, the fact that it is higher than running a single task doesn't indicate a problem with the implementation given that the created tasks are minimal and the idle task needs to perform several functions, such as energy saving mechanisms, and, still keeps a relatively low memory usage. The advertised footprint seems accurate when comparing to the value presented in section 3.4.7 by the official documentation.

Understanding how the memory usage of the application scales with the number of tasks is critical for the the design of the IEEE 802.15.4 application. This is achieved by knowing the available resources, memory wise, keeping in mind that a

more efficient memory usage by the protocol allows for more complex applications that require more RAM.

4.4.3 Radio Functionalities

In order to test the CC2420 RF transceiver, the application provided in Appendix-E was created, capable of transmitting test packets. Utilizing the Daintree Analyser sniffer functionalities, that require a minimal MAC frame format, the outcome of the application transmissions was depicted onto Figure 4.10, validating the drivers implementation.

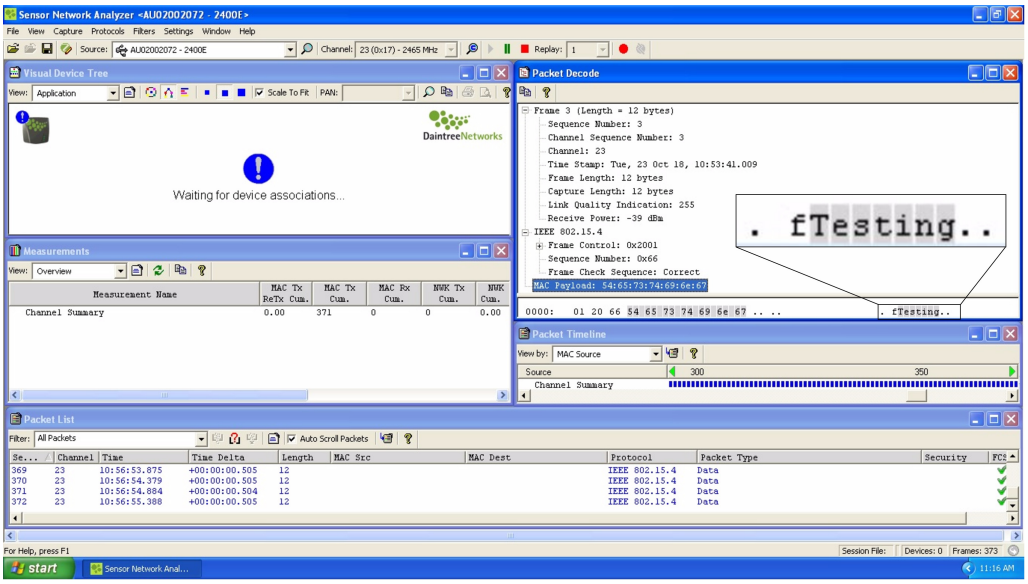


Figure 4.10: Captured packets of the radio test application (Appendix-E)

Under the "Packet Decode" tab, the frame content is interpreted and associated to the respective frame field allowing the debugging of the packet formation as well as know the payload content ("*Testing*"). Additionally, other information about the transmission, such as the frequency channel and the power of the packet, allows for the verification of the Channel Selection and Transmit Power Selection functionalities of the radio. For a periodical transmission, the application utilizes the *vTaskDelay()* function of the software timers, driven by the OS ticker. The transmission frequency, set to every 500 ms, appears very comparable to the presented value on the "Time Delta" column, under the "Packet List" window.

4.4.4 Implementation Remarks

Although not a thorough evaluation of the new ported system, these component tests verify and legitimize the TelosB implementation for the overall goal of this Thesis: Implementing the IEEE std. 802.15.4-2015 on the FreeRTOS, in particular the DSME MAC behaviour. The time granularity of the MCU and the features provided by the radio transceiver are able to meet the requirements of the protocol, while the memory management benchmark, which tests the OS scheduler, indicates that it is working correctly.

Chapter 5

Towards a Reliable and Predictable Protocol Stack

This chapter presents a preliminary design strategy to carry out the implementation of the IEEE 802.15.4 DSME MAC behaviour. Additionally, some of the most critical modules are described including the planned approach to the Multi-superframe.

When seeking a lasting and referenced implementation of the IEEE std. 802.15.4, layered, modular, fully customizable and accessible features are of paramount importance. This demands for a well designed project, while keeping the requirements in mind throughout the entire implementation.

5.1 Envisaged Architecture

This project aims to provide support for a full implementation of the IEEE std. 802.15.4-2015 on the real time operating system FreeRTOS. The targeted hardware is the TelosB, whose RF transceiver CC2420 and MSP430 microcontroller are the foundation elements to both the Standard and the RTOS operations.

Figure 5.1 maps out how these components are set to inter-operate and provide the application with a deterministic network implementation, whilst following the defined design goals. The stack is divided between hardware independent and specific components, aiming to improve cross-platform compatibility and allow for an implementation abstracted from the hardware. This way, the IEEE std. 802.15.4 is available to any platform, provided they have a FreeRTOS port and the minimum hardware (compliant radio and symbol-sensible timer), requiring only the radio and timer drivers to be interfaced with the respective SAP.

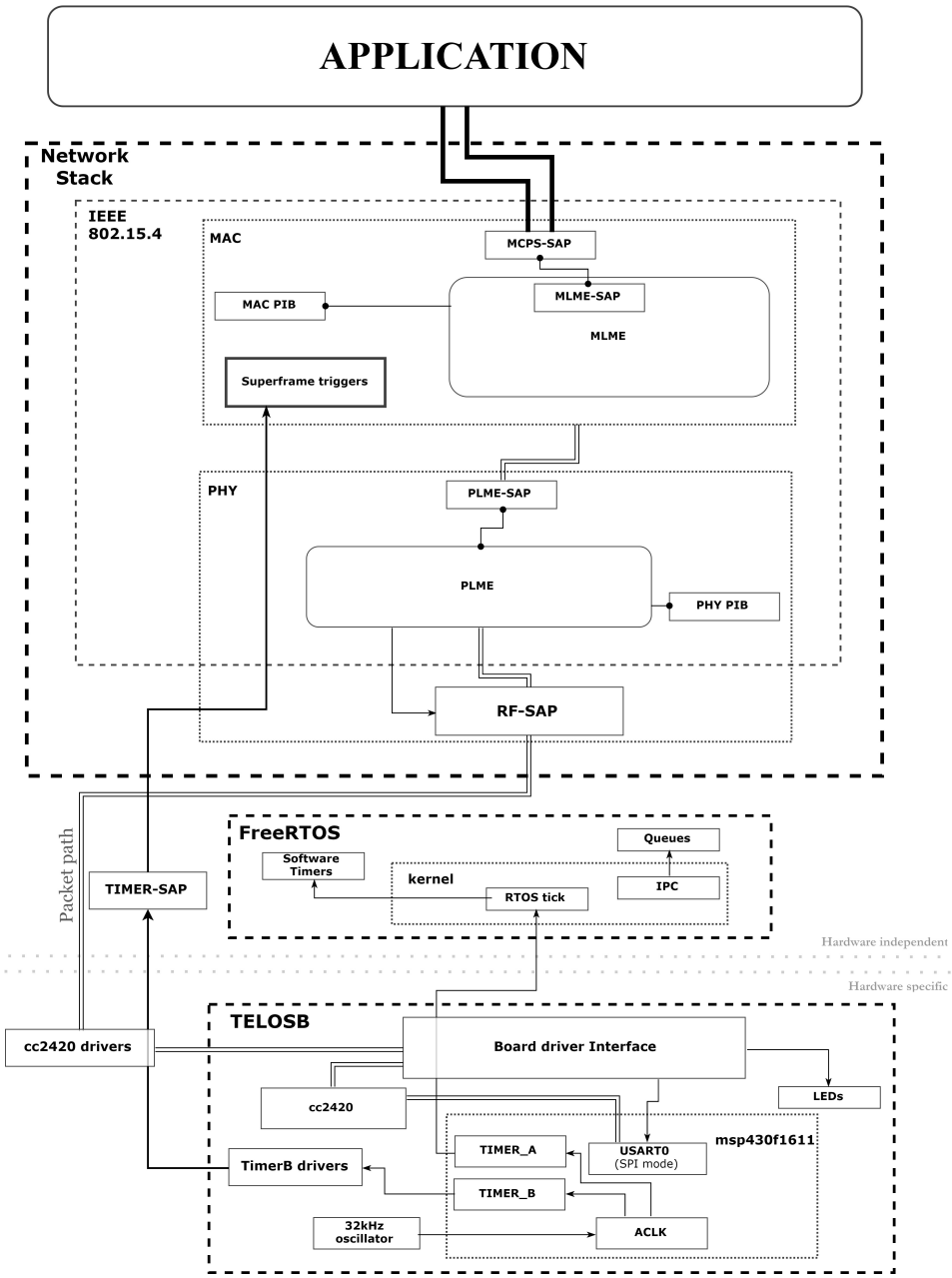


Figure 5.1: Project stack map

The hardware specific portion of the diagram was the focus of Chapter 4. With the unique hardware layer abstracted, the two service providers, RF-SAP and TIMER-SAP, are able to interface the future Network Stack with any implemented hardware platform. This increases the abstraction of the protocol implementation and enables cross-platform compatibility.

The RF-SAP module, used on the aforementioned radio test application (Appendix-E), is a generic radio interface that provides PHY with all the required functionalities (2.2.1) of the RF device by utilizing the specific radio drivers. This is achieved by compiling the appropriate radio drivers with generic function names, based on the provided platform indication through the "$BOARD" *MAKE* variable. An early version of the RF-SAP is available on Appendix-F, namely the TelosB interface, which was used during this thesis project.

The Timer-SAP, with a similar behaviour and purpose to RF-SAP, requires the hardware timer driver to fulfil the IEEE std. 802.15.4 time requirements with a symbol (16 μs) granularity. As of writing this thesis the Timer-SAP module has not yet been implemented. On the same note, a Packet building module, compliant with the protocol, is planned for implementation. Provided all the needed hardware services, the protocol Stack and FreeRTOS functionalities are accessible to the application with full abstraction.

In Figure 5.1 it is also depicted a raw representation for the planned IEEE 802.15.4 implementation, based on the components suggested by the standard and respective requirements. Additionally, the expected packet planned path is marked from the USART module up to the application. However, its expected that the protocol Stack and the Application run on concurrent tasks in order to prioritize any time critical operations of the protocol. Therefore, the communication between the application and protocol layers, namely packet exchange, is anticipated to involve the *Queue* mechanism of FreeRTOS.

5.2 Building the DSME-Superframe

The ACKL, driven by the low frequency crystal, is limited to a time granularity of $\frac{1}{32\ \ kHz} = 30,52$ μs. The time base unit of the protocol is the symbol (16 μs),. However, the TelosB is only able to provide time precision up to double the base value, still with an error of 1,48 μs every two symbols. In the face of this issue, the TKN Group [44] suggested the use of 64 kHz external crystal in order to drive a Symbol rate timer for TelosB, even providing the board design for the accessible JTAG port of the platform. This proposal, which makes use of the available XT2 oscillator slot, is a considered upgrade for the future of this project. As of now, a slot duration (60 symbols) can be used as the time base unit.

Table 5.1 provides the results of a study on how to stamp the slots required to build the DSME-Superframe using an MSP430 *timer* sourced by the reliable ACLK. Due to the granularity limits of the crystal and the resulting error of 0,74 μs the produced symbol is referred to as *"rSy"* (15,26 μs) and the forming Superframe (with a base duration) as the *"rSuperframe"*.

Table 5.1: Timer survey for the DSME-Superframe

ACLK divider	**rSy /tick**	**Ticks /slot**	**rSuperframe Duration**	**Superframe Duration**	**Error (s)**	**Error (%)**
1	2	30	14.6 ms	15.36 ms	711,6E-6	4,64
2	4	15				
4	8	7,5*				
8	16	3,75*				

*Not enough granularity to define a slot

The TelosB MCU timer allows for dividers of 1, 2, 4 and 8 that end up requiring a different amount of timer cycles, referred on the Table as *"Ticks"*, to stamp and track the slots of the superframe. The number of ticks per slot is required to have an integer value, such are the number of clock cycles,in order to not further divert from the Symbol value (16 μs). Hence the only viable divider values for the ACLK are 1 and 2, each requiring consequently a different number of ticks to total a slot. The resulting rSuperframe duration deviation from the expected *baseSuperframe* of value 15,36 ms is explicit under the "Error" columns. This deflection will only have an impact when devices based on the full Symbol (16 μs) time unit join the network. Therefore this discrepancy will not affect the end result of the implementation and only a small adjustment is needed should the full Symbol unit be adopted.

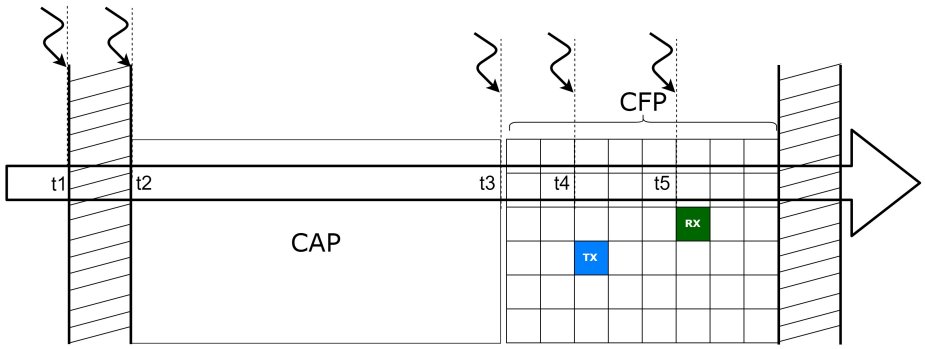


Figure 5.2: DSME-Superframe Timestamping

Should a slot granularity be achieved, building the DSME Superframe comes down to run the pertinent task and handle all associated events, following the time lines defined by the protocol variables (2.12), as is depicted in Figure 5.2.

Table 5.2: Superframe launched tasks

Event stamp	Launching task description
t1	DSME Beacon Handler (either RX or TX)
t2	Enable CSMA-CA mode and run the algorithm.
t3	CFP mode (radio off when slot is not allocated)
t4	Radio TX mode - Transmit packets on allocated slot
t5	Radio TX mode - Handle incoming packets during slot

The tasks that shape the Superframe, labelled in Table 5.2, represent an abstraction of the process required to conduct each DSME procedure, leaving aside subroutines such as the anticipated radio preparation for each task in order to maintain the timeliness of the Superframe.

Figure 5.3 portraits the FreeRTOS scheduler handling of an application running beside the IEEE 802.15.4.

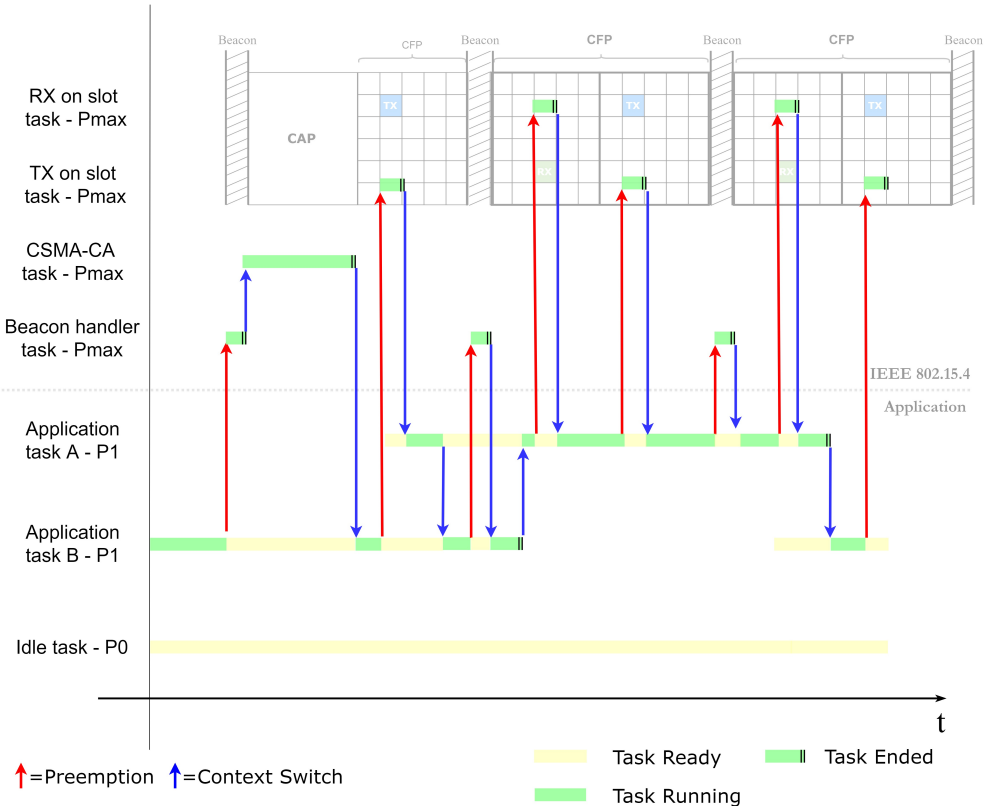


Figure 5.3: FreeRTOS Application Multitask with the IEEE 802.15.4

This example brings to light how the implementation of the DSME-superframe would impact the application execution time on a real-time OS. As visible on the first superframe of Figure 5.3, during the CAP, the usage of the CSMA-CA access method requires a total usage of the available processing time. In contrast, during the CFP region, only the allocated slots for dedicated communication employ execution time. The scheduler diagram, which handles a CAP reduction enabled DSME-superframe, further highlights the benefit of the CAP reduction that may be configured to provide the minimum necessary CAP, increasing the CFP deterministic communication span. This results on a optimization of the application execution time with an efficient and time reliable communication protocol.

Chapter 6

Conclusion and Future Work

This chapter reviews the proposed objectives of this Thesis and summarises its main contributions, detailing how the objectives were meet and the imposed work. Finally, some remarks about our future work are also presented, based on the work developed during this Thesis.

The work presented in this Thesis aims to set in motion the Implementation of the IEEE 802.15.4 DSME MAC behaviour on a real time OS. Given the magnitude of the task at hand, a thorough assessment on both the protocol and the involved technologies for its application was required to be performed. The Thesis goal was then achieved by providing the foundation support in terms of tools, software components and architecture, so that for future work, we can solely focus on the protocol implementation.

The groundwork included the TelosB hardware platform and associated drivers implementation and documentation, as well as the port of FreeRTOS for the device. This involved a survey of the available real-time OS and an extensive review of the system that eventually lead to the selection of the appropriate OS for the assigned work. Additionally, this Thesis over viewed a set of tools, the Daintree Sniffer and the MSPSim emulator, which served as support for this Thesis work and may prove helpful for the future development of the project.

All the documentation regarding the protocol and technologies involved in this work is available on this Thesis for future reference during the IEEE 802.15.4 complete implementation.

Aside from the implementation project, porting FreeRTOS, which is arguably the operating system for the IoT, to one of the most iconic motes of the IoT world, as is the case of the TelosB, is by itself quite a significant contribution. We have plans to make the port available to the community, enabling the support for a series of innovative real-time applications in the IoT arena.

6.1 Future Remarks

As suggested by the title of this Thesis, the overall proposed task is a work in progress. On that note, several of the contributions of the project are proposed approaches and procedures for the implementation of the IEEE 802.15.4 DSME protocol. Hence, the plan is to carry on the MAC behaviour implementation and build upon the work of the Thesis. Additionally, other platforms might receive a port for FreeRTOS in order to integrate the protocol implementation in other projects developed at the CISTER research centre.

Bibliography

[1] Silva Severino, Ricardo Augusto. "Improving QoS for large-scale WSNs." (2015). https://www.cister.isep.ipp.pt/docs/improving_qos_for_large_scale_wsns/1187/view.pdf, [Accessed:3-Oct-2018] [cited on p. 1, 2]

[2] Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W. and Ueda, K., 2016. Cyber-physical systems in manufacturing. CIRP Annals, 65(2), pp.621-641. [cited on p. 1]

[3] "What is An RTOS" ,
Available: https://www.freertos.org/about-RTOS.html, [Accessed:2-Oct-2018] [cited on p. 27]

[4] K. V. Prashanth, P. S. Akram and T. A. Reddy, "Real-time issues in embedded system design," 2015 International Conference on Signal Processing and Communication Engineering Systems, Guntur, 2015, pp. 167-171. [cited on p. 27]

[5] H. Kurunathan, R. Severino, A. Koubaa and E. Tovar, "IEEE 802.15.4e in a Nutshell: Survey and Performance Evaluation," in IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 1989-2010, thirdquarter 2018. [cited on p. 5, 7, 11, 12, 17]

[6] R. Severino, "On the use of IEEE 802.15.4/ZigBee for Time-Sensitive Wireless Sensor Network Applications ", https://www.cister.isep.ipp.pt/docs/on_the_use_of_ieee_802_15_4_zigbee_for_time_sensitive_wireless_sensor_network_applications/431/view.pdf, [Accessed:1-Oct-2018] [cited on p. 9]

[7] "IEEE Standard for Low-Rate Wireless Networks", in IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011), vol., no., pp.1-709, 22 April 2016 [cited on p. 2, 6, 7, 9, 10, 11, 12]

[8] "IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 15.4: Wireless medium access control

(mac) and physical layer (phy) specifications for low rate wireless personal area networks (wpans)," IEEE Std 802.15.4- 2006 (Revision of IEEE Std 802.15.4-2003), pp. 1–320, Sept 2006. [cited on p. 5]

[9] "Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans) amendment 1: Mac sublayer," IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011), pp. 1–225, April 2012. [cited on p. 6]

[10] A. Cunha, A. Koubaa, R. Severino and M. Alves, "Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS," 2007 IEEE International Conference on Mobile Adhoc and Sensor Systems, Pisa, 2007, pp. 1-12. [cited on p. 12]

[11] Wun-Cheol Jeong and Junhee Lee, "Performance evaluation of IEEE 802.15.4e DSME MAC protocol for wireless sensor networks," 2012 The First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT), Seoul, 2012, pp. 7-12. [cited on p. 15]

[12] Finn, D., Tallon, J. C., DaSilva, L. A., Van Wesemael, P., Pollin, S., Liu, W., Bouckaert, S., Vanhie-Van Gerwen, J., Michailow, N., Hauer, J.-H., Willkomm, D. and Heller, C. "Experimental Assessment of Tradeoffs among Spectrum Sensing Platforms", in Proc.of the 6th ACM Intl. Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH), pp. 11–22, September, 2011. [cited on p. 12]

[13] G. Patti, G. Alderisi and L. L. Bello, "Introducing multi-level communication in the IEEE 802.15.4e protocol: The MultiChannel-LLDN," Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, 2014, pp. 1-8. [cited on p. 13]

[14] Memsic, "TelosB Datasheet",
Available: `http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf`, [Accessed:1-Oct-2018] [cited on p. 19, 20]

[15] Texas Instruments, "MSP430f1xx1 Microcontroller Datasheet",
Available: `http://www.ti.com/lit/ds/symlink/msp430f1611.pdf`, [Accessed:1-Oct-2018] [cited on p. 19, 35, 38, 43]

[16] Texas Instruments, "MSP430fx1xx family User's guide",
Available: `https://www.ti.com/lit/ug/slau049f/slau049f.pdf`, [Accessed:8-Oct-2018] [cited on p. 21, 36, 37]

[17] Moteiv Corporation, "Telos (Rev B) Datasheet",
Available: `http://www2.ece.ohio-state.edu/~bibyk/ee582/telosMote.pdf`, [Accessed:8-Oct-2018] [cited on p. 21, 22, 38, 45]

[18] Chipcon Product from Texas Instruments, "CC2420 - 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver",
Available: `http://www.ti.com/lit/ds/symlink/cc2420.pdf`, [Accessed:1-Oct-2018] [cited on p. 19, 21, 22, 23, 38, 39, 43]

[19] Memsic, "MICAz Datasheet",
Available: `http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf`, [Accessed:1-Oct-2018] [cited on p. 20]

[20] "History of FreeRTOS",
Available: `https://www.freertos.org/RTOS.html`, [Accessed:2-Oct-2018] [cited on p. 29]

[21] "FreeRTOS - Task States",
Available: `https://www.freertos.org/RTOS-task-states.html`, [Accessed:2-Oct-2018] [cited on p. -]

[22] "FreeRTOS, Task Control",
Available: `https://www.freertos.org/a00112.html`, [Accessed:2-Oct-2018] [cited on p. 30]

[23] "FreeRTOS, Memory Management",
Available: `https://www.freertos.org/a00111.html`, [Accessed:7-Oct-2018] [cited on p. 33]

[24] C. Sabri, L. Kriaa and S. L. Azzouz, "Comparison of IoT Constrained Devices Operating Systems: A Survey," 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, 2017, pp. 369-375. [cited on p. 28, 33]

[25] O. Hahm, E. Baccelli, H. Petersen and N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things: A Survey," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 720-734, Oct. 2016. [cited on p. 28, 33]

[26] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, and T. Voigt. Accurate Network-Scale Power Profiling for Sensor Network Simulators. In Proceedings of the Fifth European Conference on Wireless Sensor Networks (EWSN2009), Cork, Ireland, pp. 312-329, February 2009. [cited on p. 26]

[27] Moteiv Corporation, "Tmote SKY Datasheet",
Available: `http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf`, [Accessed:12-Oct-2018] [cited on p. 26]

[28] M. Imran, A. M. Said and H. Hasbullah, "A survey of simulators, emulators and testbeds for wireless sensor networks," 2010 International Symposium on Information Technology, Kuala Lumpur, 2010, pp. 897-902. [cited on p. 26]

[29] MSPSim source code,
Available: https://sourceforge.net/p/mspsim/wiki/Home/, [Accessed:2-Oct-2018] [cited on p. -]

[30] "FreeRTOS Ports - Suported devices",
Available: freertos.org/a00090.html#TI, [Accessed:19-Aug-2018]. [cited on p. 35]

[31] "FreeRTOS and its port on the WSN430 hardware platform",
Available: https://github.com/iot-lab/wsn430/blob/master/OS/FreeRTOS/doc/Brief_Tutorial, [Accessed: 19-Aug-2018]. [cited on p. 33, 35, 43]

[32] Texas Instruments, "MSP430fx4x Microcontroller Datasheet",
Available: http://www.ti.com/lit/ds/symlink/msp430f449.pdf, [Accessed: 19-Aug-2018]. [cited on p. 35]

[33] "FreeRTOS Kernel files",
Available: https://sourceforge.net/projects/freertos/files/latest/download?source=files, [Accessed:4-Oct-2018] [cited on p. 35]

[34] "FreeRTOS FAQ - Memory Usage",
Available: https://www.freertos.org/FAQMem.html#RAMUse, [Accessed:7-Oct-2018] [cited on p. 33]

[35] RiotOS documentation, "Support for the TelosB board",
Available: https://riot-os.org/api/group__boards__telosb.html, [Accessed:1-Oct-2018] [cited on p. 20]

[36] TinyOS supported platforms, "Quickstart: TelosB",
Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/Quickstart:_TelosB, [Accessed:1-Oct-2018] [cited on p. 20, 43]

[37] Contiki hardware, "hardware platforms currently in the Contiki code tree: sky/telosb mote", http://www.contiki-os.org/hardware.html, [Accessed:1-Oct-2018] [cited on p. 20, 43]

[38] Daintree 2400E manual, Available: https://fccid.io/S6R2400E/User-Manual/User-Manual-566853 [Accessed:12-Sept-2018] [cited on p. 25, 26]

[39] Official MSPSim git repository,
Available: https://github.com/contiki-ng/mspsim, [Accessed:1-Oct-2018] [cited on p. 26]

[40] MSP430 USB programing tool,"GoodFET",
Available: https://github.com/travisgoodspeed/goodfet, [Accessed:5-Oct-2018] [cited on p. 42]

[41] Available drivers, "ros2 embedded riot",
Available: `https://github.com/ros2/ros2_embedded_riot/tree/master/drivers`, [Accessed:13-Oct-2018] [cited on p. 43]

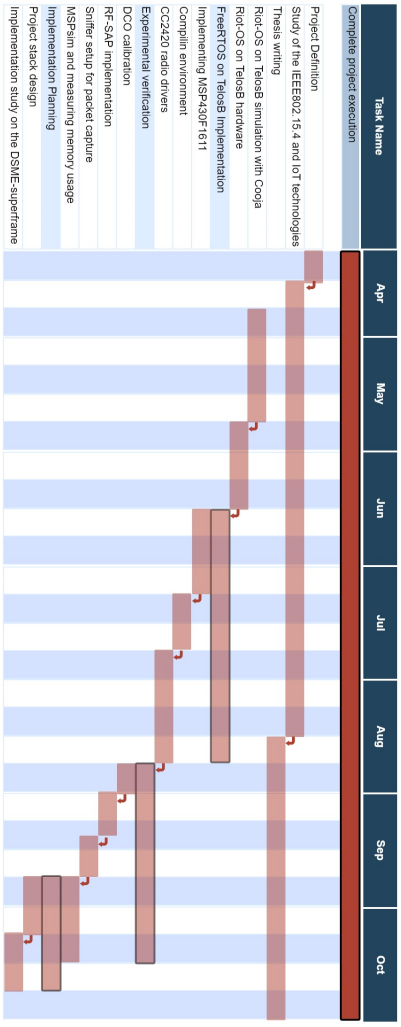
[42] Digital Storage Oscilloscopes, "TDS2000B Series Data Sheet",
Available: `http://www.testequipmenthq.com/datasheets/TEKTRONIX-TDS2014B-Datasheet.pdf`, [Accessed:15-Oct-2018] [cited on p. 45]

[43] MSP430 GCC User's Guide,
Available: `http://www.ti.com/lit/ug/slau646c/slau646c.pdf`, [Accessed:16-Oct-2018] [cited on p. 46]

[44] Köpke Andreas, Hauer Jan-hinrich, "IEEE 802.15.4 Symbol Rate Timer for TelosB", 2008
Available: `http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/timertr.pdf`, [Accessed:20-Oct-2018] [cited on p. 53]

Appendix A

Project roadmap



Appendix B

Makefile template for TelosB-FreeRTOS projects

```
# *
# * FreeRTOS Kernel V10.0.1
# * 23rd june 2018
# *
# */THIS MAKEFILE TEMPLATE WAS DESIGNED
#  FOR TELOSB AND TMOTE SKY by Crossbow
#----------------------#----------------------#----------------------#
#----------------------#----------------------#----------------------#

# Defining paths to OS source code:
MASTER_PATH  = ./../../../Telosb-FreeRTOS#should be first instruction
PROJECT_PATH = .
PORT_PATH    = $(MASTER_PATH)/Source/portable/GCC/MSP430F449#nospaces

# Defining device info:
CPU = msp430f1611
BOARD  += telosb
MODULE += phy
#--------------------------------------#-----------------------------#
#--------------------------------------#-----------------------------#
# add project headers "directory" path:

INC += ./ \
\
```

```
#-------------------------------------#-----------------------------#
#-------------------------------------#-----------------------------#
#MODULE += cc2420 #phy
include $(MASTER_PATH)/Makefiles/makefile_comon.mk
# Target name
NAMES ?= main
#-------------------------------------#-----------------------------#
#-------------------------------------#-----------------------------#
# add project sources files path:
SRC += \
        main.c \
        $(SOURCE_PATH)/portable/MemMang/heap_1.c \
\
#-------------------------------------#-----------------------------#
#-------------------------------------#-----------------------------#
include $(MAKE_PATH)/default_target.mk
#----------------------#----------------------#---------------------#
#----------------------#----------------------#---------------------#
```

Appendix C

TelosB Module

```
# File: MASTER\Makefiles\modules.mk
#-----------------------------------#----------------------------#
#-----------------------------------#----------------------------#
ifneq (,$(filter telosb,$(BOARD)))

INC += \
        $(MASTER_PATH)/Lib/drivers/telosb/include\
        $(MASTER_PATH)/Lib/drivers/cpu/msp430-common/include\
        $(MASTER_PATH)/Lib/drivers/cc2420/include\
        \

SRC += \
        $(MASTER_PATH)/Lib/drivers/telosb/board.c\
        $(MASTER_PATH)/Lib/drivers/telosb/uart.c\
        \
        \
        $(MASTER_PATH)/Lib/drivers/telosb/driver_cc2420.c\
        $(MASTER_PATH)/Lib/drivers/cc2420/cc2420_spi.c\
        $(MASTER_PATH)/Lib/drivers/cc2420/cc2420.c\
        \
        \
        $(MASTER_PATH)/Lib/drivers/cpu/msp430-common/cpu.c\
        $(MASTER_PATH)/Lib/drivers/cpu/msp430-common/irq.c\

endif
```

Appendix D

FreeRTOS RAM footprint Benchmark Application

```
1  //Pedro Neto @ CISTER 10/2018
2  /* Scheduler includes. */
3  #include "FreeRTOS.h"
4  #include "task.h"
5  #include <msp430.h>//BIT macros
6  #include "board.h"
7  #define mainLED_TASK_PRIORITY (tskIDLE_PRIORITY+1)
8
9  //=================# Memory usage test #=================//
10 #define TASK_NUMBER 10
11
12 static void vTaskMylighTask( void *pvParameters );
13 //=================# Memory usage test #=================//
14
15 //Perform Hardware initialization.
16 void telosb_init(void);
17 void cc2420_rx_handler(void);
18 /*-----------------------------------------------------------*/
19
20 void manual_delay(int steps);
21
22 int main( void )
23 {
24   /* Setup the hardware . */
25   board_init();
26
27   LED_OUT |= (BIT4 | BIT5 | BIT6); //start LEDs Off
28
29   //=================# Memory usage test #=================//
```

```
30    uint task;
31
32    while(task<TASK_NUMBER)
33    {
34      xTaskCreate( vTaskMylighTask, "task "+task,
        configMINIMAL_STACK_SIZE, NULL, mainLED_TASK_PRIORITY, NULL );
35      task++;
36    }
37
38    //====================# Memory usage test #====================//
39
40    /* Start the scheduler. */
41    vTaskStartScheduler();
42    /* As the scheduler has been started the demo application
43    tasks will be executing and we should never get here! */
44    return 0;
45  }
46
47  static void vTask_MylightTask( void *pvParameters )
48  {double i;
49    while (1)
50    {
51      // Toggle blue LED
52      for (i = 0; i < 1000; ++i){}
53      LED_OUT ^= LED_BLUE;
54    }
55  }
```

Appendix E

Radio-Test Application

```
/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"

/* Project includes */
#include "board.h"
#include "RF_SAP.h"

#define   CH  23

//MAC PDU struct-------------------------------------------------
typedef struct MPDU
{
  //info on frame type/ack/etc
  uint8_t frame_control1;
  //info on addressing fields
  uint8_t frame_control2;
  //uint16_t frame_control;
  uint8_t seq_num;
  uint8_t data[120];
}MPDU;
//--------------------------------------------------------------

/*
* Sending task
*/
static void vSendingTask(void* pvParameters);
#define SEND_TASK_PRIORITY (tskIDLE_PRIORITY+1)

void radio_setup(void);

//--------------------------------------------------------------
```

```
33 int main( void )
34 {
35   // Setup the hardware . //
36   board_init();
37
38   // Add the task
39   xTaskCreate( vSendingTask, (const signed char*) "appl",
       configMINIMAL_STACK_SIZE, NULL, SEND_TASK_PRIORITY, NULL );
40
41   // Start the scheduler.
42   vTaskStartScheduler();//loop
43   //never reached
44   return 0;
45 }
46
47
48 static void vSendingTask(void* pvParameters){
49   radio_setup();
50
51   uint8_t len, packet_bytes;
52   uint32_t i=0;
53
54   char msg[]="Testing";
55
56   //number of bytes/adresses that need to be transmited
57   len=sizeof(msg)+1;
58
59   //#-----------------Packet building-----------------------
60   struct MPDU mympdu;
61   struct MPDU *mpdu;
62   mpdu=&mympdu;
63
64
65   mympdu.frame_control1= (1<<0);//Data frame type;
66   mympdu.frame_control2= (1<<5);//frame version IEEE 802.15.4-2015
67
68   for (i = 0; i < len+1; ++i)
69   {
70     mympdu.data[i]= msg[i];
71   }
72
73   packet_bytes= 3+len;
74   //#-----------------Packet building-----------------------
75
76   //Load packet to the radio buffer
77   radio_load(&(mympdu.frame_control1), packet_bytes);
78
79
80
81
82
```

```
83    while(1)
84    {
85      radio_tx(); //Transmit what is in the buffer
86
87      LED_BLUE_TOGGLE;
88
89      vTaskDelay(500);
90    }
91  }
92
93
94  void radio_setup(void)
95  {
96    radio_init();
97    if(!radio_on())LED_RED_ON;
98    if( !(radio_set_channel(CH)==CH) )LED_RED_ON;
99  }
```

Appendix F

RF-SAP implementation for CC2420

```
1  //Pedro Neto @ CISTER 08/2018
2  #include "RF-SAP.h"
3  #include "cc2420.h"
4
5
6  void radio_init(void)
7  {
8    return cc2420_init();
9  }
10
11 bool radio_on(void)
12 {
13   return cc2420_on();
14 }
15
16 void radio_off(void)
17 {
18   return cc2420_off();
19 }
20
21 int radio_set_channel(unsigned int chan)
22 {
23   return cc2420_set_channel(chan);
24 }
25
26 bool cca(void)
27 {
28   return cc2420_channel_clear();
29 }
30
```

```
31 int radio_set_tx_power(int pow)
32 {
33   return cc2420_set_tx_power(pow);
34 }
35
36
37 //Radio transmit (load to FIFO and transmit)
38 uint8_t radio_send(uint8_t* data, uint8_t len)
39 {
40   return  cc2420_send(data,len);
41 }
42
43
44 //Load packet to the radio buffer
45 uint8_t radio_load(uint8_t* data, uint8_t len){
46   return  cc2420_load(data,len);
47 }
48
49 //Transmit what is in the buffer
50 void radio_tx(void)
51 {
52   return  cc2420_tx();
53 }
```