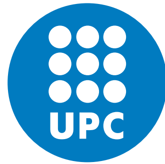
Convergence of Constrained and Non-Constrained Communication Standards

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

Submission:



Department of Network Engineering
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech
Barcelona, Spain

Author:

Martí Cervià Caballé
marti.cervia@upc.edu

Advisors:

Josep Paradells Aspas
Department of Network Engineering
josep.paradells@entel.upc.edu

Anna Calveras Augé
Department of Network Engineering
anna.calveras@entel.upc.edu

Universitat Politècnica de Catalunya

Abstract

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Departament d'Enginyeria Telemàtica

Doctor of Philosophy Thesis

by

Martí Cervià Caballé

As the Internet of Things (IoT) deployment proceeds, the number of radio solutions used in the Wireless Sensor Network (WSN) space increases. This well-known process has led to fragmentation in the IoT space, as well as to incompatibility between the constrained radio solutions used in WSN devices and the more general-purpose ones, used in most connective devices. Thus, there is a research opportunity in improving the interoperability of the overspecialized and non-inter-operable radio solutions currently used in WSN. Therefore, this thesis proposes mechanisms to achieve interoperability and, as a consequence, the convergence of WSN radio solutions. It does so by applying Wake-up Radio (WuR).

Wake-up Radio (WuR) allows devices to save power by remaining inactive while, at the same time, remaining responsive. Currently, WuR is being used to enable low-power WSN use cases with mainstream radio solutions, such as IEEE 802.11. This contribution evaluates the state-of-the-art in WSN and WuR and justifies several specific areas to contribute. First, to reduce specialization by broadening the scope of technologies that can be used in WSN, and, second, to improve compatibility between those solutions already deployed. For this purpose, this thesis develops research aimed at extending the scope of a generalist solution, legacy IEEE 802.11, to the low-power uses cases required by WSN. Next, this thesis researches improving the compatibility of already existing solutions IEEE 802.11 and IEEE 802.15.4 by using WuR to achieve interoperability.

To contribute to the adoption of IEEE 802.11 in WSN use cases, this work presents a procedure to implement a Wake-up Transmitter (WuTx) that can be implemented using legacy IEEE 802.11 transceivers. The method devised, is capable of operating at 250 kbps and is thoroughly evaluated and compared to the state-of-the-art WuR solutions compatible with IEEE 802.11 through simulations. Later, the legacy-compatible WuTx was implemented in two proof-of-concept IEEE 802.11 devices, one embedded and another Linux-based. To further contribute to enabling IEEE 802.11 use in low-power WSN, this thesis presents a WuRx (Wake-up Receiver), complementary to the previously proposed WuTx. The power consumption of this

WuRx baseband is characterized under different scenarios. Thus, jointly with the previously presented WuTx, this contribution extends low-power WSN use cases through WuR to even legacy-IEEE 802.11 devices.

Next, to improve the interoperability between WSN devices, the previous WuR developments presented in this thesis are applied to the interconnection of non-compatible devices through WuR. This application of WuR is named here WuR Cross Technology Communications (WuR-CTC). To demonstrate the WuR-CTC concept, a WuR-CTC solution is developed and implemented in a testbed, showcasing WuR-enabled communication between IEEE 802.11 and IEEE 802.15.4 devices as a way to reduce fragmentation in the WSN radio solution space. All source code and hardware design for these implementations are shared, thus, enabling reproduction and extension.

Through its research work, this thesis advances the state of the art in both low-power IEEE 802.11 use, as well as cross-technology communications. This work has showcased what can be achieved with further convergence inside the fragmented WSN radio landscape while, simultaneously, demonstrating how WuR can be applied to achieve these goals.

Acknowledgment

My acknowledgment to my coauthors, Eduard Garcia, Elena López, Ilker Demirkol that have helped me go through those journal submissions with their advise and thoughtful edition. I need to acknowledge those who have worked with me through this years in laboratory C3004, Sergio Aguilar, David Vila, Maison Hussein and Eudald Sans. Most importantly, I thank Anna Calveras and Josep Paradells, my thesis supervisors and coauthors, for their excellent advice and unending patience. This work has only been possible with the attentive support I have received from my research group, Wireless Network Group. Thanks to all of you for your knowledge, attention, understanding and tolerance for extravagant ideas.

I could not have finished this thesis without the continued support from my family, specially, my parents: Joan Cervià, and M.Queralt Caballé. Thank you for having me, in the widest sense possible. I need to thank my closest friends Pol Delgado and Jordi Orriols for the great times we have enjoyed together through these years, which have helped me to keep moving with mt work. This contribution would not be possible without my partner, Laura Grifé, who has given me all the love and support needed to go through dark times while withstanding my endless complaining and indecisiveness. Without you I would have given up long ago. I need to thank to Lluís Grifé and M.Alba Borrós for hosting me and providing a place to build a great, if somewhat improvised, workspace in the midst of a global pandemic.

My sincerest thanks to all of which have supported me through these years and without whom I would not be writing these words.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 WuR for power saving and interoperability . . . 3

1.3 Document Structure . . . 5

2 An overview of the power-saving mechanisms used in WSN **6**

2.1 The appearance of WSN . . . 6

2.2 Wireless Radio and Protocol Solutions used in WSN . . . 7

2.3 Power Consumption Reduction at MAC level . . . 9

2.4 Wake-up Radio . . . 11

2.4.1 Origin of Wake-up Radio developments . . . 11

2.4.2 State of the Art of Wake-up Radio based MAC protocols . . . 11

2.4.3 State of the Art of Wake-up Radio Receivers . . . 13

2.4.4 Wake-up Radio applied to an existing radio solution: IEEE 802.11ba . . . 16

2.5 Energy-efficient Wake-up Radio for Cross Technology Communications 18

3 A Backward-compatible IEEE 802.11 Wake-up Radio **20**

3.1 Challenges of OOK-based WuS Generation with IEEE 802.11 OFDM PHY . . . 21

3.2 Structure of IEEE 802.11a/g OFDM PHY and amplitude-based signal generation . . . 22
3.2.1 Block structure of IEEE 802.11a/g OFDM PHY . . . 22
3.2.2 Possible waveforms for WuS generation . . . 24
3.3 Generation of Peak/Flat Symbols using an IEEE 802.11a/g OFDM PHY . . . 27
3.3.1 Generation of the Peak Symbol . . . 28
3.3.2 Generation of Flat Symbols . . . 38
3.4 Simulation of the Wake-up Transmitter . . . 41
3.4.1 Implementation of the WuTx PoC . . . 42
3.4.2 PHY derived impairments in Peak-Flat Symbol generation . . . 45
3.5 Simulation of the Wake-up Receiver . . . 47
3.5.1 Architecture of the WuRx . . . 48
3.5.2 An OOK decoder . . . 49
3.5.3 A Peak detector decoder . . . 50
3.6 Results . . . 51
3.6.1 Evaluation environment . . . 51
3.6.2 Decoder evaluation results . . . 53
3.7 Compatibility with IEEE 802.11ba and other releases . . . 55
3.8 Prototype implementation . . . 56
3.8.1 Using an SDR to characterize scrambler seed variability in commodity devices . . . 56
3.8.2 Implementation of the Peak-Flat WuTx in a Linux-based device 57
3.8.3 Implementation of the Peak-Flat WuTx in an embedded device 59
3.9 Conclusion . . . 60

4 A microcontroller-based WuRx **62**

4.1 Design and Implementation of a WuRx using off-the-shelf parts . . . 63
4.1.1 WuRx RF front-end implementation 64
4.1.2 WuRx baseband implementation 65
4.2 Performance of the WuRx 71
4.2.1 An SDR-based OOK WuTx 71
4.2.2 A legacy IEEE 802.11-based WuTx 71
4.2.3 Results 72
4.3 Conclusion 73

5 Enabling Wireless Convergence with WuR 75

5.1 Contribution of WuR to the CTC state-of-the-art 76
5.1.1 Overcoming CTC limitations with WuR-CTC 79
5.2 A proof-of-concept WuR-CTC testbed 82
5.2.1 Testbed Characteristics 82
5.2.2 Standards implemented 83
5.3 PHY layer specification for the WuR-CTC testbed 84
5.3.1 WuR-CTC PPDU structure 85
5.3.2 Encapsulation of the WuR-CTC PPDU 85
5.3.3 Determining the maximum WuR-CTC PSDU size 87
5.4 Link layer specification for the WuR-CTC testbed 89
5.4.1 MAC sublayer 89
5.4.2 LLC sublayer 91
5.5 Device implementation for the WuR-CTC testbed 96
5.5.1 WuTx implementation for the WuR-CTC testbed 96
5.5.2 WuRx implementation for the WuR-CTC testbed 99

5.5.3 Software implementation of the WuR-CTC solution 104
5.6 Operation of the WuR-CTC testbed 104
5.6.1 Direct results of testbed operation 105
5.6.2 Emulation results 108
5.7 Conclusions 112

6 Conclusion and future work 114

Appendices 117

A WuRx evaluation with MATLAB 118
A.1 Implementing the WuTx 118
A.2 Implementing the transmission channel 120
A.3 Implementing the WuRx 121

B Software-based legacy-compatible WuTx implementation 124
B.1 Python WuTx implementation 124
B.2 WuTx implementation for the ESP-32 129

C Software-based WuRx baseband implementation 136
C.1 WuR-CTC receiver 136
C.2 Communications with the host device 147

D WuR-CTC host software implementation 153
D.1 WuR-CTC stack implementation 153
D.2 WuR-CTC demo scenario 158
D.2.1 Event loop 158
D.2.2 Control and display interfaces 163

List of Figures

1.1 Diagram explaining the operation of a WuR system, where the main radio is also the WuTx. 5

2.1 Circuital Diagram of a 5-element Passive WuRx. The WuRx connects to the node with the Vout terminal. 14

2.2 Block Diagram of a semi-active WuRx capable of receiving 250kbps OOK presented in [73] 15

2.3 Block Diagram of a non-heterodyne active WuRx, as proposed in [74] 15

2.4 Block Diagram of an uncertain-IF active WuRx, such as the one presented in [76]. 16

3.1 IEEE 802.11a/g OFDM PHY block structure for the data path. This block structure is also applicable to IEEE 802.11n/ac OFDM symbol generation. However, spatial multiplexing has to be taken into account. 23

3.2 Amplitude a Peak Symbol candidate, in frequency (a) and time (b) domains. Note that this representation does not include the cyclic prefix. Hence, the time domain representation has only 64 samples. . 26

3.3 Detail of a pair of Peak and Flat Symbol candidates in time domain, shown in (a) and (b), respectively. This representation was generated with the ***generate_symbol*** function. Results of low-pass filtering with a second order Butterworth filter with a cutoff of 250 kHz are outlined in red. 27

3.4 Comparison of the same symbol before (a) and after (b) guard interval insertion. 29

3.5 Linear Feedback Shift Register (LFSR) representation of the scrambler used in IEEE 802.11 OFDM PHY[84] 30

3.6 Convolutional Coder used in IEEE 802.11 PHY. Note that for each bit input, two outputs, Data A and Data B are generated. Each T_b block represents a one bit delay[84]. 31

3.7 Amplitude of best case non-ideal Peak Symbols for the "-7 -7j" subcarrier value (a) and "-1" subcarrier value (b). 37

3.8 Amplitude of the Flat Symbol generated by the binary sequence presented in (3.17) for mostly positive pilot tones (a) and mostly negative pilot tones (b). Note that its maximum amplitude is significantly lower than the peak of its respective Peak Symbol presented in Fig.3.7b. 41

3.9 Amplitude of a Flat Symbol designed for 54 Mbps with the symbol "-7 -7j", displayed with mostly positive pilot tones (a) and mostly negative pilot tones (b). 42

3.10 Insertion of the WuS inside the IEEE 802.11g frame structure. 43

3.11 The WuS as generated by the PoC WuTx, encoding the binary value "010101010101010101010" in the WuS. 44

3.12 Detail of two consecutive Peak and Flat Symbols from the waveform displayed in Fig.3.11. 45

3.13 Spectrum of a peak signal with $A_{sat} = 27$ dBm. The dashed red line represents the spectral mask. 46

3.14 Spectrum of a peak signal with $A_{sat} = 14.75$ dBm. The dashed red line represents the spectral mask. 47

3.15 Structure of the proposed WuRx. 48

3.16 Structure of the proposed decoder based on OOK reception. 49

3.17 Structure of the proposed decoder based on peak detection. 50

3.18 Diagram of the signal processing chain used for the evaluation of BER under AWGN. 52

3.19 Diagram of the signal processing chain used for TGn Ch.B PER evaluation. 53

3.20 Simulation of Peak-Flat performance on the presented receivers versus various modulations. (a) BER vs SNR in AWGN channel and, (b) PER vs SNR in TGn Ch.B. 54

3.21 The envelope of a WuS generated by a WuTx with Linksys WUSB54GCv1, using the encoding the binary value "01010101010101010" captured using a GNU Radio scope with a USRP B200 SDR. 59

4.1 A WuRx receiving a WuS and turning on its associated primary radio. 62

4.2 Structure of the proposed WuRx. 63

4.3 Components of the WuRx RF front-end. 65

4.4 Final prototype of the WuRx, with the RF front-end. 66

4.5 WuS headers. Red arrows highlight signal transitions. 68

4.6 WuS frame structure. 68

4.7 State diagram for the WuS processing. 69

4.8 OOK WuS 72

4.9 Legacy-compatible WuS to the WuRx address "0101010101010000", as received with a USRP B200. 72

5.1 Examples of communication scenarios comparing gateway-based interaction with Cross technology Communications (CTC)-based interaction 76

5.2 WuR-CTC PPDU structure. 85

5.3 WuR-CTC PPDU encapsulation format for the IEEE 802.11g WuTx. 86

5.4 WuR-CTC PPDU encapsulation format for the IEEE 802.15.4 WuTx. 87

5.5 WuR-CTC MPDU. 91

5.6 Traditional WuS interaction between two sleepy nodes. 94

5.7 Sending data to a sleepy node. 95

5.8 Request/response WuR-CTC communications between two sleepy nodes. 96

5.9 Block diagrams of the two WuRx components. (a) Structure of the RF front-end. (b) Structure of the baseband. 100

5.10 State machine representing the frame parsing procedure implemented in the WuRx microcontroller. The main reception path is highlighted in red. 103

5.11 Diagram of testbed experimental setup. 105

5.12 Throughput obtained from testbed operation. Results are shown for each of the WuTx. 106

5.13 Frame error rate obtained from testbed operation. Results are shown for each of the WuTx. 107

5.14 Transmission time obtained from testbed operation. Results are shown for each of the WuTx 108

5.15 Example of non-WuR signals and the corresponding WuRx activation length according to (5.5). 109

5.16 An iteration of the channel overlapping process. 111

5.17 Channel occupancy rate and WuRx activity rate for both scenarios. 112

A.1 Waveforms input and output by an OOK detector Simulink model. The orange waveform corresponds to the WuS waveform contaminated with noise by an AWGN channel model. The yellow waveform corresponds to the low-pass filtered input signal, fed to the positive terminal of the decoder comparator. The blue waveform corresponds to the detection threshold, fed to the negative terminal of the decoder comparator. Finally, the green waveform is the comparator output, which corresponds to the model output. 123

C.1 State diagram of the WuRx baseband activation. 137

C.2 I2C frame to perform register selection. 148

C.3 I2C frame to read WuR-CTC frame reception status. 149

C.4 I2C frame to read or write WuR-CTC address. 149

C.5 I2C frame to read WuR-CTC frame buffer. 149

D.1 State diagram of the transmitter demonstration. The internal loop that sends and validates the test frames is highlighted in red 159

D.2 Screenshot of the embedded ESP-32 control webapp 166

D.3 The EFR32MG12 development board. 169

D.4 The main screen of the application. 169

D.5 Screen showing test status. 170

List of Tables

2.1 IEEE 802.15.4 transceiver comparative. 8

2.2 Summary of the presented WuR MAC developments. 13

3.1 Parameters of the convolutional coder used in IEEE 802.11 OFDM PHY 31

3.2 Possible symbol values derived from uniform bit blocks 35

3.3 PAPR obtained for each symbol and pilot tone phase combination . . 36

3.4 Bits per symbol corresponding to each data rate supported by IEEE 802.11a/g 58

4.1 Component parameters. 64

4.2 Microcontroller parameters. 67

4.3 WuRx baseband energy consumption. 73

5.1 Characteristics of CSMA-based proposals referenced 78

5.2 Characteristics of Emulation-based proposals referenced 80

5.3 Maximum WuR-CTC PSDU Size 88

5.4 WuR CSMA/CA Parameters for the reconfigurable radio 90

5.5 Components of the WuRx RF front-end. 101

5.6 Components of the WuRx Baseband. 102

Glossary

ADC Analog-to-Digital Converter.

API Application Programming Interface.

ARQ Automatic Repeat Request.

AWGN Additive White Gaussian Noise.

CDMA Code Division Multiple Access.

CMOS Complimentary Metallic Oxide Semiconductor.

CSMA Carrier Sense Multiple Access.

CSMA-CA Carrier Sense Multiple Access with Collision Avoidance.

CSMA/CA Carrier Sense Multiple Access (CSMA) with Collision Avoidance.

CTC Cross technology Communications.

CTS Clear To Send.

DFT Discrete Fourier Transform.

FSK Frequency-Shift Keying.

GSFK Gaussian Frequency-Shift Keying (FSK).

IDFT Inverse Discrete Fourier Transform.

IEEE Institute of Electrical and Electronic Engineers.

IF Intermediate Frequency.

IFFT Inverse Fast Fourier Transform.

IoT Internet of Things.

IP Internet protocol.

LFSR Linear Feedback Shift Register.

LLC Logical Link Control.

LNA Low-Noise Amplifier.

LPWAN Low Power Wide Area Networks.

MAC Media Access Control Layer.

MCU Microcontroller.

MPDU MAC Protocol Data Unit.

MSDU MAC Service Data Unit.

MTU Maximum Transmission Unit.

NAV Network Allocation Vector.

ODMAC On-Demand MAC.

OFDM Orthogonal Frequency Division Multiplexing.

OOK On-Off Keying.

OP-amp Operational Amplifier.

OS Operating System.

OSI Open System Interconnection.

PAPR Peak-to-Average Power Ratio.

PGA Programmable Gain Amplifier.

PHY Physical Layer.

PLCP Physical Layer Convergence Procedure.

PLL Phase-Locked Loop.

POC Proof of Concept.

PPDU Physical Protocol Data Unit.

PSDU Physical Service Data Unit.

PSM Power Saving Mode.

PTW Pipelined Tone Wakeup.

RFID Radio-Frequency Identification.

RICER Receiver Initiated Cycled Receiver.

RTS Request To Send.

SDR Software Defined Radio.

SIFS Short Interframe Space.

SNR Signal to Noise Ratio.

SOC System on a Chip.

STEM Sparse Topology and Energy Management.

TCP Transport Control Protocol.

TICER Transmitter Initiated Cycled Receiver.

WINS Wireless Integrated Network Sensors.

WLAN Wireless Local Area Network.

WPAN Wireless Personal Area Networks.

WSN Wireless Sensor Networks.

WuR Wake-up Radio.

WuR-CTC WuR assisted CTC.

WuRx Wake-up Receiver.

WuS Wake-up Signal.

WuTx Wake-up Transmitter.

Chapter 1

Introduction

1.1 Motivation

During the last two decades, the Internet of Things (IoT) has evolved from an academic concept to a widely applied paradigm. Its developments are used to solve everyday challenges, such as preventive maintenance, home automation, and asset tracking, among others. From its inception, the IoT has been developed to implement the "pervasive computing" concept, presented in the influential article *The Computer for the 21st Century* [1]. Citing the article preface:

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

Pervasive computing aims to use computing to improve daily life, making it so tightly integrated that it becomes unnoticed and, consequently, taken for granted. In the same way, the IoT proposes a global network of smart objects to improve the commodity and productivity of its users in a transparent manner. Currently, IoT related developments have been incorporated in application domains as varied as industrial control, healthcare, home automation, and logistics [2].

The IoT paradigm is wide, including several areas of research in communications and networks. This work centers mainly on one of them, Wireless Sensor Networks (WSN). As defined in [3], research in WSN focuses on the study of wireless networks that perform monitoring and actuation over a large physical environment. Nonetheless, WSN research is currently applied to more reduced environments, such as in the field of domotics. Each of the elements of the WSN, also called a *node*, performs data acquisition and processing. Additionally, nodes communicate with other nodes and external networks. WSN networks face unique constraints: first, they are deployed in sites that might not feature reliable power sources, and second, usually,

the WSN includes a large number of nodes, which is required to cover the target area. Therefore, to face those challenges, the nodes used in WSN networks need to be both low-power and low-cost, making the deployment of a large enough number of them in harsh environments feasible.

The state-of-the-art in WSN has greatly evolved during the last two decades. It has moved from nodes using ad-hoc radio solutions, to energy-constrained radio standards, such as the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 specification [4] or Bluetooth release 4.1 and above [5]. Following the requirements of WSN applications, these energy-constrained solutions were developed with efficiency in mind. Therefore, they prioritize low power consumption over throughput and range. In this way, energy-constrained solutions use low-power transceivers, typically, with an active current consumption lower than 10mA. Moreover, energy-constrained solutions use Media Access Control Layer (MAC) protocols optimized for energy efficiency. Those focus on reducing the time that the transceiver is kept powered on without receiving incoming frames, i.e., the idle listening time. However, because of these domain-specific optimizations, energy-constraint solutions only offer up to a few Mbps data rates, with Maximum Transmission Unit (MTU) lower than 251 [5] bytes.

Unexpectedly, the development of wireless solutions for WSN has been prolific and it is still ongoing. Consequently, multiple non-compatible solutions compete for the WSN niche in IoT applications. Therefore, fragmentation defines the current landscape, which nowadays features multiple competing solutions and a rapid obsolescence cycle. The effect of fragmentation in the deployment of new WSN developments is considerable. For example, due to the growing number of IoT solutions, new developments might be put on hold, waiting for a dominant solution to emerge. Additionally, the extension of existing developments is slowed by the obsolescence of the solutions chosen for the initial development. Those quickly become incompatible with the solutions currently found in the market.

The low bit rates and reduced frame sizes supported by WSN-specific solutions difficult their integration with the commonly used Transport Control Protocol (TCP)/Internet protocol (IP) stack used by the bulk of internet-connected devices. Typically, devices implementing the full TCP/IP stack are non-energy-constrained and are optimized both for increased throughput and range, while their power consumption is a secondary concern. Non-energy-constrained wireless technologies include the various cellular solutions standardized by the 3GPP, such as [6] and UMTS [7], among others). Nonetheless, there are low-power cellular-oriented technologies, such as NB-IoT [8], LTE-M [8], and others [9]. Another example of a non-constrained radio technology would be the family of IEEE 802.11 specifications for Wireless Local Area Network (WLAN). The current market implantation of these technologies is much higher than the ones used for WSN, for example, IEEE 802.11, alone, will ship up to 20 billion devices in the 2019-2024 period, according to analysts [10].

The receiver consumption of non-constrained devices (which dominates the total power consumption of radio devices) is, typically, an order of magnitude higher than those of energy-constrained solutions used by WSN nodes. Nonetheless, due to their increased throughput, non-energy-constrained devices are more efficient transmitters, with lower energies per bit and higher spectral efficiency. However, due to their increased receiver consumption, devices integrating non-energy-constrained solutions cannot be used as WSN nodes in most applications. Regardless, devices using non-energy-constrained solutions are preferred as end-user devices, such as laptops and smartphones.

Therefore, there is a clear divide between constrained and non-constrained devices, which are non-compatible and, therefore, cannot interact directly. Moreover, as mentioned earlier, there is also a divide between energy-constrained devices implementing incompatible solutions. In both cases, communications between non-compatible devices must flow through gateway devices, which perform protocol translation between incompatible protocol stacks. Thus, gateways enable the formation of WSNs with heterogeneous nodes as well as their connection to the Internet through non-energy constrained endpoints.

Protocol translation has led gateways to a prominent spot in IoT networks, but its use does not come without disadvantages [11]. Gateways, by performing protocol translation, add one additional transmission to each message. Therefore, gateway use adds an additional hop that reduces the overall spectral efficiency of a network. Moreover, a gateway needs to face coexistence issues that arise between the multiple transmitters that it incorporates. Finally, gateway use introduces a reliability problem since the gateway becomes a single-point-of-failure for the whole network. For example, a gateway malfunction can interrupt both communications between non-compatible WSN nodes and internet connectivity.

Therefore, shifting WSN away from gateway-dependency would benefit the roll-out speed of the IoT, by providing networks that are more reliable and cohesive. The approach chosen in this thesis tries to face the two issues identified, the divide between energy-constrained and non-energy-constrained devices, and the fragmentation in energy-constrained solutions, using one solution: Wake-up Radio (WuR). Using WuR, the energy consumption of non-energy-constrained devices can be reduced, allowing them to operate in more energy-constrained roles in WSN. Moreover, WuR provides a way to connect constrained and non-constrained devices directly, as well as constrained devices implementing non-compatible wireless solutions.

1.2 WuR for power saving and interoperability

As mentioned earlier, the main factor that blocks the use of a certain wireless solution in WSN applications is power consumption. Specifically, the energy consumed

by performing idle reception, which dominates the overall figure among the traffic profiles seen in WSN [12]. Therefore, it is most important to reduce the energy wasted by the receiver when not receiving incoming frames, the *idle listening* time. Thus, even a non-low-power receiver, such as those found in non-energy-constrained devices can still be used in an energy-constrained role if idle listening is minimized.

Idle listening can be reduced by using MAC schemes that minimize the time that the receiver is kept enabled, waiting for frames. For example, both Bluetooth [13] and IEEE 802.11 Power Saving Mode (PSM) [14], disable the receiver most of the time. In those solutions, the receiver is only activated periodically for a brief interval. Such a scheme increases the latency of the interactions, which become, on average, half the duration of the period that the receiver spends powered off. Here, non-energy-constrained devices are at disadvantage since the energy consumption introduced by their less efficient receivers must be compensated for. To obtain a duty cycle low-enough to enable WSN operation with non-energy-constrained receivers, the proportion of time said receiver spends active must be extremely low. Consequently, the time that the receiver is powered off increases, and, as expected, so does latency. For example, the low power modes introduced by IEEE 802.11ah define sleep periods that can go from one second, and up to years [15].

There are other techniques, other than synchronous duty-cycling that allow for energy savings at low latency. For example, LoRaWAN class-A endpoints [16] can asynchronously wake up once they have frames buffered for transmission and then remain awoken for a certain time, waiting for responses. However, to offer this power-saving measure, the LoRa gateway device cannot save power by sleeping and must remain active permanently. This technique is then not applicable in the mesh topologies typical of WSN. In those, some devices will be acting as routers and, with such a scheme, will not be able to save power.

Nonetheless, WuR [17] enables a low receiver duty cycle on all devices present in the network, and additionally, provides low latency. In WuR, all devices incorporate an ultra-low-power receiver, called Wake-up Receiver (WuRx). When not expecting transmissions, WuR enabled devices turn off their main radio and go to a low-power state, i.e., to sleep. However, the WuRx of the device is kept active, listening to the medium, for Wake-up Signal (WuS) sent from other stations that, when received correctly, awakes the device from sleep. The device that generates the WuS is the Wake-up Transmitter (WuTx), which can be either a specifically purposed transmitter or just the main radio of the device. The interaction of a wake-up is clearly explained in Fig.1.1. Also, since, with WuR, the devices are only activated on-demand, WuR power-saving is competitive against optimized traditional duty-cycled schemes [18].

With WuR, non-energy-constraint devices can reduce their power consumption. Thus, allowing them to perform energy-constrained roles in WSN applications and, at the same time, retain their advantages in range and throughput. This way, WuR

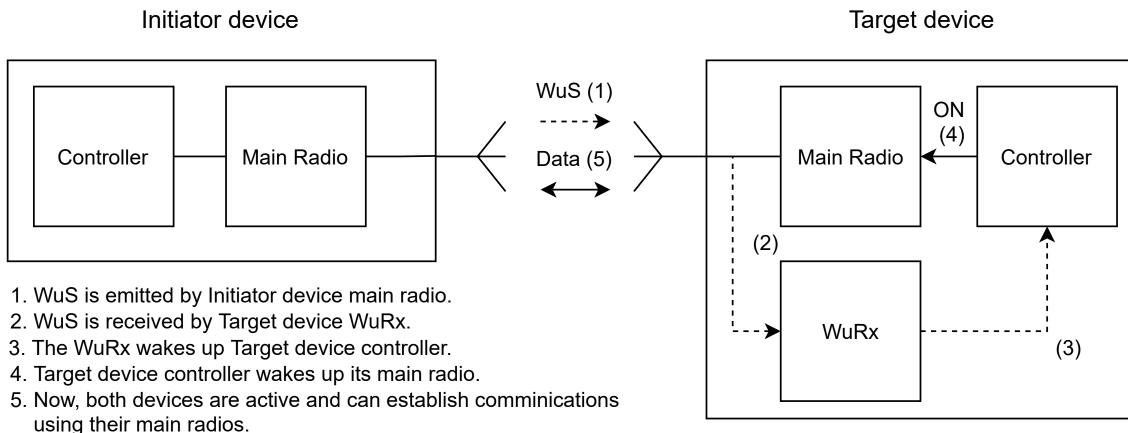


Figure 1.1: Diagram explaining the operation of a WuR system, where the main radio is also the WuTx.

can enable WSN applications to benefit from the economies of scale of general-purpose wireless devices. Moreover, the WuS can carry binary data, a capability originally introduced to add addressing information. However, this capability also allows devices to interact with their WuR hardware, regardless of the wireless solutions adopted for their main radio. This way, WuR can be used to allow non-compatible WSN nodes can interact with each other, as well as enable direct connectivity from the WSN to end-devices and the Internet.

1.3 Document Structure

This thesis is structured in the following way: first, Chapter 2 presents the general background for this thesis and motivates its developments; second, Chapter 3 presents the development of a legacy-compatible alternative to IEEE 8023.11ba; thirdly, Chapter 4 introduces the implementation of a low-cost microcontroller-based WuRx compatible with the previously presented legacy-compatible WuR solution; next, Chapter 5 describes the application of the previous developments to another field, Cross Standard Communications; finally, 6 presents the conclusions. Afterward, the appendices introduce supplementary material.

Chapter 2

An overview of the power-saving mechanisms used in WSN

2.1 The appearance of WSN

As with many other technologies, the first WSN fulfilled military purposes. When introduced in the 90s, the objective of these networks was to perform reconnaissance and track enemy targets [19, 20], for this purpose, a self-organizing wireless mesh network was proposed. One important result that appeared from these programs was the Wireless Integrated Network Sensors (WINS) project, which used a single Complimentary Metallic Oxide Semiconductor (CMOS) chip, integrating sensors, processing, and a low-power radio on a reduced component footprint [19].

Eventually, further developments introduced applications of WSN in civilian uses. Those are wide, comprising maintenance, monitoring, domotics, entertainment, and surveillance, among others [2]. A pioneer in this approach was the Smart Dust project [21]. This influential paper introduced the “mote”, as in mote of dust, to name WSN nodes. The naming was inspired by the pervasive nanotechnology-based devices featured in the cyberpunk novel *The Diamond Age* by Neal Stephenson [22]. Smart Dust presents the applicability of WSN and elaborates on the design challenges to be faced. Finally, it proposes a theoretical WSN solution based on low-power nodes implemented with monolithic integrated circuits, like those proposed by WINS [19]. Smart Dust also proposed the use of a mesh network structure, which was to be formed by a high number of devices optimized for low power consumption.

However, at that point, there were no wireless specifications that conformed to the WSN requirements. Therefore, subsequent WSN developments used ad-hoc radio solutions, as well as purpose build protocol stacks. Nonetheless, due to the increased costs intrinsic to these specific solutions, the applicability of WSN developments to

widely applicable commercial applications remained low.

Consequently, standardization efforts were started on several fronts. These efforts were aimed at enabling commercially viable WSN developments. The results of this drive were materialized in several specialized wireless solutions, as well as the adaptation of other solutions previously developed.

2.2 Wireless Radio and Protocol Solutions used in WSN

To fulfill the requirements of WSN, solutions involving several layers of the Open System Interconnection (OSI) protocol stack are needed. In this way, Physical Layer (PHY) and MAC solutions for low-power radio links were developed concurrently with the advances on the network layer.

WSN research has greatly benefited from the development of Wireless Personal Area Networks (WPAN), originally targeted at body area networking, intending to support wearable sensors and various computing interfaces. Similar to WSN, WPAN require low-power, low-complexity, and physically reduced radios. Consequently, research in WSN adopted WPAN standards, as they share most of the same requirements and, with some amendments, are also suited for operation in WSN.

The most prevalent solution through the development of WSN has been IEEE 802.15.4 [23], which is supported by the IEEE task force dedicated to WSN use [24]. As with other popular IEEE-backed specifications, the IEEE 802.15.4 defines the first two layers of the OSI protocol model, the PHY and link layers. Several protocol stacks used in WSN research, as well as commercial applications, have extended IEEE 802.15.4 with functionality in higher protocol layers. The most prominent of those are Zigbee [25] and Thread [26]. IEEE 802.15.4 is also used with several 6LoWPAN-based stacks [27], such as the uIP stack [28], featured in the Contiki Operating System (OS) [29]. Afterward, Bluetooth [13] entered the WSN space in 2018, with its fifth release, Bluetooth 5.0, which finally added first-class support for mesh networks into the main Bluetooth specification. Nonetheless, are other specifications that are still relevant to commercial applications of WSN. For example, Z-Wave and Insteon [30, 31] are closed specifications that gained traction in the home automation market. Nonetheless, those have failed to gain significant traction in research. Moreover, there are other low-power radio solutions on the market, such as those under the Low Power Wide Area Networks (LPWAN) category [32]. Although also widely used in IoT applications, those are not related to WSN as its network topology is not a mesh, but a star, closely following the model of cellular radio system.

Regardless of its origin, specifications targeted at WPAN and WSN contemplate small devices without access to a non-finite reliable power supply. For example, those considered use a battery of limited capacity (i.e, less than 1000 mAh) as a power source, which might be complemented with energy-harvesting. A widely-used power source in WSN devices without energy harvesting are coin-sized lithium-ion batteries. Those have rated capacities that usually fall lower than 500 mAh, supporting current loads of up to 25 mA without significant reduction in performance [33]. Nonetheless, devices with energy harvesting capabilities use a wider range of energy storage solutions, including rechargeable batteries and super-capacitors [34].

As a consequence, the physical layer of those specifications uses uncomplicated modulations, paired with low-power transceivers. Specifically, those that use signaling based on phase or frequency, which do not require complex highly-linear amplification nor highly performant decoder structures. For example, the IEEE 802.15.4 specification [23] uses several phase-based modulations in its PHY implementations. Another example is Bluetooth [13], which uses a variant of FSK with gaussian windowing, Gaussian FSK (GSFK). Consequently, state-of-the-art transmitters implementing these specifications have low power consumptions, both in transmission, and reception modes. Currently, those are in the order of 10 mA in reception mode, and 20 mA in transmission mode, when considering an output power in the order of 10 dBm [35, 36]. The specifications of several state-of-the-art devices used in WSN can be found below, in table 2.1.

Table 2.1: IEEE 802.15.4 transceiver comparative.

Device	Tx Power	Sensitivity	Tx Current	Rx Current
CC2650[37]	5 dBm	-100 dBm	5.9 mA	6.1 mA
NRF52840[36]	8 dBm	-100 dBm	4.6 mA	4.8 mA
EFR32MG13[38]	19.5 dBm	-102.7 dBm	131 mA	10.3 mA

Nevertheless, a low-power transceiver is not enough to support extended battery-powered operation. Especially, for longer periods (i.e, months to years), as is required in WSN applications. For example, here is the case of an IEEE 802.15.4 radio featured in Table 2.1, the nRF52840, which consumes 4.8 mA on reception mode. Considering continuous operation in reception mode, a rough approximation of the battery life with a 500 mAh coin battery would approximately be only 4.35 days.

Fortunately, WSN use cases require the nodes to send and receive small amounts of data periodically, with a long period of inactivity. Therefore, to maximize battery life, the device, and its radio, can be kept inactive most of the time. Using the same example as before, reducing the receiver activity to 1% would extend the battery life of the nRF52840 to 435 days. Thus, by reducing idle listening time, extended battery-powered operation can be achieved with current WSN devices.

The power-saving mechanisms used to reduce idle listening time control the activation of the transceiver. Therefore, these regulate when the transceiver can send or

receive messages. Consequently, these power-saving mechanisms are considered part of the MAC sublayer. Their design principles, as well as a taxonomy of the existing MAC-level power-saving mechanisms, will be discussed in the next section.

2.3 Power Consumption Reduction at MAC level

A wide array of MAC-level energy-saving techniques are described in the literature. The aim of those is to allow WSN devices that can either be powered using energy scavenging [39] or withstand years of use with a single battery charge. The last is a requirement for most WSN applications [40]. Consequently, MAC power-saving mechanisms aim to reduce the device duty cycle by reducing *idle listening time*, which adds up most of the total energy consumption of a WSN node [17].

If devices do not coordinate their activation status, communications become impossible, since inactive nodes by definition cannot receive messages. Therefore, coordination between nodes implementing MAC-level power-saving schemes is required. As a consequence, each power-saving MAC scheme presents a coordination mechanism, also called a *rendez-vous* scheme. These can be classified into three different types according to the taxonomy presented in [41]:

1. **Synchronous rendez-vous schemes**:

   In synchronous schemes, the nodes coordinate their wake-ups at periodic time intervals. This type of technique can reduce greatly the amount of idle listening time, just at the cost of the protocol overhead required for device synchronization. The main drawback of synchronous schemes is waste. Devices need to wake up according to the schedule, regardless if there is any message to be sent or received. Another significant drawback of these types of schemes is that they introduce a compromise between power saving and latency. By using a lower duty cycle, a device sleeps for more time each cycle, increasing the time between wake-ups. As a consequence, the average latency of communications, which is half the sleep interval, increases. Additionally, for long sleep cycles, low drift clocks become a requirement. These are not typically found in feature-limited devices and, therefore, increment the cost of a WSN deployment. Research in synchronous protocols gained interest with the development of S-MAC, which proposed the use of synchronized device clusters [42]. A prolific variation of duty-cycled MAC protocols appeared consecutively with specialized variations and refinements of S-MAC for diverse low-power applications [43]. Additionally, a synchronous *rendez-vous* scheme is used to reduce power drain in a non-constrained protocol, IEEE 802.11 [14].

2. **Pseudo-synchronous rendez-vous schemes**:

In this type of scheme, the nodes sleep with a certain periodicity, however, there is no synchronization between the wake-up intervals of the different nodes. Alternatively to the synchronous schemes, here, the *rendez-vous* happens opportunistically, when the transmission of one node coincides with the wake-up of another. This system provides a simple manner to achieve energy savings through duty-cycling, without any of the protocol overhead required for synchronization. However, it is less efficient in channel usage than the synchronous version. To awaken the receiving node, the transmitter must send lengthy preambles to ensure that the frames bearing data are received [44]. Nonetheless, this scheme suffers from, generally, poor synchronization, which leads to frame repetitions, increasing the overall energy consumption of nodes. Additionally, pseudo-synchronous schemes suffer from the same latency vs efficiency trade-off as synchronous schemes, since the nodes still have a wake-up schedule. A pseudo-asynchronous *rendez-vous* scheme was first used in 2002 in the Mica platform [45] and, later, in 2004 on the WiseNET project [46, 47]. The research on asynchronous MAC schemes produced more results such as B-MAC [48] and the default MAC used in the Contiki project, ContikiMAC [49].

3. **Asynchronous rendez-vous schemes**:

   In asynchronous schemes, nodes wake up other nodes on-demand, using a side-channel for wake-up signaling. This technique, ideally, offers better energy efficiency than synchronous schemes, as the device will be only woken up when it is required. Besides, this scheme does not introduce a power vs latency trade-off since wake-ups occur on-demand. However, it requires the addition of a secondary ultra-low-power transceiver to the device. This specialized hardware allows the device to receive wake-up requests from other nodes. This comes at the cost of extra hardware and protocol overhead for the wake-up requests. Additionally, the secondary transceiver increases the energy consumption in sleep mode. When the transceiver is a radio-frequency device, this scheme is called WuR, which appeared applied to WSN in 2002 [50], when advancements in CMOS technology allowed for the production of ultra-low-powered wake-up circuits [51]. Since then, research in asynchronous *rendez-vous* has matured both in the radio and non-radio based alternatives applying the latest developments in microelectronics to reduce transceiver power consumption, down to the nano-watt scale [52]. This has allowed wake-up radio to improve over the energy efficiency results obtained with synchronous *rendez-vous schemes* [18].

2.4 Wake-up Radio

2.4.1 Origin of Wake-up Radio developments

The use of a remote wake-up signal in WSN as a *rendez-vous* mechanism was not mentioned until 2002, with the Picoradio WSN project, developed by the University of California [53]. One of the articles produced by the Picoradio project [50] proposed using WuR as a MAC scheme to save power. The goal for the WuR was to achieve an average node current consumption lower than 100μW. This figure was proposed in [50] to enable either indefinite WSN node operation using power-harvesting technologies or a lifespan of at least one year on batteries.

The main fields in WuR research are protocols and WuRx designs. The following subsections contain the state of the art in these two topics.

2.4.2 State of the Art of Wake-up Radio based MAC protocols

Protocol level research into WuR has explored several possibilities to reduce power consumption, latency or increase the security of WSN devices. Most of these research initiatives are not developed in hardware testbeds but in network simulators like OMNET++ [54] or the NS family [55]. A quick historical view of MAC WuR protocol research, with a balanced view of the most impactful articles, is presented next. It has been compiled using references from the fine survey compiled by Djiroun and Djenouri [56] and the survey developed by Piyare et al. [52].

As presented before, the first WuR MAC came in 2001, later into the development of the Picoradio project. The authors of [57] present a WuR MAC protocol with addressing, as well as Code Division Multiple Access (CDMA). The MAC proposed includes an algorithm to assign neighboring nodes different CDMA codes for their WuS, thus, if a WuS is transmitted with the code of one of the neighbors, the rest would not receive it, which avoids spending energy on false wake-ups.

Afterward, in 2002, [58] proposed a novel WuR MAC protocol, Sparse Topology and Energy Management (STEM). The main feature of STEM was the use of two different frequency bands: one for the primary radio and, another, for the WuR. In this way, STEM introduced out-of-band WuR to reduce congestion of the main radio frequency band. STEM was implemented in a testbed but, due to the still poor WuR transceiver state-of-the-art, the WuRx was implemented using a low-power receiver with a light duty cycle.

An optimization of STEM, called the Pipelined Tone Wakeup (PTW) scheme, was

presented in 2004 [59]. PTW introduced latency minimization in multihop WSN. Authors of [59] proposed chaining the wake-ups of all nodes in the path of the transmitted frame to minimize the total latency, i.e., the WuS for the next hop is transmitted by the next hop immediately after waking up, waking up the whole transmission chain before forwarding the frame to it using the main radio.

Later, in 2005 [60] presented a WuR MAC for multihop WSN. The protocol presented relied on buffering to reduce the number of device wake-ups. Moreover, it balanced the wake-ups requests between multiple nodes using estimations of their incoming frame rates. By using buffering this protocol sacrifices part of the latency advantage of WuR, but it reduces wake-up overhead and, as a consequence, it further reduces device power consumption. Nonetheless, the efforts presented in [60] were started, partly, due to the still considerable power consumption of WuRx from its period. For more context, the available WuRx of the period used more than 380 μW [53], while state-of-the-art WuRx can present an energy consumption lower than several nW [52].

In 2009, the usage of an Out-of-Band WuR was proposed not only for wake-up functionality but also for the transmission of management messages [61]. The proposal embedded the request for using the transmission medium and its acknowledgment (i.e. the Request To Send (RTS) and Clear To Send (CTS) frames used in many wireless protocols [4, 62]) into the WuS.

Afterward, in 2011, On-Demand MAC (ODMAC) [63] a receiver-initiated MAC protocol. In ODMAC, a node wakes up its neighbors periodically via beacons, indicating that is ready to receive frames. Such scheme is caller Receiver Initiated Cycled Receiver (RICER). If any of the neighbors has a frame to transmit, they contend for the medium using Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) to send buffered frames. Lastly, in 2013, [64] introduced a WuR enabled variant of the same concept with WuR-Transmitter Initiated Cycled Receiver (TICER) which is analogous to RICER, but with the transmitter sending a WuS when it has frames buffered and ready to transmit.

Further research in WuR MAC has found uses in several applications. Some highlights are the health monitoring oriented MAC developed in 2012 by [65]; also, the WPAN network-oriented MAC defined in 2015 by [66]; the BATS project, developed in 2016 by [67], defines a WuR MAC for nodes located on bats flying in the wild, intending to acquire their location. Moreover, recently a WuR MAC has been developed by the IEEE 802.11ba task group for its use with IEEE 802.11[68]. A summary of the MAC-layer developments is shown in Table 2.2.

Table 2.2: Summary of the presented WuR MAC developments.

Protocol	Highlight
Picoradio [57]	CDMA for Addressing
STEM [58]	Out-of-Band WuR
PTW [59]	Mesh Oriented WuR
[60]	Buffering and scheduling
WUR-MAC[61]	Management frames on WuR channel
ODMAC[63]	Introduction of RICER to WuR
WhMAC[64]	Introduction of TICER to WuR

2.4.3 State of the Art of Wake-up Radio Receivers

Another avenue of WuR research is WuRx hardware development. WuRx are critical to any WuR scheme since their power consumption determines how much energy the device consumes while inactive. Therefore, the trend in this area is to use newer CMOS technologies to produce lower power radios, as well as to propose architectural improvements to increase range or further reduce power consumption. WuRx implementations need to balance a trade-off of range vs. power consumption. For example, lower power designs, tend to be mostly passive and, therefore, have poorer amplification and filtering characteristics. Typically, these tend to present poor sensitivities, limiting the WuRx range. Nonetheless, most of the WuRx developed tend to favor lowering the power consumption as much as possible, while keeping the sensitivity threshold low enough to operate in their target application.

This section introduces a timeline of relevant WuRx research using a taxonomy of three different architectural types of WuRx: passive, semi-active, and active. The aforementioned types of WuRx will be introduced along with one or more relevant implementation examples. As in the previous section, this compilation has been produced mainly with references extracted from the Piyare et al. [52] survey, as well as Demirkol et al. [69] survey.

Passive WuRx are exclusively implemented using passive components and, in the same way as Radio-Frequency Identification (RFID) devices, only use the incident RF power from the WuS to operate. In [70] authors proposed using passive radio-triggered circuits for WuR. These circuits need to accumulate enough energy from the WuS to produce a high voltage level that triggers the wake-up of the node via interruption. In this way, depending on the distance and the transmitted power, radio-triggered circuits can take seconds to awake the target node. Moreover, simple radio-triggered circuits cannot support addressing in the WuS. Those only use the continued presence of power in the transmission media as their signaling mechanism, without further discriminators. For example, in [70] a basic radio-triggered circuit is presented. The block diagram of this circuit is shown in Fig.2.1. The circuit just features 5 elements: an antenna, an RF-transformer to increase voltage, a series

resistance to limit current, a low-threshold diode for rectification of the incoming RF WuS, and a capacitor to store charge. Of course, passive WuRx has the advantage in energy efficiency when compared to other WuRx solutions. These do not add additional power consumption when the node is asleep, waiting for a WuS. However, regardless of their superior energy efficiency, fully passive designs have not gained traction in the current state of the art, as they suffer either from long wake-up times or short ranges.

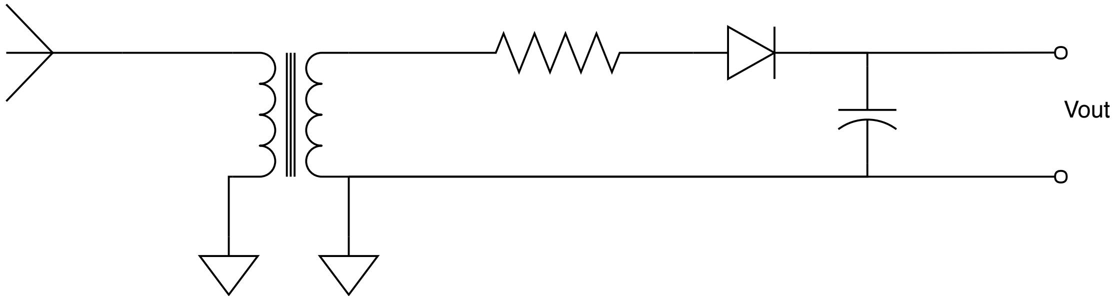


Figure 2.1: Circuital Diagram of a 5-element Passive WuRx. The WuRx connects to the node with the Vout terminal.

However, authors of [70] also presented the following category, **semi-active WuRx**. Semi-active designs use the same radio-triggered RF front-end as passive designs, however, they feature active components that enable them to receive, and parse, digital data. In [70] a semi-active design is proposed using the same basic radio-triggered-circuit shown in Fig.2.1 as its radio front-end. The circuit adds a digital baseband used to receive digital data as well as a baseband amplifier to extend its operating range. Since the design includes active components, it is no longer a passive WuRx and has a static power consumption figure. Moreover, by adding a digital baseband section, semi-active WuRx can distinguish between device addresses with the help of an address correlator.

Several variations of semi-active designs can be found in the literature, however, all use a mostly passive RF front-end circuitry, which is terminated with an active signal processing chain. Currently, semi-active designs include short-range nano-watt scale WuRx. One of the first prominent semi-active WuRx designs was featured in CargoNet, presented in 2007 [71]. The CargoNet prototype was designed and implemented to monitor environmental conditions of individual packages inside a shipping container, using short-range wireless communications and a semi-active WuRx, with a power consumption of 2.8 μW and a throughput of 750 bps. Later, in 2009, authors of [72] presented a semi-active WuRx with addressing and digital data transmission with a power consumption of 12.528 μW for a tested range of 10 meters [72].

Active WuRx uses powered circuitry both in the radio front-end and the baseband section. In contrast to semi-active designs, active designs feature non-baseband amplification and, in some cases, even heterodyne signal detection. Therefore, active

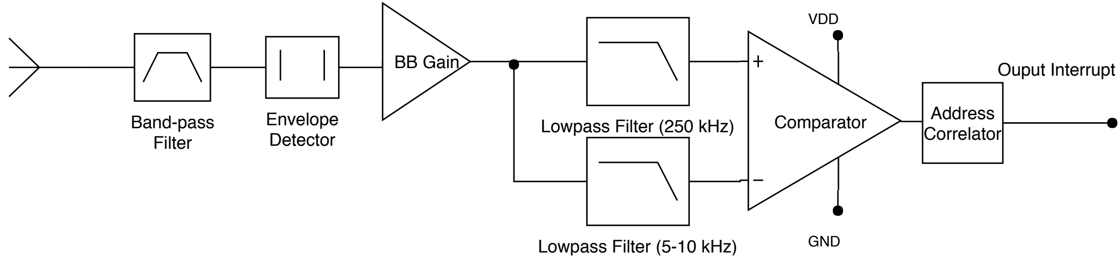


Figure 2.2: Block Diagram of a semi-active WuRx capable of receiving 250kbps OOK presented in [73]

WuRx designs present higher power consumption than semi-passive WuRx. In contrast, active designs present lower sensitivity as well as higher throughput. Due to their characteristics, active WuRx are used in long-range and low-latency WuR systems. One of the first fully-featured active WuRx was presented in 2007 [74], pioneering the development of a WuRx implemented in a CMOS integrated circuit. The prototype achieved a power consumption of 65μW, with a sensitivity of -50 dBm using 40 kbps with On-Off Keying (OOK) modulation, well under the proposed upper limit for WuRx power consumption of 100μW formulated in [50]. To reduce power consumption a non-heterodyne receiver design was used. Such a receiver does not make use of some of the most power-intensive RF components, e.g., Phase-Locked Loops (PLLs) and high-frequency oscillators. In their place, to demodulate the incoming RF signal to baseband, the WuRx proposed by them uses a CMOS based envelope detector, which was based on the design proposed in [75]. This design introduced common functional blocks of non-heterodyne WuRx RF front-end: an envelope detector, a Programmable Gain Amplifier (PGA), and an Analog-to-Digital Converter (ADC), which is implemented with a single comparator in most designs.

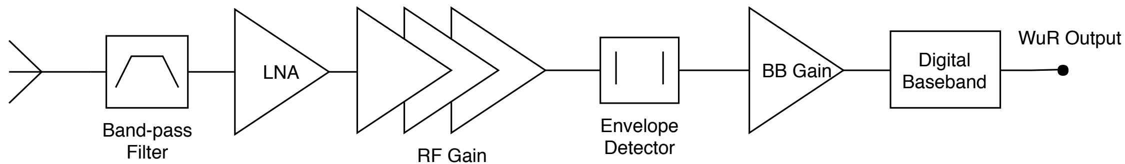


Figure 2.3: Block Diagram of a non-heterodyne active WuRx, as proposed in [74]

A landmark development in active WuRx development was the development of low-power heterodyne designs, presented in [76]. Contrary to standard heterodyne radio designs, the design proposed by them does not use precise oscillators and a PLLs to synthesize the frequency required for demodulation. Instead, it uses a 3 stage CMOS ring oscillator, the simplest possible, with 3 NOT gates. Although the oscillator has an important frequency error (in [76] frequency errors are in the range of +- 100 Mhz on a 2 GHz carrier), the demodulated signal produced is at an Intermediate Frequency (IF) *close enough* to baseband. As a consequence, signal amplification becomes much more energy-efficient. In this way, the design presented in [76] achieved

a 52 μW power consumption and a sensitivity of -72 dBm at 100kbps.

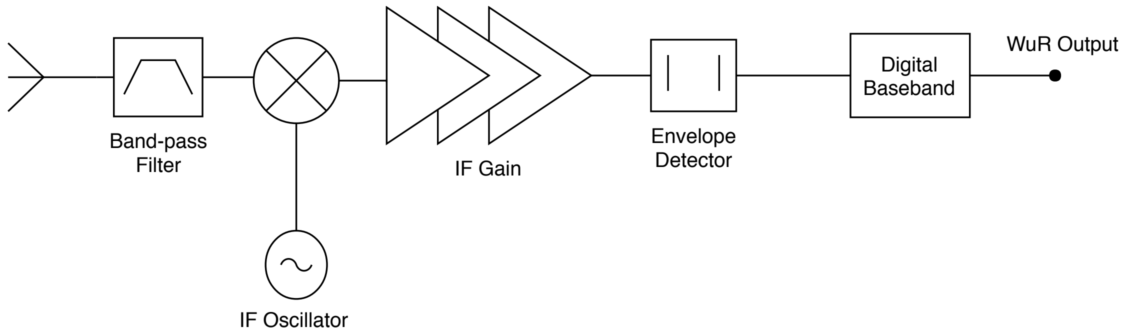


Figure 2.4: Block Diagram of an uncertain-IF active WuRx, such as the one presented in [76].

The current state-of-the-art WuRx features both semi-active and active designs aimed at different use cases. Semi-active designs are leading in terms of power consumption. This is exemplified by the semi-active design in [77], which achieves a power consumption of 4.5nW at a sensitivity of -69 dBm. This design achieves a remarkable power consumption using an almost-passive RF front-end where the only active element is a CMOS-based active envelope detector. This low-power front-end ends with an ultra-low-power comparator and a power-optimized baseband, working at 300 bps. However, both in sensitivity and throughput, active designs lead the way. One example is the active WuRx presented by [78], which reaches -97 dBm of sensitivity, at 10 kbps, while consuming 99μW. It can be considered a refinement of the uncertain-IF architecture presented in [76] that solves some of the noise floor problems introduced by the usage of an unknown IF through feedback loops.WW

2.4.4 Wake-up Radio applied to an existing radio solution: IEEE 802.11ba

Recently, WLAN) solutions have entered the IoT market in force, capturing up to 48% of new developments [79]. Nonetheless, WLAN devices still are not able to operate in most energy-restricted domains. Thus, enabling non-energy-constrained devices for low-power uses remains a challenge for the development of the WSN. In this way, WSN use cases can benefit from the reduced cost of these solutions, as well as from their greater compatibility.

The idle listening time represents the bulk of the energy consumption of non-energy-constrained devices. Nonetheless, the power consumption of non-constrained radios operating in reception is up to an order of magnitude higher, when comparing state-of-the-art devices [37, 80]. Therefore, it is challenging to achieve a power consumption low enough to permit battery-powered operation during long periods. For example, if a MAC power-saving mechanism with a traditional synchronous *rendez-*

vous mechanism is used, for a given activity period, sleep intervals need to be up to ten times higher than with energy-constrained devices. Conversely, the activity period could be reduced by a factor of ten, to maintain latency. However, this introduces tighter synchronization requirements, which might no longer be feasible to implement in feature-limited devices. Thus, the increased energy consumption inherent to non-energy constrained devices often leads to an intolerable latency increase. This effect is magnified in multi-hop environments, where several wake-ups must be coordinated for a given data packet. However,WuR provides a solution to the problem, since it offers, potentially, extremely low duty cycles without introducing significant latency penalties. As a consequence, using WuR appears as a viable way of enabling non-energy-constrained devices for low-power operation, principally in applications that require low latency. Additionally, non-energy constrained devices are more efficient as transmitters, with radios that provide lower energies per bit, as well as higher throughput. This improves the energy efficiency of non-energy constrained devices in WuR schemes.

The introduction of WuR to non-energy-constrained devices can be performed in multiple ways. The most naive one being the addition of a pair of stand-alone WuTx and WuRx to each device. However, such a solution increments the cost per device. A more elegant solution is to use the transmitter of the main radio as WuTx. Moreover, common-transmitter solutions can maintain the interoperability between legacy and WuR enabled devices. If the legacy transmitter can be used as WuTx, then, legacy devices can wake up sleeping WuR-enabled devices to interact with them.

Consequently, due to the growing interest in WuR, IEEE initiated a standardization effort to incorporate WuR into the IEEE 802.11 specification. This standardization process is, at the moment of writing, being drafted in the forthcoming IEEE 802.11ba specification [81] [68]. One of the defining characteristics of IEEE 802.11ba is its use of the IEEE 802.11 radio as WuTx to reduce implementation costs. IEEE 802.11ba, like most other WuR systems [17], transmits a WuS coded using OOK. However, none of the PHY specified in the current IEEE 802.11 specification supports the transmission of symbols with low constant amplitude, as required by OOK. As a consequence, legacy IEEE 802.11 devices are unable to transmit the OOK modulation with fidelity. Consequently, IEEE 802.11ba proposes a modified version of the 802.11 Orthogonal Frequency Division Multiplexing (OFDM) PHY transmitter that is capable of transmitting waveforms with close to zero amplitude, thus, allowing for reliable OOK symbol transmission.

As a result of these modifications, the transmitters of existing, and already deployed, IEEE 802.11 devices are incompatible with the new specification, since they cannot generate the WuS defined by IEEE 802.11ba. Keeping backward compatibility with legacy transmitters is necessary for interoperability between legacy IEEE 802.11 devices and the new IEEE 802.11ba enabled devices. Without backward compatibility, a legacy device is not able to interact directly with a sleeping IEEE 802.11ba device

since it cannot send the WuS required for its wake-up.

Therefore, the development of a legacy compatible solution, which could send IEEE 802.11 symbols with legacy transmitters is a worthwhile contribution. Such a solution is capable of accelerating the implantation of WuR in WLAN technologies. This thesis presents a software-based WuTx design that is compatible with legacy IEEE 802.11g transceivers. First, the solution was evaluated using simulations, and, afterward, it was implemented and validated in a Linux-based device as well as in the ESP-32 embedded IEEE 802.11 platform. Finally, a low-cost WuRx design compatible with the legacy WuTx solution is presented. To improve reproducibility and access, the WuRx is based on off-the-shelf parts.

2.5 Energy-efficient Wake-up Radio for Cross Technology Communications

As mentioned earlier, the currently deployed WSN have a dependency on gateway devices. Gateways are required for interactions between heterogeneous devices inside the WSN, as well as for interaction with devices external to the WSN, which implement WLAN technologies. Nonetheless, as mentioned earlier, in Section 1.1, gateway devices come with drawbacks [11]. Thus, there is an interest in developing solutions that enable heterogeneous devices to interact directly. This way, appeared CTC as a research area that investigates ways of providing direct interaction between devices that implement non-compatible wireless solutions.

To communicate non-compatible devices, CTC developments propose communication modes that, in general, were not intended by the designers of wireless solutions. As a consequence, CTC solutions do come with prominent drawbacks. There are different approaches to CTC, which are discussed in depth in Chapter 5. However, all of them suffer, at least, from one of the two following drawbacks. The first is unidirectional communications. For example, in a prominent CTC implementation, IEEE 802.11 devices can send messages to IEEE 802.15.4 devices, but communication cannot occur in the inverse order, i.e, from the IEEE 802.15.4 devices to the IEEE 802.11 ones. The second prominent drawback is the extremely low data rates achieved by most CTC solutions. This drawback affects those systems that can achieve bidirectional communications. Specifically, CTC solutions in the literature offer throughput rates as low as 16 bits per second [82]. Additionally, CTC transmitters typically use their full-rate spectrum consumption to achieve low CTC data rates. An IEEE 802.11 transmitter from [82] achieves a 16 bits per second throughput while using its full 20 MHz bandwidth, thus only providing a spectral efficiency of 0,0000008 bits/Hz. In contrast, a simple OOK implementation, like those used in WuR, can ideally achieve a rate of 1 bit/Hz.

This contribution on WuR, which adds additional communication capabilities to devices. In the same way, WuR-enabled devices can send binary addresses, they can also send arbitrary binary data. Thus, WuR provides a secondary communications channel that is independent of the technology used in the main radio of the device. Therefore, heterogeneous devices can use their WuR hardware to establish CTC.

WuR assisted CTC (WuR-CTC) can provide improvements over existing CTC solutions. First, WuR offers a symmetric communication link. All devices can incorporate both a WuTx and a WuRx. Thus, WuR-CTC enables fully bidirectional communications, in a clear improvement over the majority of CTC solutions. Additionally, WuR systems, as previously shown in Section 2.4, can achieve relatively high data rates, which can be up to the hundreds of kbps, as it is specified in the IEEE 802.11ba drafts [68]. Those are orders of magnitude higher than most of those offered by most CTC implementations and closer to the rates offered by widely used WSN solutions, such as Bluetooth and IEEE 802.15.4. Thus, WuR-CTC overcomes simultaneously both of the main drawbacks of CTC, in a clear improvement over the state of the art. However, these advantages over previous CTC solutions come at a cost, the introduction of additional hardware elements to the device to support WuR. Typically, this is fulfilled by the addition of a WuRx, while the main radio also performs as WuTx. Moreover, the presence of WuR hardware also enables the device to reduce its power consumption to the level required for its target WSN application. Thus, WuR-CTC adds additional functionality using hardware that, can be present in WSN devices for power-saving purposes.

To achieve interoperability, WuR hardware incorporated in devices must remain compatible. Currently, with an ongoing WuR standardization process, it is critical to explore WuR-CTC concept, which provides future WuR specifications a compelling reason to maintain compatibility with previous WuR developments, such as IEEE 802.11ba.

Thus, researching the feasibility of the application of WuR to CTC is a timely and worthwhile contribution. Moreover, the legacy-compatible IEEE 802.11 WuR implementation developed for this work provides the building blocks for the evaluation of WuR-CTC. This thesis contributes with a formulation of the WuR-CTC concept, as well as with a real-world testbed of WuR-CTC between energy-constrained devices and WLAN devices. Thus, showcasing the advantages of using WuR to enable direct interaction between energy-constrained nodes on a WSN and external devices and networks. With this contribution, it is demonstrated that WuR-CTC can enable direct communications between heterogeneous devices.

Chapter 3

A Backward-compatible IEEE 802.11 Wake-up Radio

Following the motivation presented in Section 2.4.4, this chapter presents the design and implementation of a WuR solution that is compatible with legacy IEEE 802.11 transmitters. It features a WuTx that is backward-compatible with the legacy IEEE 802.11 OFDM PHY specification. Instead of using OOK symbols, which cannot be created with a legacy IEEE 802.11 transmitter, the WuTx uses of a novel amplitude-based modulation. This modulation, Peak-Flat, can be received by OOK receivers as an imperfect OOK modulation. Thus, allowing its use for low-power and feature-limited WuRx. Thus, this chapter presents the design, the implementation and the evaluation of this backward-compatible WuTx solution.

This chapter is structured as follows. Section 3.1 exposes the challenges of enabling WuR with IEEE 802.11-related technologies, which use OFDM radios. Section 3.2 presents the structure of IEEE 802.11 OFDM PHY and proposes the Peak-Flat modulation as a legacy-compatible alternative to OOK. Section 3.3 explains the generation of Peak-Flat Symbols using an IEEE 802.11g transmitter. Section 3.4 proposes a WuS generation method using Peak-Flat for IEEE 802.11g. Section 3.5 describes and compares two possible WuRx architectures. Section 3.6 compares the performance of Peak-Flat against OOK-based modulations using the previously described WuTx and WuRx. Section 3.7 develops the topic of compatibility between the proposed WuR system and IEEE 802.11ba. Section 3.8 presents two physical implementations of the WuTx derived from this work. Finally, Section 3.9 presents the conclusion of this chapter.

3.1 Challenges of OOK-based WuS Generation with IEEE 802.11 OFDM PHY

An OFDM transmitter is a flexible radio that can create complex waveforms from digital data. Therefore, an OFDM transmitter can be used, with the right inputs, to generate amplitude-based waveforms that can be decoded by low-power receivers [83].

The use of OFDM to transmit data was introduced to the IEEE 802.11 specification with the IEEE 802.11 OFDM PHY, which firstly appeared with IEEE 802.11a specification. Afterward, in 2003, IEEE 802.11g was published, which extended OFDM PHY for operation at the 2.4 GHz ISM band. IEEE 802.11 OFDM PHY data rates were improved subsequently with IEEE 802.11n High-Throughput (HT) in 2009 and again with IEEE 802.11ac Very High Throughput (VHT) published in 2013.

As mentioned in Section 2.4.4, IEEE 802.11 OFDM PHY is not able to generate good quality OOK signals. A critical parameter of an OOK transmitter is its extinction ratio. This parameter is defined as the relation between the squared amplitude of the "0" symbol and the squared amplitude of the "1" symbol. This way, a higher extinction ratio implies that "0" symbols are further away in the constellation from the "1" symbols, for a given transmission power level, therefore reducing the probability of errors in symbol detection. The main challenge for creating an OOK modulation using a transmitter implementing the IEEE 802.11 OFDM PHY is achieving a sufficient extinction ratio.

Two IEEE 802.11 OFDM PHY characteristics reduce the achievable extinction ratio by adding a constant power to the OFDM symbol. First, the addition of four high amplitude pilot tones to each OFDM symbol, and second, the lack of symbols with "0" amplitude in the constellations by IEEE 802.11 OFDM PHY [84]. Therefore, there is no way to create an OFDM symbol with an amplitude close to 0 with legacy IEEE 802.11 OFDM PHY.

In IEEE 802.11ba, these problems are sidestepped by using a modified IEEE 802.11 OFDM PHY, which allows setting subcarrier values to 0. This feature enables the generation of "0" OOK symbols with constant 0 amplitude. Additionally, the DC subcarrier, deactivated in the original IEEE 802.11 OFDM PHY, is also enabled in IEEE 802.11ba to help with the generation of smoother pulses for "1" OOK symbols. The contributors to the IEEE 802.11ba have reached good results in synthesizing OOK symbols with this technique, called Multi-Carrier OOK (MC-OOK) [85, 86].

Thus, enabling WuR with legacy IEEE radios requires an alternative procedure to generate an amplitude-based modulation that can be received by low-power WuRx. Such an alternative would enable the generation of an amplitude-based signal capable of bearing digital information using an unmodified IEEE 802.11 OFDM PHY

transmitter. Moreover, if the symbols produced by the alternative could be decoded by OOK receivers, a compatibility level with IEEE 802.11ba could be achieved.

3.2 Structure of IEEE 802.11a/g OFDM PHY and amplitude-based signal generation

The first design decision is to determine in which IEEE 802.11 amendment should the legacy-compatible WuR solution be developed. As the purpose of this development is to include legacy compatibility, the ideal targets for the effort are the oldest and more widespread OFDM-based amendments, IEEE 802.11a/g. Moreover, devices implementing newer amendments are required to support these for backward compatibility. For this reason, 802.11a/g is chosen for the implementation of the WuR Proof of Concept (POC) presented in this thesis.

Additionally, the extra features found in subsequent amendments of the IEEE 802.11 OFDM PHY do not impose modifications to the WuR concept discussed and allow for the applicability of the same principles.

3.2.1 Block structure of IEEE 802.11a/g OFDM PHY

To implement a backward-compatible IEEE 802.11 WuTx, the complete and unmodified IEEE 802.11 OFDM PHY must be used. This way, the resulting WuTx will be compatible with virtually all already deployed devices, without further modifications. The implementation of such a WuTx does only require software/firmware upgrades. Nonetheless, the IEEE 802.11 OFDM PHY specification defines a complex signal processing chain, which incorporates several features such as scrambling, interleaving, and channel codification. All of these procedures perform modifications to the bitstream inputted to the IEEE 802.11 OFDM PHY and change the waveform of the OFDM symbol output by the radio. Therefore, a software-based WuTx must address its effects to obtain a predictable signal at the output of the IEEE 802.11 OFDM PHY. The block structure of the IEEE 802.11a/g OFDM PHY, from the insertion of the entry bitstream to the radio output is presented in Fig.3.1.

The signal processing chain for IEEE 802.11a/g is the following. First, the entry bitstream is injected into the Scrambler block, which adds by an XOR function a pseudo-random sequence to the entry bitstream. This procedure decreases the probability of systematic errors and improves transmitter performance by reducing the output signal dynamic range. Afterward, the output of the Scrambler block is encoded with the Convolutional Coder block. For this, the IEEE 802.11a/g specification defines coding rates between 1/2 and 3/4. Once coded, the resulting bitstream

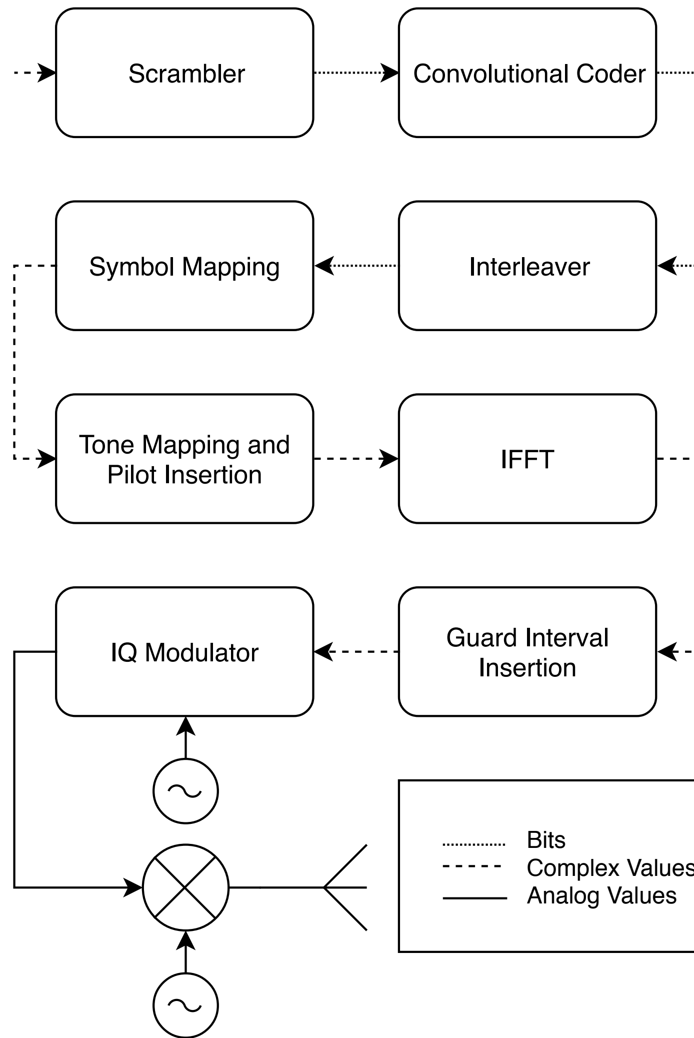


Figure 3.1: IEEE 802.11a/g OFDM PHY block structure for the data path. This block structure is also applicable to IEEE 802.11n/ac OFDM symbol generation. However, spatial multiplexing has to be taken into account.

is separated into blocks of bits, each of these the size of an OFDM symbol payload, and their contents are interleaved by the Interleaver block. This procedure separates consecutive bits of the entry bitstream, thus reducing the likelihood of error bursts and improving the effectiveness of the convolutional code.

Subsequent blocks, i.e., from Symbol Mapping to IQ Modulation, perform the transformation of each of the aforementioned blocks of bits to OFDM symbols. First, the Symbol Mapping block converts the binary values from the bit block into complex symbols, using one of the four available constellations: BPSK, QPSK, 16-QAM, or 64-QAM. Thus, the bit block output by the Interleaver results in a symbol block containing 48 complex samples. Then, four maximum amplitude symbols are added through the Tone Mapping and Pilot insertion block. This procedure, called tone addition, increments the number of symbols in the block from 48 to 52. Next, the

52 subcarriers are padded with “0” symbols, increasing the size of the symbol block to a multiple of 2, 64 samples. Later, the symbol block is sent to the Inverse Fast Fourier Transform (IFFT) block, which applies a 64-point IFFT to the bit block. Its output is a sequence of 64 time-domain data samples, which are the values of the OFDM symbol in the time domain. Finally, the cyclic prefix is prepended in the Guard Interval Insertion block. The cyclic prefix, which comprises the last 16 samples of the time domain OFDM symbol, is added to mitigate the effect of multipath propagation on the OFDM symbol and keep inter-carrier interference low. Thus, the final OFDM symbol consists of a total of 80 time-domain samples, which are sent to the IQ Modulator block. Subsequently, the modulator transforms the samples of the OFDM symbol into the analog domain at a rate of 20 Msps, obtaining a 4μs long waveform. Finally, this waveform is sent through the RF front end.

3.2.2 Possible waveforms for WuS generation

A simulation of the last stages of IEEE 802.11a/g OFDM PHY transmitter (i.e, from Symbol Mapping to Guard Interval Insertion) was developed to study which waveforms could be generated using software with IEEE 802.11a/g OFDM PHY. For this purpose, firstly, the MATLAB WLAN Toolbox [87] was studied. Nonetheless, this toolbox did not provide explicit functional support to implement only this subset of blocks from the complete signal processing pipeline. Therefore, the simulation framework for this task was implemented using MATLAB standard library functions. To ensure accuracy, the results obtained in this simulation framework were cross-validated with the output waveforms obtained with MATLAB WLAN Toolbox on equivalent inputs. According to the limitations of legacy IEEE 802.11 transmitters, other types of amplitude-based waveforms than OOK symbols were considered.

The core of this effort is the ***generate_symbol*** function, shown below. This function codes a block of bits, as if it was output by the Interleaver block, and obtains its corresponding waveform in the time domain. Thus, the ***generate_symbol*** function implements symbol mapping, pilot tone insertion, and cyclic prefix addition, as well as windowing. This function takes as inputs the bit block, the size of the symbol block, and the phase of the pilot tones.

```
1 function waveform = generate_symbol(data_packet, Bits_per_Symbol, ...
    p_phase)
2     %64 samples for IFFT
3     N = 64;
4 
5     %80 time samples adding the cyclic prefix
6     N2 = 80;
7 
8     %bits per subcarrier
9     data_mapped = wlanConstellationMap(data_packet_interleaved, ...
```

```
        encoded_symbol_size/48);
10
11    %now, map the symbols on their respective subcarriers and add ...
        pilot tones.
12    data_sequenced_pos = zeros(N/2, 1);
13    data_sequenced_neg = zeros(N/2, 1);
14
15    %negative half of ifft
16    data_sequenced_neg(1:6) = 0;
17    data_sequenced_neg(7:11) = data_mapped(1:5);
18    data_sequenced_neg(12) = 1*p_phase;
19    data_sequenced_neg(13:25) = data_mapped(6:18);
20    data_sequenced_neg(26) = 1*p_phase;
21    data_sequenced_neg(27:32) = data_mapped(19:24);
22
23    %positive half of ifft
24
25    data_sequenced_pos(1) = 0;
26    data_sequenced_pos(2:7) = data_mapped(25:30);
27    data_sequenced_pos(8) = 1*p_phase;
28    data_sequenced_pos(9:21) = data_mapped(31:43);
29    data_sequenced_pos(22) = -1*p_phase;
30    data_sequenced_pos(23:27) = data_mapped(44:48);
31    data_sequenced_pos(28:32) = 0;
32
33    data_sequenced = [data_sequenced_pos; data_sequenced_neg];
34
35    data_in_time = ifft(data_sequenced, N);
36
37    %add another sample for windowing with next symbol.
38    waveform = zeros(1,N2 + 1);
39    %add cyclic prefix
40
41    %add the windowing as WLAN Toolbox and IEEE 802.11a standard ...
        suggest
42    waveform(1) = data_in_time(49)/2;
43    waveform(2:16) = data_in_time(50:64);
44    waveform(17:80) = data_in_time;
45    waveform(81) = data_in_time(1)/2;
46
47 end
```

In the process of transforming a bit block into an analog waveform, the discrete version of the Fourier Transform, the Discrete Fourier Transform (DFT), is used. As explained before, its inverse, the Inverse Discrete Fourier Transform (IDFT) is used to transform the symbol block derived from the bitstream into the OFDM symbol that is sent through the radio interface. Fortunately, DFT and its inverse share many properties with their analog counterpart, the Fourier Transform. One of them is frequency scaling, relating the temporal signal length with its transform signal length in the frequency domain, and vice versa. Signal expansion in the frequency domain leads to signal contraction in the time domain. The complete proof of this

property for DFT can be found in [88]. Consequently, the generation of a wide constant pulse in the frequency domain leads to a very narrow waveform in the time domain, being most of its energy contained on a very narrow temporal region, i.e., a peak. Therefore, this property can be useful for the generation of amplitude-based waveforms. For example, a signal containing a very pronounced peak, if low-pass filtered, resembles an OOK “0” symbol, albeit, with a lower extinction ratio.

With this approach, several input sequences ***generate_symbol*** were trialed. Nearly constant subcarrier blocks were used to produce OFDM symbols that, after the application of the IDFT, had very distinctive regions, with pronounced peaks and valleys. With this type of waveform, the presence of one peak concentrates the symbol energy on a very narrow temporal interval. Besides the “peak” region, this waveform also features a larger “valley” section, conferring it a high Peak-to-Average Power Ratio (PAPR). Conversely, it also possible to input sequences to the IDFT block that result in symbols with nearly constant amplitude. Those, by being flatter, present low PAPR. Using these two types of waveforms, a binary constellation can be derived. The constellation comprises one symbol with very high PAPR, henceforth referred to as *Peak Symbol*, and another symbol with a very low PAPR referred to as *Flat Symbol*. An ideal Peak Symbol candidate, with no contribution from pilot tone addition, and no cyclic prefix, is shown in Fig.3.2.

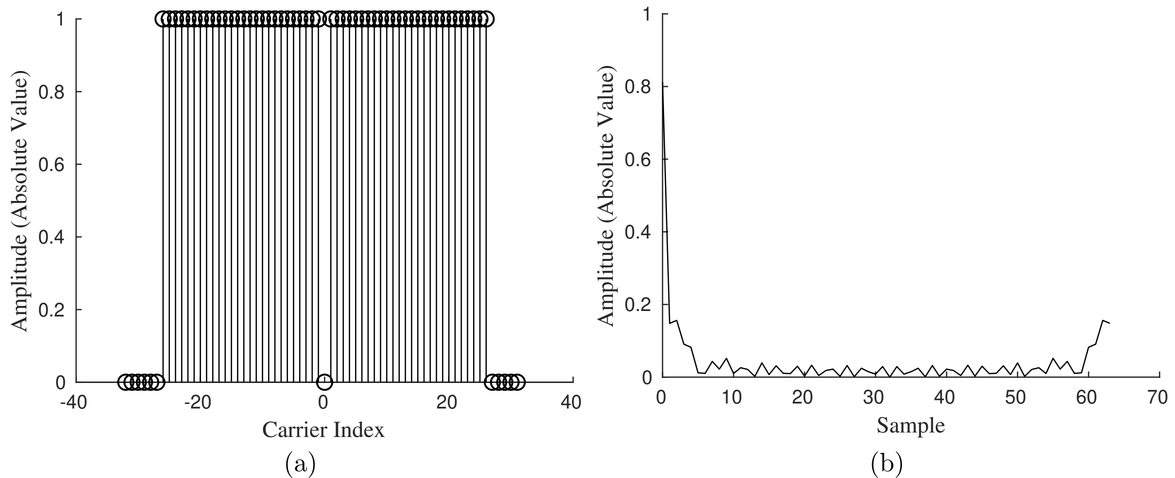


Figure 3.2: Amplitude a Peak Symbol candidate, in frequency (a) and time (b) domains. Note that this representation does not include the cyclic prefix. Hence, the time domain representation has only 64 samples.

The symbols of this modulation can then be easily discriminated with low-complexity non-coherent receivers. One approach is to discriminate them using their peak height with a non-linear peak detector circuit. This way, the higher that the PAPR is, the more separated that the symbols become in the constellation. Another approach is to receive these symbols as if they were ideal OOK symbols, using a simple OOK detector with low-pass filtering. Using this detector, the low-pass filter produces a

pseudo-OOK modulation using the Peak and Flat Symbols. This way, Peak Symbols resemble imperfect OOK "0" symbols and Flat Symbols, "1" OOK symbols. This is shown in a pair of examples for the Peak and Flat Symbols in Fig.3.3 Nonetheless, this approach suffers from a reduced extinction ratio, since the "0" symbols obtained with this pseudo-OOK approach are far from perfect. However, it is compatible with other OOK receivers, such as those used in IEEE 802.11ba.

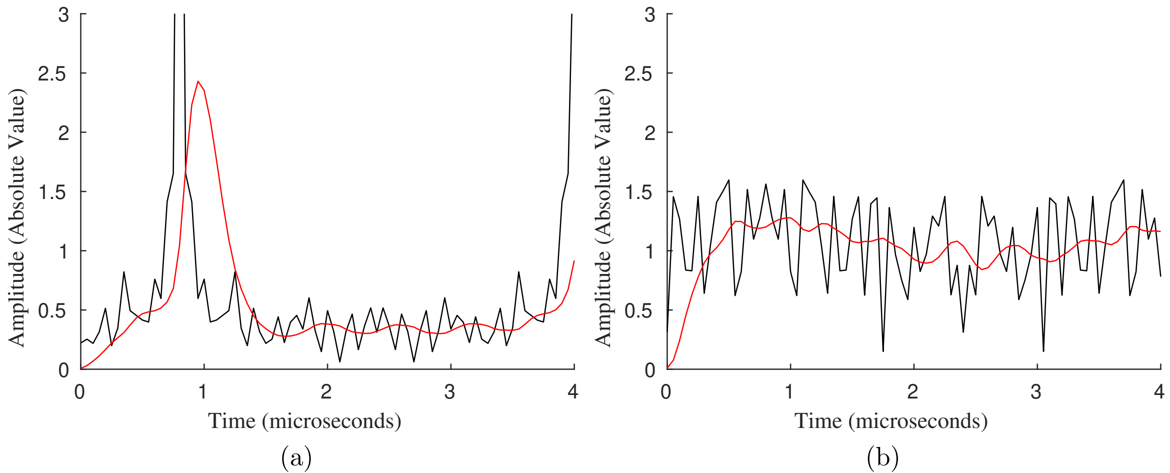


Figure 3.3: Detail of a pair of Peak and Flat Symbol candidates in time domain, shown in (a) and (b), respectively. This representation was generated with the ***generate_symbol*** function. Results of low-pass filtering with a second order Butterworth filter with a cutoff of 250 kHz are outlined in red.

3.3 Generation of Peak/Flat Symbols using an IEEE 802.11a/g OFDM PHY

The WuTx implementation can only become compatible with standard-compliant transceivers if no hardware changes are required. Hence, an implementation done using exclusively software-level access in standard-compliant transceivers is required. The generation of Peak-Flat is proposed by the means of inputting a bitstream to the transmission chain of a standard-compliant IEEE 802.11a/g OFDM PHY. This is, by no means a trivial challenge. Following, Subsections 3.3.1 and 3.3.2 introduce the methodology used for the generation of Peak and Flat Symbols using data bits from higher layers on the IEEE 802.11 OFDM PHY, particularizing for the IEEE 802.11g transmitter used as PoC.

3.3.1 Generation of the Peak Symbol

An Idealized Peak Symbol with IEEE 802.11a/g

According to the previous design principle to obtain an optimal Peak Symbol, one must use the widest possible constant waveform at the input of the IDFT block. With the IEEE 802.11 OFDM PHY, the closest implementation is a uniform symbol block, created by assigning the same value to all the available OFDM subcarriers. IEEE 802.11g specifies that values for the subcarrier with index 0 as well as the subcarriers that belong to the ranges going from -27 to -32 and from 26 to 31 must set to “0” [84].

Then, according to IEEE 802.11a/g, the resulting symbol block is described mathematically in (3.1), where k represents the subcarrier index, and a, an arbitrary complex value. Such a symbol is shown in Fig.3.2a. Nonetheless, this is not feasible as the value of a minority of the subcarriers, i.e., the pilot tones, cannot be controlled directly by the bits input to the IEEE 802.11 OFDM PHY. The effect of these will be addressed later, in Section 3.1). Thus, such a symbol as one described in (3.1) cannot be implemented. However, it is possible to obtain close-enough implementations of this ideal Peak Symbol, despite pilot tone interference.

$$X[k] = \begin{cases} 0 & k < -26 \\ a & -26 \leq k < 0 \\ 0 & k = 0 \\ a & 0 < k < 26 \\ 0 & k \geq 26 \end{cases} \tag{3.1}$$

In the time domain, the ideal Peak Symbol adopts the expression of (3.2), where n corresponds to the sample index, which goes from 0 to 63. A plot based on (3.2) is shown in Fig.3.2b.

The IEEE 802.11a/g OFDM PHY adds a guard interval (also called or cyclic prefix) of 16 samples to the OFDM symbol. Being its values a repetition of the last 16 samples of the OFDM symbol and increasing the length of the time-domain OFDM symbol to 80 samples. The introduction of this guard interval, for the time-domain OFDM symbol described by (3.2) is showcased in Fig.3.4.

$$x[n] = \frac{1}{64} \frac{sin(2\pi \frac{53}{64} n)}{sin(2\pi \frac{1}{64} n)} - 1 \,, \quad 0 \leq n \leq 63 \tag{3.2}$$

As shown by (3.1), the Peak Symbol can be produced by any uniform symbol block independently of the complex value used to initialize its 52 samples. Therefore,

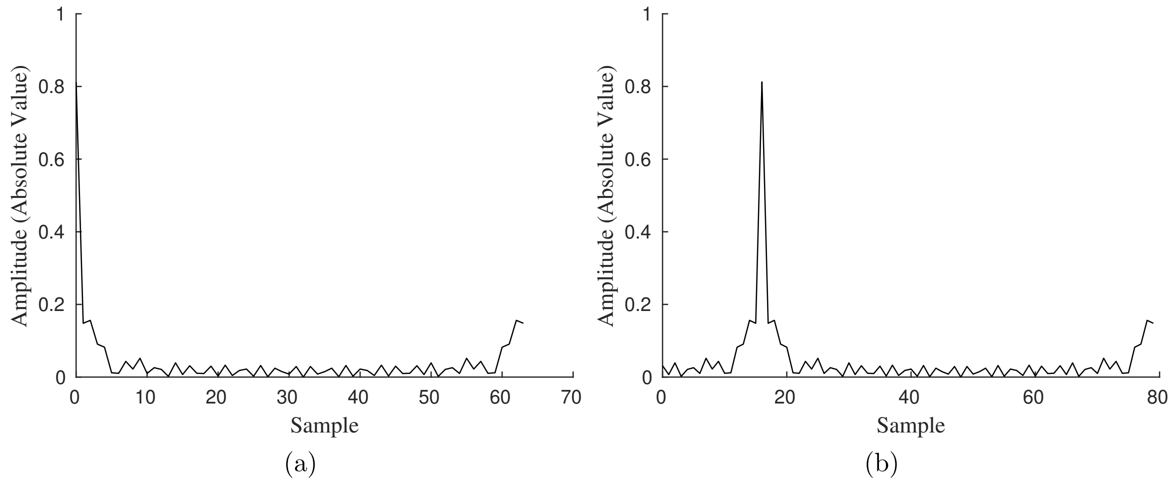


Figure 3.4: Comparison of the same symbol before (a) and after (b) guard interval insertion.

all possible constellation symbol values supported by IEEE 802.11 OFDM PHY produce a waveform with the same shape and PAPR, although, with different scale factors.

Nonetheless, the implementation of the Peak Symbol devised in this section using a complete IEEE 802.11a/g OFDM PHY still requires a method to obtain a suitable bit sequence. Such a sequence, after being manipulated by the blocks found before the Symbol Mapping block, produces a uniform symbol block at the input of the IDFT block. This way, the rest of the section presents the solutions taken to build the Peak Symbol from the bitstream at the entry of the IEEE 802.11g OFDM PHY data path.

Addressing the effect of the Scrambler block

The first block of the IEEE 802.11g OFDM PHY block structure is the Scrambler. Its purpose, to randomize the input bitstream to avoid long runs of "0" or "1". The goal of this is to minimize both the average PAPR of OFDM symbols and the DC bias at the reception.

The IEEE 802.11 scrambler is implemented using an LFSR with the characteristic polynomial presented in (5.4). Its graphical representation is also shown in Fig.3.5, where S_{in} is the input bitstream, and S_{out} the scrambled bitstream.

$$S[x] = 1 + x^4 + x^7 \tag{3.3}$$

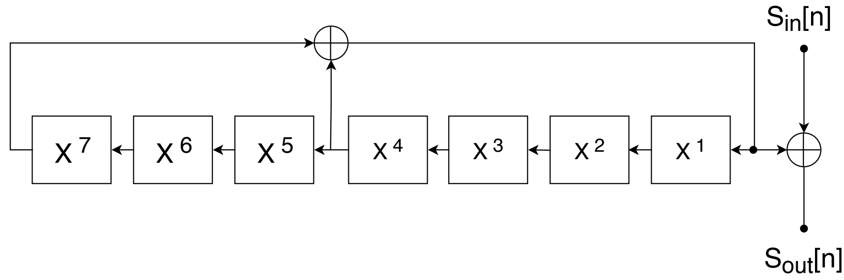


Figure 3.5: LFSR representation of the scrambler used in IEEE 802.11 OFDM PHY[84]

The specification determines that the seed used to initialize the LFSR should change at the start of each frame transmission [84]. Additionally, there is no requirement for IEEE 802.11 OFDM PHY implementations to provide read or write access to the scrambler seed. Therefore, the seed is a priori an unknown value to the users of the transceivers.

Nonetheless, in some cases, the scrambler seed can be determined using information external to the transceiver as pointed out by [89]. Various transmitters use either a constant seed to initialize the LFSR or one that can be predicted using data external to the IEEE 802.11 OFDM PHY, such as the MAC address of the transceiver and the number of transmitted bits.

To produce the Peak-Flat modulation, a controlled sequence is required at the output of the scrambler. Hence, the randomization introduced by the scrambler must be compensated. If the initialization seed is known, then, the effect of the Scrambler can be canceled by scrambling the input bitstream in advance. This can be achieved by pre-scrambling the input data using the same bit sequence that the Scrambler will generate. The proof of this uncomplicated solution can be seen in (5.5)-(3.7). Where $x[n]$ consists of the intended bit sequence at the output of the scrambler, $y[n]$ the pseudo-random bit sequence generated by the Scrambler, $s_{in}[n]$ the sequence at the input of the Scrambler, and finally, $s_{out}[n]$ the output sequence of the Scrambler.

$$s_{in}[n] = y[n] \oplus x[n] \tag{3.4}$$

$$s_{out}[n] = s_{in}[n] \oplus y[n] \tag{3.5}$$

$$s_{out}[n] = x[n] \oplus y[n] \oplus y[n] \tag{3.6}$$

$$s_{out}[n] = x[n] \tag{3.7}$$

Therefore, the effect of the scrambler in the input sequence can be compensated for, but only if the Scrambler initialization seed is known in advance and the input

sequence is pre-scrambled. This way, any input bit sequence can cross the scrambler unmodified, and, therefore, advance one step closer to the Symbol Mapping Block.

Addressing the effect of the Convolutional Coder block

The Convolutional Coder block protects the incoming data from random errors with a Forward Error Correcting (FEC) code. The convolutional coder used by IEEE 802.11 is parametrized by the following variables, where r represents the coding rate, k the window length, being g_0 and g_1 are the generator polynomials. All these are defined in Table 3.1.

Table 3.1: Parameters of the convolutional coder used in IEEE 802.11 OFDM PHY

Symbol	Value
r	2
k	7
g_0	133_8
g_1	171_8

The coder, as shown in Fig.3.6, generates two output bits for every input bit and has a memory of 6 bits. The coding rate implemented is 1/2 but additional coding rates of 2/3 and 3/4 can be obtained from the same by using Puncturing. This technique removes bits from the output of the coder, which are introduced again in the receiver, before the decoder block, as "0" values.

The use of the convolutional coder is required with IEEE 802.11a/g. Nonetheless, in the following releases, the presence of a convolutional coder on the device is still a requirement. This way, the solution presented here is compatible with later releases, such as IEEE 802.11n/ac.

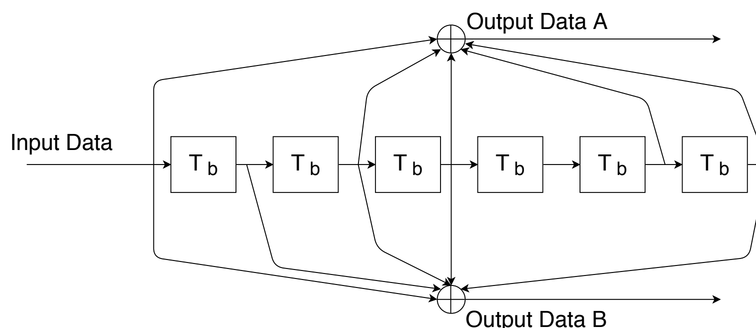


Figure 3.6: Convolutional Coder used in IEEE 802.11 PHY. Note that for each bit input, two outputs, Data A and Data B are generated. Each T_b block represents a one bit delay[84].

If the goal is to obtain a controlled sequence of bits at the input of the symbol

mapping stage, the Convolutional Coder is the most challenging block to work with. Not every bit sequence is possible at the output due to the effect of coding and the redundancies added. However, both uniform bit sequences (i.e, one with "0" and another with "1") can produce a Peak Symbol at the symbol mapping stage and are not modified by the addition of coding. They still appear uniform at the output, only that their length is multiplied by the inverse of the coding rate used.

For example, the uniform input sequence of "0" bits, results in another uniform output sequence of "0" bits, but grown, i.e., a sequence of 50 "0" at the input, coded with a rate of 1/2, results in a sequence of 100 "0" at the output. This is a consequence of the exclusive use of the coder only implements linear operations, i.e., XORs. As seen in (3.8)-(3.11), for each "0" at the input, two "0" are obtained at the output.

$$s[n] = \{0, 0, 0, 0, 0, 0\} \tag{3.8}$$

$$x = 0 \tag{3.9}$$

$$y_a = x \oplus s[2] \oplus s[3] \oplus s[5] \oplus s[6] = 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 0 \tag{3.10}$$

$$y_b = x \oplus s[1] \oplus s[2] \oplus s[3] \oplus s[6] = 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 0 \tag{3.11}$$

where x represents the input bit, $s[n]$, the 6-bit wide scrambler state, y_a the Data A output of the coder, obtained by applying g_0 to x, and, finally, y_b the Data B output, generated in a similar way by g_1. Both Data A and Data B correspond to the outputs displayed in Fig.3.6.

Consequently, the sequence with uniform "1" also appears unmodified at the output, as long as the coder state bits are also set all to "1". This is a consequence of the odd number of XOR operations in both branches, which is five. An odd number of XOR operation with the same value, here "1". This can be seen in (3.12)-(3.15), using the same variable notation that was used in (3.8)-(3.11). This condition holds in IEEE 802.11a/g/n/ac, which include the odd-numbered generator polynomials shown here.

$$s[n] = \{1, 1, 1, 1, 1, 1\} \tag{3.12}$$

$$x = 0 \tag{3.13}$$

$$
\begin{aligned}
y_a &= x \oplus s[2] \oplus s[3] \oplus s[5] \oplus s[6] \\
&= 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \\
&= ((((1 \oplus 1) \oplus 1) \oplus 1) \oplus 1) \\
&= (((0 \oplus 1) \oplus 1) \oplus 1) \\
&= ((1 \oplus 1) \oplus 1) \\
&= (0 \oplus 1) = 1
\end{aligned} \tag{3.14}
$$

$$
y_b = x \oplus s[1] \oplus s[2] \oplus s[3] \oplus s[6] = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \tag{3.15}
$$

Nonetheless, the Convolutional Coder has a state. This state modifies subsequent bits depending on the values of the previous ones. Thus, its effects affect the transitions between Peak and Flat Symbols, where the input sequence goes from uniform values to non-uniform values. This issue is addressed in Section 3.3.2, which introduces the Flat Symbol generation procedure, as well as the measures taken to avoid this issue.

Therefore, in conclusion, the two uniform binary sequences can cross the coder unmodified, i.e., all “1” and all “0” sequences. These sequences are of interest since they produce uniform symbol blocks at the input of the IDFT block, and those allow the generation of Peak Symbols. Then, uniform bit sequences, that are pre-scrambled can cross unmodified both the Scrambler and the Convolutional Coder blocks. Next section, the effect of the Interleaver will be addressed.

Addressing the effect of the Interleaver block

The purpose of an interleaver is to distribute bit errors uniformly, therefore maximizing the effectiveness of FEC coding. The Interleaver block in the IEEE 802.11a/g OFDM PHY separates the incoming bitstream on blocks of an integer number of bits. The bits from these blocks will be jointly encoded in the same OFDM symbol. The length of these bit blocks depends only on the modulation used on the Symbol Mapping block. Afterward, the Interleaver shuffles the bits in the block in two steps. First, it distributes the adjacent bits within the block so that they will be encoded into non-adjacent subcarriers. Thus, protecting the payload against error bursts caused by narrow-band fading, which cannot be recovered by FEC. Second, the bits assigned to each subcarrier are distributed among the less and more significant bits of the QAM constellation symbol. This step only makes sense when subcarriers bear at least 4 bits, consequently, it is only applied when QAM symbols are employed to map subcarriers. This last mechanism randomizes the effect of the greater systematic error rate that the least significant bits of QAM symbols suffer.

Returning to Peak Symbol generation, the effect of the Interleaver block can be canceled by pre-interleaving the sequence at the data input. that is, moving bits so

they are shuffled into their intended positions by the Interleaver. Nonetheless, this is not necessary for the two uniform "0" and "1" sequences proposed before. As they are uniform, they are, by definition, unaffected by any transposition of their elements.

Effect of the Symbol Mapping block

The Symbol Mapping block terminates the section of the processing chain that works directly with binary values. It does so by converting the bit blocks output by the Interleaver into symbol blocks with complex values. The symbols in the block are also referred to as subcarriers since each of those corresponds to one of the various tones that compose an OFDM symbol in the frequency domain. In IEEE 802.11a/g, each symbol block comprises 48 complex symbol values [84], a number that remains constant over all data rates supported. Nonetheless, subsequent amendments can support a higher number of subcarriers per OFDM symbol. For example, up to 108 symbols in IEEE 802.11n [90], 468 in IEEE 802.11ac [91] and 980 in IEEE 802.11ax [92].

To convert from bits to symbols, this block uses one of the constellations available in IEEE 802.11a/g OFDM PHY: BPSK, QPSK, 16-QAM, 64-QAM. Nonetheless, in posterior releases, these 256-QAM and 1024-QAM were added. The first four constellations are supported in IEEE 802.11a/g/n while, 256-QAM, is only supported from IEEE 802.11ac and 1024 QAM is only supported by IEEE 802.11ax.

As determined by the results of Section 3.3.1, Peak Symbols can be generated by uniform symbol blocks. Moreover, the shape of the waveform generated by those is the same, regardless of which symbol it is used. Additionally, as shown in the previous sections, pre-scrambled uniform bit sequences can cross the previous blocks without suffering modifications. Those produce uniform bit blocks at the output of the Interleaver Block, which the Symbol Mapped Block encodes into uniform symbol blocks. Nonetheless, since uniform bit blocks only contain bits of equal value, they can only be encoded into symbol blocks using a subset of symbols that represent exclusively either "0" or "1" bits. Those change according to the different constellations used for the data rates OFDM PHY defined in the IEEE 802.11a/g specification. The symbol values that can be generated by uniform bit blocks are found in Table 3.2.

Effect of pilot tone, and cyclic prefix addition

After the bit blocks are mapped into symbol blocks, pilot tones are inserted by the Tone Mapping and Pilot Tone Insertion block. This adds 4 pilot symbols to the existing symbol block, increasing its length from 48 to 52 subcarriers. Those are

Table 3.2: Possible symbol values derived from uniform bit blocks

Modulation	Symbol value	Coded Bits
BPSK	-1	"0"
BPSK	$+1$	"1"
QPSK	$+1+1j$	"11"
QPSK	$-1-1j$	"00"
16-QAM	$1+1j$	"1111"
16-QAM	$-3-3j$	"0000"
64-QAM	$3+3j$	"111111"
64-QAM	$-7-7j$	"000000"

added to indexes 7, 21, -7, and -21 of the symbol block. The amplitude of pilot tones is fixed at 1, regardless of the constellation used to map the subcarriers, nonetheless, their phase can vary. The specification fixes that three of the tones share the same phase, and only one must present a phase π radians opposite. According to these principles, the contribution to the symbol block is shown in sequence $P_{-26,26}$ from (3.16).

$$\begin{aligned} P_{-26,25} = \{&0,0,0,0,0,1,0,0,0,0,0,0,0,\\ &0,0,0,0,0,0,1,0,0,0,0,0,0,\\ &0,0,0,0,0,0,0,1,0,0,0,0,0,\\ &0,0,0,0,0,0,0,0,-1,0,0,0,\\ &0,0\} \end{aligned} \tag{3.16}$$

Moreover, the phase of the pilot tones is switched after each OFDM symbol. The switching of the pilot tone phase occurs according to a pseudo-random sequence that is generated by an LFSR equal to the one used by the Scrambler (see Section 3.3.1). When the LFSR outputs a "0", the phase of the pilot tones is switched by π radians (i.e, the values of (3.16) are multiplied by "-1"), otherwise, it is left unchanged [84]. According to this dynamic phase component, two pilot tone combinations alternate in a pseudo-random pattern. The first, comprises three tones with the "+1" value and one tone with the "-1", thus, with mostly positive values; and the second comprises three tones with the "-1" value, and one the "+1" value, therefore, with mostly negative values.

The addition of these pilot tones breaks the uniformity of the symbol block, which, in turn, alters the resulting OFDM symbol, which no longer corresponds to the ideal representation presented in 3.3.1. Consequently, as the symbol block is no longer perfectly uniform, its PAPR is degraded with respect to the ideal Peak Symbol. The magnitude of this effect depends on the disparity between the symbol value of the subcarriers conforming the symbol block and the pilot tones. e.g., if the value of the

majority of the pilot tones is “-1”, the PAPR of the resulting Peak Symbol will be higher if the rest of its subcarriers are also “-1”, and lower if they are “+1”.

To optimize the performance of the WuTx, it is important to determine which symbol block values provided better PAPR, and accordingly, less degraded Peak Symbols concerning the ideal. For this purpose, the MALAB function described in Section 3.2.2, ***generate_symbols*** was used to generate Peak Symbols with all the subcarrier values generated with uniform bit blocks (see Table 3.2) and pilot tone combinations. The result of this study, which includes the addition of the cyclic prefix, can be found in Table 3.3. To represent the effect of pilot tones, the average PAPR for a given subcarrier symbol value is derived by averaging the PAPR of the two possible Peak Symbols, one with each pilot tone combination (in-phase and on opposite phase).

Table 3.3: PAPR obtained for each symbol and pilot tone phase combination

Modulation	Symbol value	Negative pilot tones PAPR(dB)	Positive pilot tones PAPR(dB)	Average PAPR(dB)
BPSK	-1	17.39	16.59	16.99
BPSK	$+1$	16.59	17.39	16.99
QPSK	$1+1j$	16.76	17.24	17.00
QPSK	$-1-1j$	17.24	16.76	17.00
16-QAM	$1+1j$	15.17	16.27	15.72
16-QAM	$-3-3j$	17.35	16.99	17.17
64-QAM	$3+3j$	16.16	16.90	16.53
64-QAM	$-7-7j$	17.38	17.05	17.22

According to the results, the best subcarrier symbol value, with an average PAPR of 17.22 dB, is 64-QAM “-7 -7j”, which encodes 6 “0” bits. The BPSK “-1” and “+1” values, representing “0” and “1” logic values, present the second-best results. Nonetheless, BPSK symbols present the best PAPR results when the pilot tones are in phase, with a PAPR of 17.39 dB. In contrast, Peak Symbols built using BPSK subcarrier values are on average lower when the pilot tones are on the opposite phase. Consequently, the average PAPR for both BPSK “+1” and “-1” is 16.99, approximately 0,2 dB lower than the average PAPR obtained by 64-QAM “-7 -7j”. Below, in Fig.3.7 the waveforms generated by the Peak Symbols corresponding to subcarrier symbol values “-7 - 7j” and “-1” are shown.

Bit sequences used to generate the Peak Symbol

Previous Sections propose that uniform bit sequences, after being pre-scrambled, can go through the blocks of the IEEE 802.11 OFDM PHY data path and produce

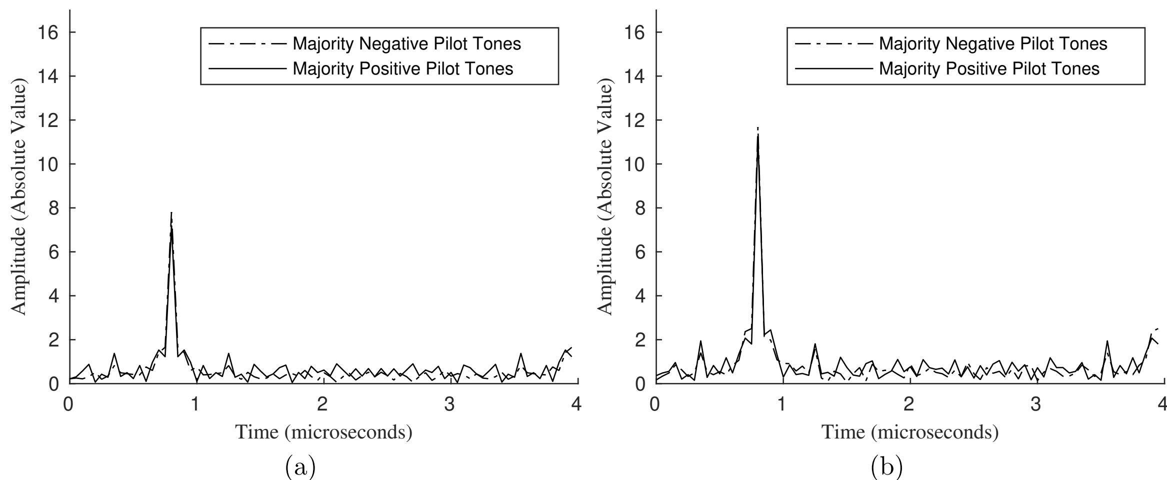


Figure 3.7: Amplitude of best case non-ideal Peak Symbols for the “-7 -7j” subcarrier value (a) and “-1” subcarrier value (b).

uniform symbol blocks. These sequences can generate an array of different Peak Symbols, depending on the data rate used by the IEEE 802.11 OFDM PHY.

To summarize, the requirements that these Peak Symbol bit input sequences must fulfill are condensed in the following statements.

1. They need to be pre-scrambled with the same scrambler sequence generated by the Scrambler block.
2. The bit sequence at the output of the scrambler must be uniform, i.e, all values must be either “0” or “1”.
3. The length of the sequence must be equal to the OFDM symbol payload length.
4. The sequence must be aligned with OFDM symbol boundaries to be encoded jointly in the same OFDM symbol.

Using these conditions, the input bit sequence necessary to generate a Peak Symbol can be obtained on any of the supported data rates of the IEEE 802.11a/g OFDM PHY (i.e, any of those shown in 3.2). For example, the sequence used at the input of the OFDM PHY to obtain the Peak Symbol that corresponds to the 6 Mbps data rate, and the “-1” constellation value. This sequence is a bit sequence of 24 “0” bits, that needs to be pre-scrambled before being input to the IEEE 802.11a/g OFDM PHY. The “0” bits are mapped to the “-1” symbol value and the length of 24 bits is equal to the OFDM symbol payload used for the 6 Mbps data rate. Nonetheless, the values used for the pre-scrambling procedure will vary according to the seed used by the Scrambler block.

Therefore, this Section presents a procedure to determine bit sequences that generate Peak Symbols using software-level access compatible with standard-compliant IEEE 802.11a/g transmitters. With this, the first of the two symbols proposed by the Peak-Flat modulation can be generated with exclusively software level access on standard-compliant IEEE 802.11a/g transmitters. Nonetheless, a procedure to generate Flat Symbols is still to be defined.

3.3.2 Generation of Flat Symbols

To complement the Peak Symbol, a Flat Symbol, with the minimum possible PAPR must be designed. This section presents the procedure used to generate the Flat Symbol and proposes one example of Flat Symbol generation.

Relation between the Peak and the matching Flat Symbol

As proposed in Section 3.2.2, the Peak Symbol requires a counterpart, with low PAPR, i.e, as flat as possible. This allows this symbol to be distinguished from the Peak Symbol, either by its lower higher peak-value, or, after low-pass filtering, by its higher average amplitude. Additionally, the Flat Symbol must be constructed using the same modulation and coding scheme as its complementary Peak Symbol. IEEE 802.11g/a OFDM PHY frames combining two different data rates after the SIGNAL field are not supported by standard-compliant transmitters [84]. Consequently, a different Flat Symbol needs to be matched to each of the Peak Symbols presented in Section 3.3.1.

To preserve compatibility with standard-compliant IEEE 802.11a/g transmitters, in the same way as before, the Peak Symbol must be built using the input bitstream to the IEEE 802.11 OFDM PHY. Therefore, the Flat Symbol will be provided by the bit sequence that generates the OFDM symbol with the lowest PAPR.

Moreover, the use of the Peak Symbol cannot interfere with subsequent Flat Symbols and vice versa. The convolutional coder has a six-sample memory, which can modify the bits that will be coded into the next symbol, depending on previous values. For example, if the bit block that generates the Flat Symbol contained, for example, any "1" bits on its last 6 samples, it could introduce unexpected values on the bit block coding the next Peak or Flat Symbol. Thus, a solution is to choose the last 6 values of the bit block generating the Flat Symbol, so the state of the Convolutional Coder can be reset to a given value. For this purpose, the last six bits of the bit block sequence have to be either "0" or "1". Thus, to eliminate this inter-symbol interference, the value of these last bits needs is chosen to equal to the value of the uniform bit sequence used to generate the corresponding Peak Symbol. e.g., if the corresponding Peak Symbol is generated by a block of "0" bits, then, the last six

bits of its matching Flat Symbol need to be chosen as “0”.

Therefore, with this constraint, the search space for an optimal Flat Symbol is determined by its matched Peak Symbol. Thus, it includes all bit sequences equal in length to the OFDM symbol payload for a given data rate. Additionally, all these sequences must end with six “1” or six “0” bits, being their value matched to the uniform sequence that generates the corresponding Peak Symbol.

Example of Flat Symbol generation

With the constraints mentioned above, finding the optimum Flat Symbol becomes an optimization problem, and its complexity depends on the data rate used. The size of the search space is defined by the size of the bit block, and this is, in turn, defined by the data rate used.

Thus, the ***generate_symbol*** developed for Section 3.2.2 was extended to include the complete IEEE 802.11a/g processing chain, so the PAPR of all the sequences in the problem space can be obtained. The code for this updated version, which uses MATLAB WLAN Toolkit functions to emulate the first 3 blocks can be found below.

```
1 function waveform = generate_symbol(data_packet, Bits_per_Symbol, ...
     code_ratio, p_phase)
2     %64 samples for IFFT
3     N = 64;
4
5     %80 time samples adding the cyclic prefix
6     N2 = 80;
7
8     data_packet_coded = wlanBCCEncode(data_packet, code_ratio);
9
10    encoded_symbol_size = Bits_per_Symbol/eval(code_ratio);
11    data_packet_interleaved = ...
          wlanBCCInterleave(data_packet_coded,'Non-HT',
12 encoded_symbol_size);
13
14    %bits per subcarrier
15    data_mapped = wlanConstellationMap(data_packet_interleaved, ...
          encoded_symbol_size/48);
16
17    %now, map the symbols on their respective subcarriers and add pilot
18    %tones.
19
20    data_sequenced_pos = zeros(N/2, 1);
21    data_sequenced_neg = zeros(N/2, 1);
22
23    %negative half of ifft
24    data_sequenced_neg(1:6) = 0;
```

```
25      data_sequenced_neg(7:11) = data_mapped(1:5);
26      data_sequenced_neg(12) = 1*p_phase;
27      data_sequenced_neg(13:25) = data_mapped(6:18);
28      data_sequenced_neg(26) = 1*p_phase;
29      data_sequenced_neg(27:32) = data_mapped(19:24);
30
31      %positive half of ifft
32
33      data_sequenced_pos(1) = 0;
34      data_sequenced_pos(2:7) = data_mapped(25:30);
35      data_sequenced_pos(8) = 1*p_phase;
36      data_sequenced_pos(9:21) = data_mapped(31:43);
37      data_sequenced_pos(22) = -1*p_phase;
38      data_sequenced_pos(23:27) = data_mapped(44:48);
39      data_sequenced_pos(28:32) = 0;
40
41      data_sequenced = [data_sequenced_pos; data_sequenced_neg];
42
43      data_in_time = ifft(data_sequenced, N);
44
45      %add another sample for windowing with next symbol.
46      waveform = zeros(1,N2 + 1);
47      %add cyclic prefix
48
49      waveform(1) = data_in_time(49)/2;
50      waveform(2:16) = data_in_time(50:64);
51      waveform(17:80) = data_in_time;
52      waveform(81) = data_in_time(1)/2;
53
54      %use the same scale factor as waveform generator
55      waveform = waveform.*10;
56
57  end
```

Fortunately, this naive solution can be applied when the problem space is small, such as in the low data rates defined by IEEE 802.11a/g. For example, the bit sequence that generates the Flat Symbol for the Peak Symbol examples proposed in Section 3.3.1 can be found this way.

Thus, let us find the optimal Flat Symbol for the Peak Symbol defined by a data rate of 6 Mbps, with the BPSK symbol value "-1". The bit block length for this modulation and coding rate is 24 bits, consequently, the search space for the optimal Flat Symbol comprises all sequences of 24 bits that end with 6 consecutive "0" bits. That adds up to a total of 2^{18} possible sequences. This is still a tractable number of sequences to be explored exhaustively with the OFDM PHY simulator previously developed. This way, the PAPR of the OFDM symbols generated by every bit sequence in the problem space was calculated to obtain the optimal Flat Symbol. This procedure took close to 3 minutes, using MATLAB 2018 on a desktop PC, with an Intel i7 2600.

The resulting Flat Symbol, found for the 6 Mbps data rate and “BPSK” “-1” symbol, can be generated by the bit sequence $s[n]$ from (3.17). It presents an average PAPR of 3.09 dB, using both pilot tone polarities. It is displayed in Fig.3.8.

$$
\begin{aligned}
s[n] = \{&1,1,0,0,1,1,1,0,1,0,1,1,\\
&0,1,0,1,0,0,0,0,0,0,0,0\}
\end{aligned}
\tag{3.17}
$$

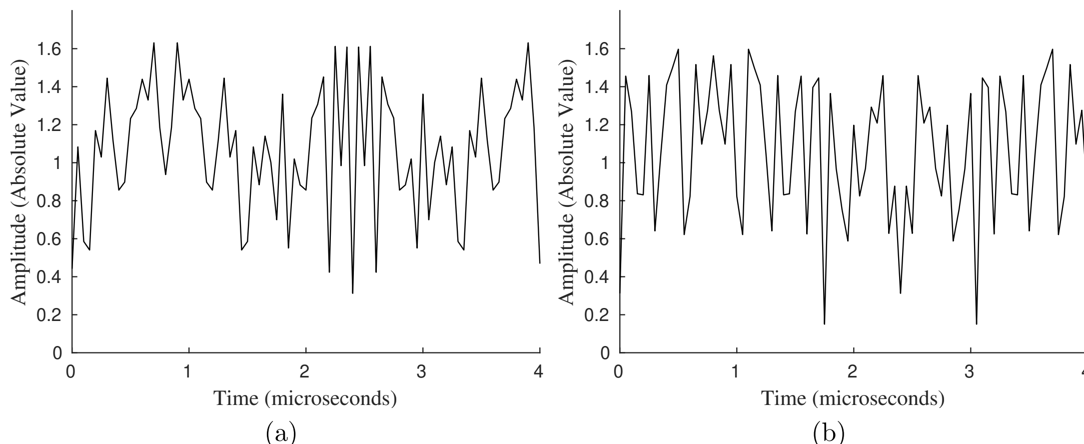


Figure 3.8: Amplitude of the Flat Symbol generated by the binary sequence presented in (3.17) for mostly positive pilot tones (a) and mostly negative pilot tones (b). Note that its maximum amplitude is significantly lower than the peak of its respective Peak Symbol presented in Fig.3.7b.

For other bit rates that employ longer bit block lengths, the total number of sequences is not tractable with this naive optimization approach. However, sub-optimal symbol values, with PAPR lower than 4 dB, were found for those modulations by searching only 2^{20} sequences.

For example, a sub-optimal Flat Symbol for 54 Mbps and the “-7 -7j” symbol value was obtained after exploring 2^{20} random sequences. Its averaged PAPR is 3.90 dB, and it is displayed in Fig.3.9. The runtime of the search procedure took less than 20 min.

3.4 Simulation of the Wake-up Transmitter

Now that the procedure to obtain the bit sequences that generate both the Peak and Flat Symbols for a given modulation have been found, they can be applied to the implementation of Peak-Flat-based WuTx. To evaluate this implementation, a PoC WuTx based on the MATLAB WLAN toolbox is proposed. Given an entry

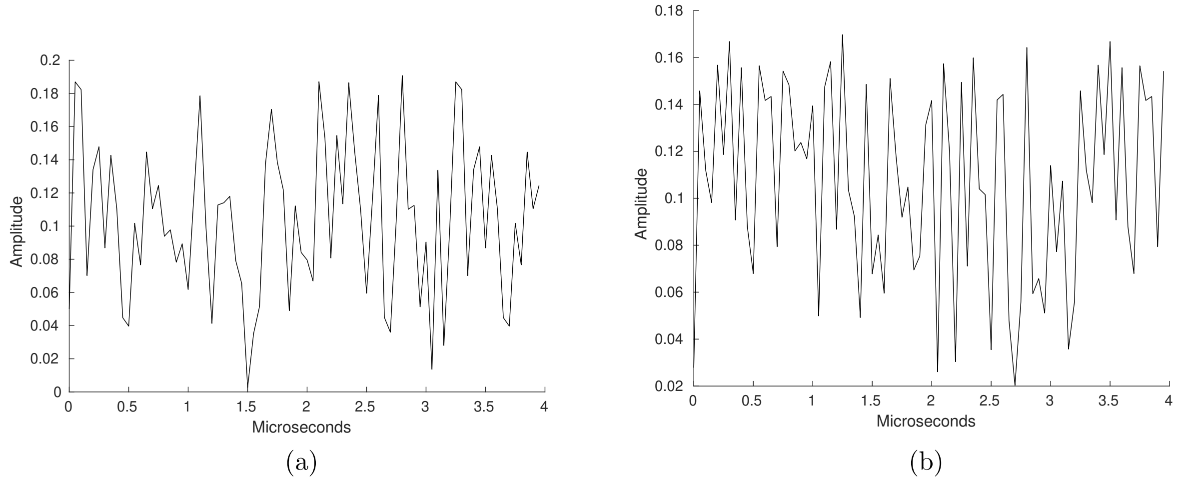


Figure 3.9: Amplitude of a Flat Symbol designed for 54 Mbps with the symbol "-7 -7j", displayed with mostly positive pilot tones (a) and mostly negative pilot tones (b).

bitstream, the MATLAB WLAN Toolbox implements functions that emulate the entire IEEE 802.11a/g OFDM PHY. Moreover, it allows setting the initial state of the scrambler, so the prerequisite detailed in Section 3.3.1 to use the Peak-Flat modulation is met. Moreover, the evaluation of the WuTx can be used with simulated WuRx for the evaluation of the complete system.

3.4.1 Implementation of the WuTx PoC

For this PoC, the following parameters are used:

- BPSK as modulation.
- "-1" symbol value.
- 6 Mbps data rate.

To match OOK reception, the Peak Symbol is set to encode the "0" logic value and the Flat Symbol is used to encode the "1" logic value. Using one of the standard MATLAB WLAN Toolkit functions, the bitstream that generates the WuS is injected at PSDU level, following the legacy non-HT IEEE 802.11 frame structure used by IEEE 802.11a/g [84].

The Peak/Flat modulated WuS is injected into the PSDU of the WLAN frame, thus, the rest of the headers and footers defined by the standard remain unmodified. This

allows better coexistence with other IEEE 802.11 stations since the frame includes all the coexistence mechanisms defined by IEEE 802.11a/g, including the Network Allocation Vector (NAV) headers. The insertion of the WuTx inside a standard-compliant IEEE 802.11g non-HT frame with the short Physical Layer Convergence Procedure (PLCP) preamble is represented in Fig.3.10.

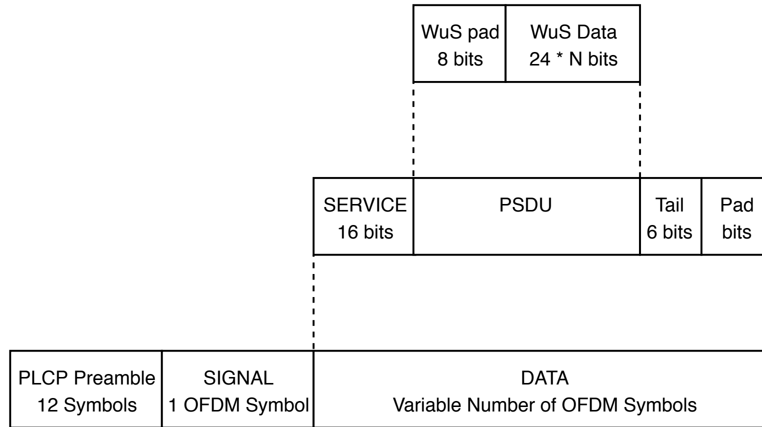


Figure 3.10: Insertion of the WuS inside the IEEE 802.11g frame structure.

The following procedure allows generating a bit sequence that produces a Peak-Flat-based WuS (see Section 3.3). As mentioned in previous sections, this procedure is compatible with any standard compliant IEEE 802.11a/g OFDM PHY transmitter if the scrambler seed can be obtained or set after sending a frame.

1. **Add padding to the bit sequence.** The first 16 bits corresponding to the IEEE 802.11 non-HT PPDU are occupied by the Service field, which is not part of the PSDU. Padding is required to begin the bit sequence at the OFDM symbol boundary. Thus, 8 bits of padding are appended at the start of the PSDU to complete the first 24-bit OFDM symbol. Therefore, aligning the bit sequence to the next OFDM symbol boundary and allowing the next bits, which codify the WuS, to start on the second OFDM symbol of the PSDU.

2. **Append the WuS data to the bit sequence.** Add a bit block for each WuS payload bit. If the WuS bit value is “1”, the 24-bit Flat Symbol sequence derived in Section 3.3.2 can be used. Otherwise, codify a “0” value, the 24 bit “0” sequence for a Peak Symbol derived in Section 3.3.1 needs to be used.

3. **Pre-scramble the bit sequence**. Apply the predicted scrambler sequence, starting after the padding previously added to the bit sequence. Nonetheless, proper alignment must be ensured. For this, the value of the seed must be advanced 24 bits, to match the real scrambler state at the start of the WuS. After this, for every bit in the sequence added in step 2), advance the predicted scrambler state one bit and XOR the predicted scrambler result to that bit.

4. **Send the resulting binary sequence to the transmitter.** After 3) the bit sequence is completed. Thus, it can be input to the IEEE 802.11a/g PHY for transmission.

The previous procedure was applied to the MATLAB WLAN Toolkit generating a WuS frame ‘with the payload “0101010101010101010”, as displayed in Fig. 3.11.

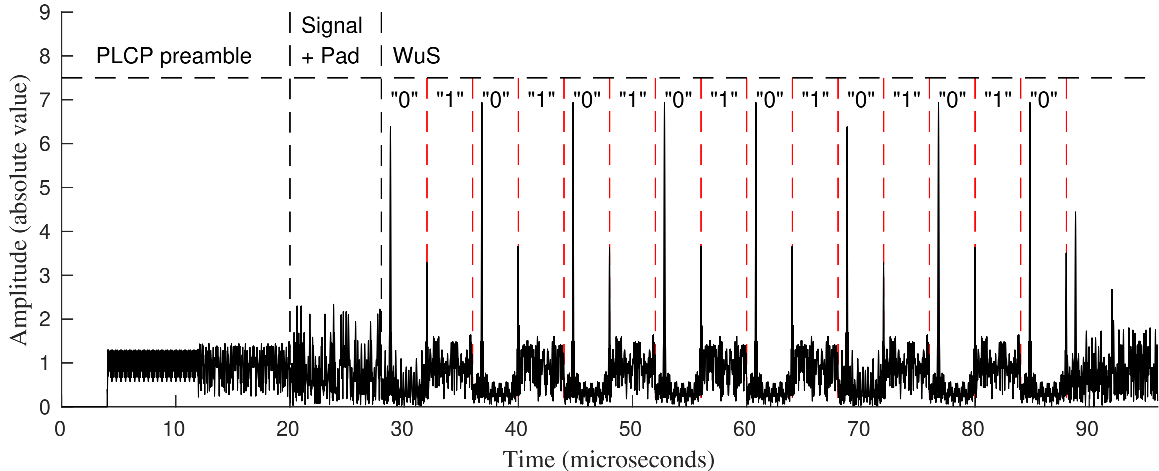


Figure 3.11: The WuS as generated by the PoC WuTx, encoding the binary value “0101010101010101010” in the WuS.

This last sequence is useful to highlight the differences between Peak and Flat symbols since it alternates them. As it can be better observed in Fig.3.12, the Flat Symbols that follow a Peak Symbol display a secondary peak, with roughly half of the amplitude of the peak featured in the previous Peak Symbol. This is caused by the MATLAB WLAN Toolbox implementation of windowing, which occurs at the start of every OFDM symbol. The purpose of this mechanism, which is not fulfilled in this case, is to smooth transitions between consecutive symbols.

Consider the waveform generated by the WuTx and represented in Fig.3.11. The WuS displayed carries the WuS payload “010101010101010”. This sequence, which alternates Peak and Flat Symbols, displays the differences between the two possible waveforms of the Peak-Flat modulation. Peak Symbols, labeled as “0” feature their characteristic peak and the low amplitude region that follows it. In contrast, Flat Symbols, labeled as “1”, have less disparity between sample values and a more rectangular shape. As it can be better observed in Fig.3.12, the Flat Symbols that follow a Peak Symbol display a secondary peak with roughly half of the amplitude of the peak featured in the previous Peak Symbol. This is caused by the MATLAB WLAN Toolbox implementation of windowing, occurring at the start of every OFDM symbol. The purpose of this mechanism, which in no way is fulfilled in this case, is to smooth OFDM symbol transitions.

Newer IEEE OFDM PHY releases, such as those defined in IEEE 802.11n/ac/ax

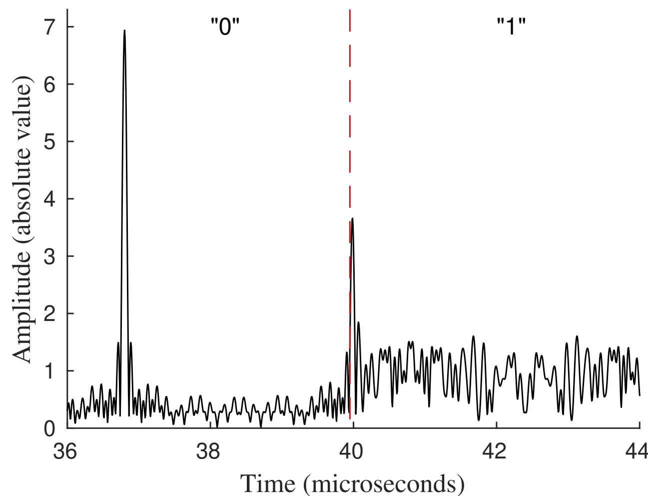


Figure 3.12: Detail of two consecutive Peak and Flat Symbols from the waveform displayed in Fig.3.11.

define transmission modes entailing a signal processing chain that is compatible with the presented method. In this way, the described procedure can be applied to single spatial stream transmissions, i.e., from MCS 0 to MCS 7. This is valid if the transmitter is configured to use the compulsory Convolutional Coder-based FEC and the optional space-time block coding feature is not in use.

3.4.2 PHY derived impairments in Peak-Flat Symbol generation

Although the present work introduces a software-based method to generate a WuS, the impact of the proposed waveforms on the physical components of the transmitter must be considered. Even if Peak-Flat defines what is a completely valid IEEE 802.11 signal, the high dynamic range of Peak Symbols can introduce distortion and, as a consequence, out-of-band emissions. This section addresses the effects on power amplifier non-linearity and saturation, which can affect the regulatory compliance of the Peak-Flat modulation, as well as the reception of the Peak-Flat Symbols.

Thus, to analyze this possible issue, the regulatory compliance of Peak-Flat signals has been evaluated with a simulated non-ideal power amplifier model. Fortunately, the MATLAB WLAN Toolbox spectrum compliance tools to check for out-of-band emissions against the spectral mask defined by IEEE 802.11a, which can be easily integrated with the previously developed WuTx PoC. These tools were used with a non-linear amplifier model to evaluate the effect of non-linearity.

The effect of non-linear amplification has been modeled with a Rapp model [93] with

parameter $p = 3$. To emulate the most restrictive scenario, the evaluation includes the Peak-Flat configuration that generates the highest PAPR (IEEE 802.11g with 6 Mbps data rate and the "-1" BPSK symbol, with a maximum of 17,39 dB PAPR). Another parameter of the model is the saturation power of the amplifier. The MATLAB model allows setting this value as relative to the average signal power. Here, saturation power is expressed in dB, relative to the mean average transmission power of the frame. PAPR models the maximum height of the peak, versus the frame average, thus, the peak power of the Peak Symbols considered is 17.39 dB. To test compliance, the spectrum of the non-ideal Peak Symbols is fit against a spectral mask. Moreover, the clipping suffered by the Peak Symbol is also measured.

Following this setup, the effect of the amplifier saturation, from here on A_{sat}, referred to as dB above the mean signal power is evaluated.

Initially, a saturation A_{sat} value of 27 dB causes no clipping, as can be seen in Fig. 3.13. Consequently, the signal remains completely inside the spectral mask.

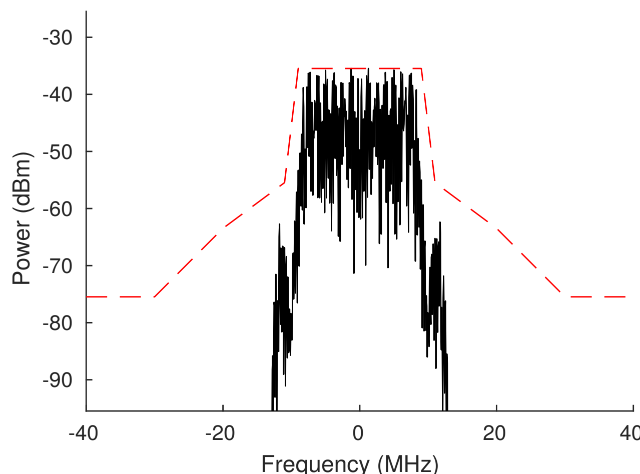


Figure 3.13: Spectrum of a peak signal with $A_{sat} = 27$ dBm. The dashed red line represents the spectral mask.

Later, the A_{sat} was lowered until the signal no longer fit inside the spectral mask. Unexpectedly, Peak Flat can operate with an amount of distortion. For example, even with an A_{sat} of 14.75 dB, which falls below the peak power, it maintains compliance. Nonetheless, such a level of distortion causes visible clipping of the peaks, and, consequently, reduces the performance of WuRx receiver architectures based on peak-detectors. The resulting waveform is displayed in Fig. 3.14, and the peak amplitude, while clipped, maintains its compliance with the spectrum mask.

However, an A_{sat} lower than 14.75 dB causes Peak-Flat to fail compliance tests. Nonetheless, even with lower A_{sat} values, compliance with the spectrum mask is still possible, although, the transmitted power must be reduced accordingly. For example, if the mean average transmission power is reduced to -3 dB in relation to the reference, the system remains compliant down to an A_{sat} of 11.75 dB.

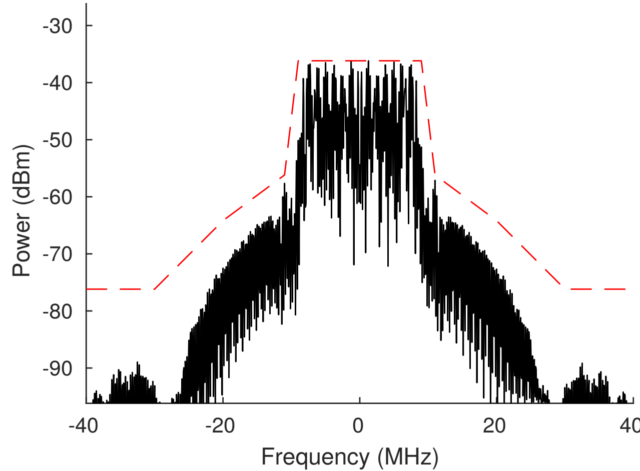


Figure 3.14: Spectrum of a peak signal with $A_{sat} = 14.75$ dBm. The dashed red line represents the spectral mask.

In conclusion, Peak-Flat remains compliant with the IEEE spectrum masks for OFDM, if the saturation power of the amplifier remains at most 2.64 dB lower than the peak transmitted power. After that point, the transmission power must be lowered accordingly to ensure compliance. Therefore, as with the rest of IEEE 802.11 signals, the final limit to allowed transmission power for Peak-Flat depends on both the regional regulatory limits and the saturation limit imposed by the power amplifier of the device.

Nonetheless, regulatory compliance can be improved by using Peak Symbols with low-level peaks. This can be effectively achieved using modulations with multilevel symbols. For example, the 16-QAM symbol “1 + 1j”, from the Table 3.2. This symbol produces the lowest relative Peak of those featured, with an amplitude 3 times lower than the highest amplitude in its constellation. Since this symbol is produced by a uniform block of low-amplitude symbols, its average amplitude is lower when compared to its respective Flat Symbol. Thus, it presents a higher extinction ratio than other Peak-Flat variants, improving its reception with an OOK receiver. However, this comes at the cost of lowering its performance with a WuRx that uses a peak-detector design.

To evaluate this effect, the performance of using 16-QAM and BPSK under two WuRx architectures will be determined in the next Section.

3.5 Simulation of the Wake-up Receiver

Two architectures of WuRx are proposed to receive the Peak-Flat modulated WuS, one using a peak-detector circuit, and another implementing an OOK receiver. This

section presents two WuRx, each based on one of the proposals. Moreover, these receivers serve as a base of a benchmark used to evaluate the performance of the Peak-Flat against OOK-based signals.

As a WuTx PoC was already developed with MATLAB, a Simulink-based model was selected to evaluate WuRx architectures. The designs proposed are presented as a means to study and compare the performance of Peak-Flat in each receiver architecture, as well as to compare it with other modulations used in the WuR state-of-the-art. Consequently, both proposals feature minimal designs, modeled using off-the-shelf components. Of course, the results obtained by these WuRx are by no means intended to be compared against the current WuRx state of the art, nor serve as optimal receivers for the Peak-Flat modulation, nor any other.

3.5.1 Architecture of the WuRx

Although this section presents two architectural proposals for the WuRx, these present some common elements. The proposed WuRx, which is shown in Fig.3.15, follows an active WuRx architecture, a common design pattern used in many low-power WuRx designs [17]. Thus, it can be divided into two main blocks. The first is the Radio Front End, which is tasked with signal conditioning. Its output is the incoming signal filtered and modulated down to a baseband waveform. Second, follows the decoder, which decodes the incoming analog waveform into a binary stream. The two WuRx architectures evaluated differ in the decoder section and use a common Radio Front End- section.

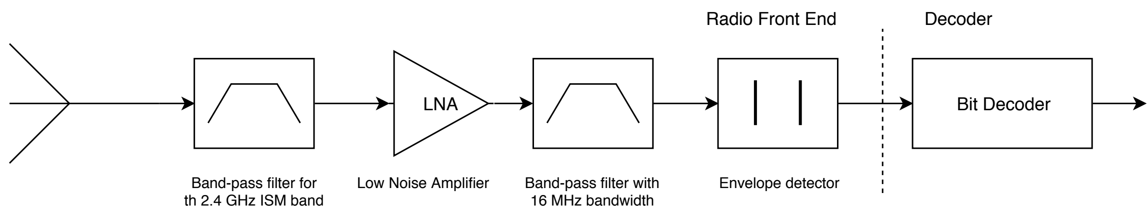


Figure 3.15: Structure of the proposed WuRx.

The proposed Radio Front End consists of the following elements: an antenna, a band-pass filter (tuned to the 2.4 GHz ISM band), a Low Noise Amplifier (LNA), a band-pass filter, and, finally, an envelope detector. Nonetheless, since the MATLAB WLAN toolkit already outputs complex signals in baseband, these do not need to be implemented for the simulated WuRx benchmark.

Following, the two decoder alternatives, one for each architecture are presented.

3.5.2 An OOK decoder

The first WuRx decoder implementation follows the basic design for OOK signal detection. As shown in Section 3.3.1 and Fig.3.12 the Flat Symbols have a higher mean amplitude than Peak Symbols. Thus, Peak-Flat can be received by OOK receivers as an imperfect OOK modulation, with reduced extinction ratios. Of course, this receiver is still capable of decoding OOK-based modulations, like the one used in the current IEEE 802.11ba proposal. More interestingly, Peak-Flat can be detected by standard OOK receivers, albeit, with degraded performance when compared to ideal OOK symbols.

To facilitate its simulation, the OOK receiver implementation shown here follows a minimal design based on off-the-shelf components: two filters, and a comparator. A block diagram of the proposed design is shown in Fig.3.16. The first filter is tuned close to 250 kHz, matching it with the IEEE 802.11g OFDM symbol rate. This first low-pass filters out noise and smooths the envelope of the signal. Its cut-off frequency of 250 kHz was determined heuristically, by comparing the obtained bit error rates in a parametric simulation.

The second filter provides a reference signal for the comparator; thus, it is tuned to a frequency that allows it to obtain an almost constant envelope from a regular IEEE 802.11 frame. Also obtained by parametric simulations, its cut-off frequency is fixed at 2.5 kHz.

Finally, a comparator is used to compare the output of both filters. If a Flat Symbol is present, the output of the first filter (which tracks signal envelope) is higher than the one from the second filter (which provides the reference value), therefore, the comparator output becomes high. On the contrary, for a Peak Symbol, the output from the first filter is lower than the reference value, thus, the comparator output becomes low.

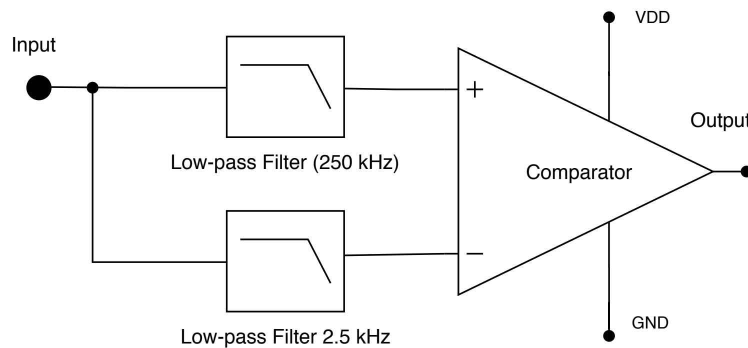


Figure 3.16: Structure of the proposed decoder based on OOK reception.

3.5.3 A Peak detector decoder

The second proposed decoder implementation for the WuRx distinguishes between Peak and Flat Symbols by the presence of a peak above a certain amplitude. Due to their design, Peak Symbols produce a detectable peak, well above the maximum signal level produced by Flat Symbols. The presence of this peak can effectively be detected using a non-linear peak detector circuit.

The implementation of the Bit Decoder is composed of a peak detector circuit, a voltage divider, a low-pass filter, and a comparator. A block diagram of the proposed design is shown in Fig.3.17.

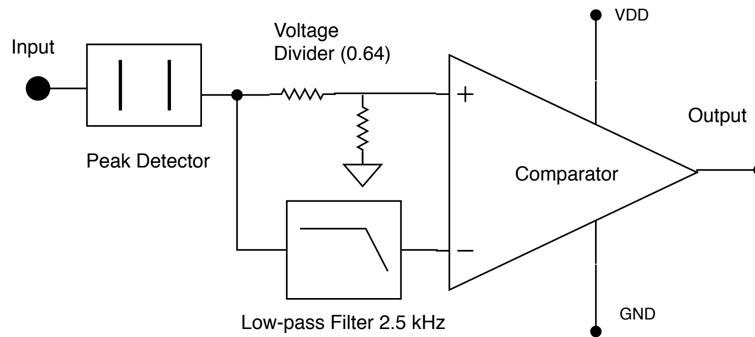


Figure 3.17: Structure of the proposed decoder based on peak detection.

The peak detector circuit is a non-linear circuit, which is also used for AM signal demodulation. It features a Schottky diode, followed by a capacitor for charge storage and a resistance, connected in parallel. When the input signal is greater than the diode voltage threshold, the capacitor will charge through the parasite series resistance of the diode, which is low. However, the capacitor must discharge through the output terminal, as the diode blocks the flow of current back to the input terminal. This slows the discharge of the capacitor and, consequently, generates an asymmetry between the charge and discharge rates of the diode. Therefore, when a peak is received, the peak detector charges fast, and then, after the peak ends, it discharges at a slower speed. If a high enough equivalent resistance is presented by the output terminal, this non-linear effect increases the duration of the peak, which is originally 50ns (a single sample at 20 Msps). This increase in the duration of the peak protects the signal against synchronization issues, typical of feature limited electronics, and reduces the error rate.

To differentiate between Peak and Flat Symbols the peak voltage level must be compared to a reference signal. Therefore, the low-pass filter element in this receiver has the same purpose as the second filter of the OOK detector architecture (i.e., to provide a reference value based on the frame envelope). This component is placed after peak detection since this non-linear procedure changes the average amplitude of the received signal.

To reduce false peak detection a voltage divider is connected at the output of the peak detector. Without this component, whenever the peak detector output raised slightly above the average amplitude of the frame, a peak would be detected. The values of the resistances need to be relatively high since they are also the discharge path of the peak detector capacitor.

Finally, the decoding decision is determined by the comparator output. The voltage divider output is connected to the comparator non-inverting input, whereas the low-pass filter output is connected to its inverting input. Using this configuration, a peak only produces a high level at the output of the comparator when its amplitude, diminished by the voltage divider, is higher than the average envelope, provided by the low-pass filter. Conversely to the OOK receiver, the peak detector generates a high logic value for Peak Symbols and a low one for Flat Symbols.

For the Simulink implementation of this receiver, the filter keeping the frame reference level is, as in the OOK implementation, a first-order RC filter tuned at 2.5 kHz. The value for the capacitance used for the peak detector is 1nF, heuristically determined using a parametric sweep on the Simulink model. The Peak detector, jointly with the resistances of the voltage divider that provide its discharge path, increments the duration of the peak to 1 μs. For this purpose, the divider uses resistances of 4 kΩ and 7.6 kΩ.

3.6 Results

3.6.1 Evaluation environment

The simulated WuRx presented before are used to compare the performance of WuR modulations, including Peak-Flat, OOK, and the specific Manchester-coded OOK version used in IEEE 802.11ba fast data rate. Thus, to evaluate these modulations two simulations were produced.

First, the BER vs. SNR curve was obtained in an Additive White Gaussian Noise (AWGN) channel using the complete simulated WuR system, including the WuTx and WuRx. Second, the PER vs SNR curve was also simulated on a fading channel model, TGn Channel B. This channel model applies an attenuation at frame level, consequently, the bit error probability for bits in the same frame is not independent from the others. Thus, BER simulations need a much larger value of iterations to converge. That number was determined greater than it is feasible to simulate with the available computing hardware. Thus, to obtain results that are relevant to the performance of the evaluated modulations, the TGn Channel B evaluation uses the PER instead of the BER as its figure of merit.

In both AWNG and TGn Channel B simulations, the noise considered, and the resulting SNR, are specified at 20 MHz bandwidth. Nonetheless, the equivalent noise bandwidth that affects the WuRx is lower due to the low-pass filter used before envelope detection and the filters added in both receiver models. However, using SNR referred at 20 MHz allows comparing the results of the WuR modulations with standard WLAN signals.

The signal processing chain used in MATLAB for the evaluation in an AWGN channel is shown in Fig.3.18. It is based on the WuRx structure proposed in Section 3.5.1 and displayed in Fig.3.15, whoever, the processing chain only includes the baseband elements since the MATLAB WLAN Toolkit already produces baseband signals. Moreover, the evaluation of BER vs SNR using bandpass signals was not computationally feasible with the resources available. In this chain, the AWGN noise is added to the WuS generated by the WuTx at the start. Then, it is filtered and demodulated using an ideal envelope detector. Finally, the resulting signal is input to the decoder. Finally, the binarized waveform output from the decoder is sampled and compared with the input bit stream for BER computation. To obtain accurate BER values, the evaluation of BER vs SNR is performed in increments of 0.5 dB of SNR. And, to obtain a BER with a resolution lower than 10^{-4} , 64000 bits are used in each SNR value.

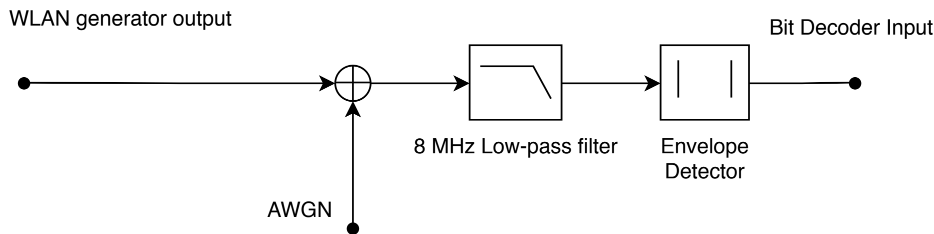


Figure 3.18: Diagram of the signal processing chain used for the evaluation of BER under AWGN.

The TGn Ch.B simulation signal processing chain used to simulate PER results in fading channels is shown in Fig. 3.19. In contrast to the prior AWGN simulation, now the incoming WLAN signal is distorted by a fading channel based on the TGn Channel B model. As before, the SNR increases in steps of 0.5 dB. To obtain reliable PER results, 5000 frames are simulated per each SNR step, having each frame a length of 128 bits.

This work evaluates the reception of an ideal 250 kBd OOK, and the 250 kbps Manchester-coded OOK used in IEEE 802.11ba fast data rate. These state-of-the-art solutions are evaluated only with the OOK WuRx architecture. To support Manchester code at 250 kbps, IEEE 802.11ba operates using 2 μs OOK pulses. Thus, to improve the reception of this signal, the receiver cut-off frequency for its first low-pass filter was increased to 500 kHz when evaluating this modulation.

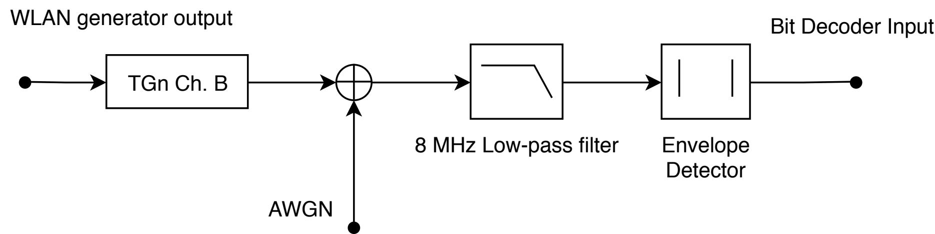


Figure 3.19: Diagram of the signal processing chain used for TGn Ch.B PER evaluation.

Two variants of the Peak-Flat modulation are evaluated with the OOK WuRx architecture. First, the Peak-Flat modulation using BPSK, which is optimized for PAPR. Second, the Peak-Flat modulation with 16-QAM constellation proposed in Section 3.4.2, which provides a reduced PAPR, but higher extinction ratios. Only the Peak-Flat modulation using the BPSK constellation is tested with the peak-detector-based WuRx. This is the variant of Peak-Flat that achieved the highest PAPR, thus, it is the one that benefits more from this receiver architecture.

Further detail concerning the software implementation of these simulations is contained in Appendix A.

3.6.2 Decoder evaluation results

First, the AWGN simulation results are shown in Fig.3.20a, which includes BER values down to $5 \cdot 10^{-4}$. The results favor the BPSK-based Peak-Flat modulation, evaluated with the peak detector. It achieves a 10^{-3} BER before all other modulations, at roughly 3.5 dB of SNR. The next, the manchester-coded IEEE 802.11ba, needs at least 4.5 dB SNR to achieve the same result. 250 kBd OOK falls behind at roughly 5 dB SNR for a sensitivity of 10^{-3}. Peak-Flat variants fall well behind when received with an OOK detector. Low-PAPR Peak-Flat (obtained using the 16-QAM constellation) achieved the 10^{-3} BER at 8.5 dB of SNR, 3.5 dB later than an equivalent ideal OOK. Finally, BPSK Peak-Flat modulation achieved sensitivity at 12.5 dB, 7.5 dB below an equivalent ideal OOK.

These results indicate that BPSK-based Peak-Flat performance under AWGN is better than state-of-the-art if a Peak-Detector WuRx is used, however, it falls behind when received by a WuRx based on an OOK detector. Nonetheless, this disadvantage is reduced if a low-PAPR Peak-Flat variant is used, such as one based on the 16-QAM constellation. The reduced performance of Peak-Flat variants with an OOK-based WuRx remains consistent with their reduced extinction ratios versus an ideal OOK signal.

Second, the TGn Ch.B simulation results are shown in Fig.3.20b, which include

PER values down to $5 \cdot 10^{-2}$. In a coherent manner with the addition of a fading channel, all modulations present worsened results in these simulations. The best result is obtained by the BPSK constellation-based Peak-Flat modulation evaluated with the peak detector. It achieves a 10^{-1} PER at 10 dB of SNR, higher than both state-of-the-art modulations evaluated with the OOK-based WuRx. These achieve the same result, obtaining a 10^{-1} PER at 13 dB of SNR. Surprisingly, the low-PAPR Peak-Flat variant achieves the same performance as ideal OOK modulations. Nonetheless, the high-PAPR Peak-Flat modulation, decoded with OOK detector, falls behind the other options, achieving a 10^{-1} at 19 dB, a result that falls outside the area shown in Fig.3.20b.

These results, consistent with the AWGN simulations, indicate that BPSK-based Peak-Flat performance with the peak-detector WuRx is, at least, equivalent to the state of the art. Moreover, they indicate that low-PAPR Peak-Flat performance is similar to an ideal equivalent-rate OOK under realistic channel conditions.

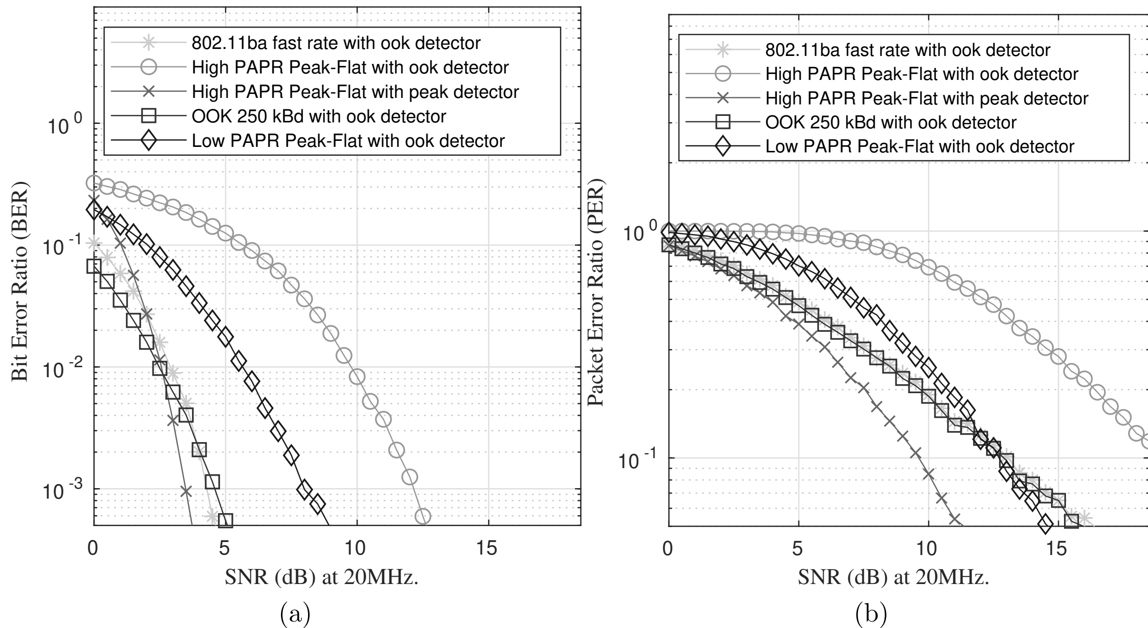


Figure 3.20: Simulation of Peak-Flat performance on the presented receivers versus various modulations. (a) BER vs SNR in AWGN channel and, (b) PER vs SNR in TGn Ch.B.

Thus, the Peak-Flat modulation performance appears to be better or equal to state-of-the-art WuR modulations, when tested with the proposed WuRx implementations. However, more complex and specialized IEEE 802.11ba receivers can achieve better results than the receivers proposed in this chapter, which were designed to provide equitable demonstration platforms. In this way, preliminary results appearing in several TGba documents already suggest that better sensitivity figures are achievable for IEEE 802.11ba [85, 86]. Nonetheless, the comparisons here use equivalent receivers and provide evidence that Peak-Flat performance can be, at

least, competitive with equivalent rate OOK modulations. Additionally, Peak-Flat modulation provides the added value of being compatible with legacy IEEE 802.11 devices. Moreover, the Peak-Flat modulation receiver can be evolved, from the PoC receiver presented, into a more advanced one.

3.7 Compatibility with IEEE 802.11ba and other releases

This Ph.D. thesis proposes a WuR system for legacy IEEE 802.11 devices that is an alternative to what is defined in IEEE 802.11ba. Nonetheless, enabling legacy and IEEE 802.11ba-enabled devices to interact would improve the applicability of this proposal.

Both IEEE 802.11ba and the WuR system proposed in this chapter propose amplitude-based modulations that can be received with OOK detectors. Moreover, IEEE 802.11ba Low Rate (LR) [68], uses a 62.5 kbps Manchester code. The codewords are formed by 4 OOK pulses, alternating high and low levels. Each of the aforementioned pulses is 4 μs long, thus, this signaling scheme can be reproduced by chaining 4 Peak-Flat Symbols. These would be received by IEEE 802.11ba receivers as pseudo-OOK symbols.

This opens the possibility of interoperability between the two WuR systems. Nonetheless, due to the framing mechanisms used by IEEE 802.11ba, the resulting interoperability is not straightforward. There are then two possible interactions between devices implementing our Peak-Flat-based WuR solution and IEEE 802.11ba devices.

First, let us consider a situation where an IEEE 802.11ba device uses its WuTx to wake up a Peak-Flat-enabled device. As discussed in Section VIII, the WuRx for the Peak-Flat device can be implemented using two different architectures. If the OOK detector architecture is used, then, the LR IEEE 802.11ba WuS signal will be detected since it is already OOK-coded. However, the OOK-coded signal cannot be reliably decoded by the peak detector design due to its higher detection threshold, which misses '1' OOK pulses. Nonetheless, the OOK WuTx should be adapted to correctly receive the synchronization preambles used by IEEE 802.11ba, which use a sequence of 2μs OOK symbols. Despite this, devices incorporating Peak-Flat WuRx could interpret an IEEE 802.11ba WuS, and consequently, be awoken by standard-compliant IEEE 802.11ba devices.

Second, let us consider the reverse, a Peak-Flat device awakening an IEEE 802.11ba device. Here, even if the symbols that compose the data field of frames are compatible, the framing is not. The rest of the framing, including synchronization headers,

needs to be compatible to enable a wake-up. These headers use the 2 μs pulses defined by the current draft of IEEE 802.11ba [68], which cannot be generated with the legacy-compatible Peak-Flat modulation. Consequently, the WuR system described in this work will not be able to wake up sleeping IEEE 802.11ba stations.

3.8 Prototype implementation

The developments in the previous sections open the door to the implementation of the Peak-Flat WuTx in commodity IEEE 802.11 hardware. Nonetheless, a precondition to implement this WuTx is having access to the scrambler seed of the IEEE 802.11 transmitter. This can be solved either by direct access to it through the manufacturer driver, or through any means to predict its value using meta-information (i.e., MAC addresses, number of bytes transmitted) for the given piece of hardware [89, 94].

Currently, there is no way to obtain or set the scrambler seed in openly documented operating systems. For example, the Linux IEEE 802.11 driver interface does not support either setting or retrieving this data [95]. Moreover, an in-depth search did not reveal any commodity WLAN chipset that allowed access to these values directly through any public driver interfaces, or extensions. Therefore, a reliable way to obtain scrambler sequences through other means was needed.

According to the literature [89, 94] various IEEE 802.11a/g implementations produce predictable scrambler seeds. Nonetheless, most of these require knowledge of a previous scrambler value to infer subsequent values. However, some of these use always the same scrambler state, i.e., the RTL8192CU chipset, among others (see next subsection).

The scrambler seed used for a given frame is obtained by the receiving station using the first 7 SERVICE field bits. As these bits are always set to 0, their value can be used by the receiver to retrieve the scrambler state. However, access to this field from incoming packets is not available to the clients of commodity WLAN cards. Consequently, packet analyzers such as Wireshark do not include this information in captures. To search for other IEEE 802.11 implementations with fixed scrambler sequences, an Software Defined Radio (SDR)-based solution was devised.

3.8.1 Using an SDR to characterize scrambler seed variability in commodity devices

With an SDR, all the signal processing is either done or controlled, by software [96]. Thus, all stages in the signal processing chain of an SDR receiver can be inspected.

There are various SDR-based implementations of IEEE 802.11 prior to this work. One of them, published by Bloessl et al. [97], is available for the popular GNU-Radio SDR framework [98]. This SDR implementation has also been made open-source and shared with the wider community in a Github repository [99].

The repository contains various application examples, one of them is a complete IEEE 802.11a/g/p receiver, which outputs incoming frames to a Wireshark-formated capture file. The frames in the capture contain additional physical parameters to those that can be obtained through standard IEEE 802.11 frame captures with commodity hardware. In the example, the frequency offset is added to those. Nonetheless, this example does not include the scrambler seed of the received frames.

To display the scrambler seed, the GNU-Radio block tasked with decoding the incoming MAC frame was modified to output the scrambler seed received. This value was added to the Wireshark capture via RFtap protocol.

Using this receiver, the scrambler seeds used by various commodity IEEE 802.11 hardware that were already acquired for other projects were tested. This was used to search for those that use a fixed scrambler seed. This way, several of those devices were found:

- The Linksys WUSB54GCv1, a USB-based device, using the Railink RT2501 chipset.
- The Edimax Nano USB EW-7811Un, a USB-based device, using the Realtek RTL8188CUS chipset.
- The ESP-32 microcontroller, developed by Espressif.

These devices comprise USB transceivers, which are usable in Linux-based computers, as well as embedded platforms, such as the ESP-32. All of these are susceptible to be used as WuTx, using the software-based method exposed in 3.4.1.

3.8.2 Implementation of the Peak-Flat WuTx in a Linux-based device

To check the viability of the Peak-Flat WuTx, its implementation was started using a Linux PC with the 4.1 kernel version and the Linksys WUSB54GCv1 USB-based transceiver.

The implementation was undertaken using the Python language. Although it is a high-level programming language, Python allows interacting with network interfaces directly, and send data at the raw socket level. Nonetheless, user-level APIs in Linux

do not allow enough low-level access to implement the procedure described in Section 3.4.1, which introduces the WuS inside the Physical Service Data Unit (PSDU). Only those bytes set in the IEEE 802.11 MPDU can be freely modified using raw-socket access.

Consequently, the procedure described in 3.4 was modified to account for further offset introduced by the MAC headers, which are set before the start of the WuS. Evaluated for a unicast frame, these headers add 36 bytes of data before the start of the WuS. As the length of these is specified in bytes, they will be mapped into a different number of symbols for each data rate, therefore, the scrambler is advanced a number N of bits, where N is obtained using (3.18) and depends on Sym_{bits} the number of data bits coded per OFDM symbol, which can be found in Table 3.4.

$$N = \left\lceil \frac{36\dot{8}}{Sym_{bits}} \right\rceil \tag{3.18}$$

Moreover, the padding also varies in function of Sym_{bits} and needs to be calculated using the expression in (3.19).

$$N = \left\lceil \frac{36 \cdot 8}{Sym_{bits}} \right\rceil - 36 \cdot 8 \tag{3.19}$$

Table 3.4: Bits per symbol corresponding to each data rate supported by IEEE 802.11a/g

Data Rate (Mbps)	Bits
6	24
9	36
12	48
18	72
24	96
36	144
48	192
54	216

The WuTx functionality was packaged in a command-line interface called ***packet_generator.py***. It allows the user to send arbitrary IEEE 802.11 frames with WuS data, using any of the data rates and symbol combinations defined previously in 3.3. Of course, provided that the correct scrambler seed is correctly passed in the input.

In Fig. 3.21 the output generated by the WuTx based on Linksys WUSB54GCv1 can be seen for the WuS sequence: "0101010101010101010".

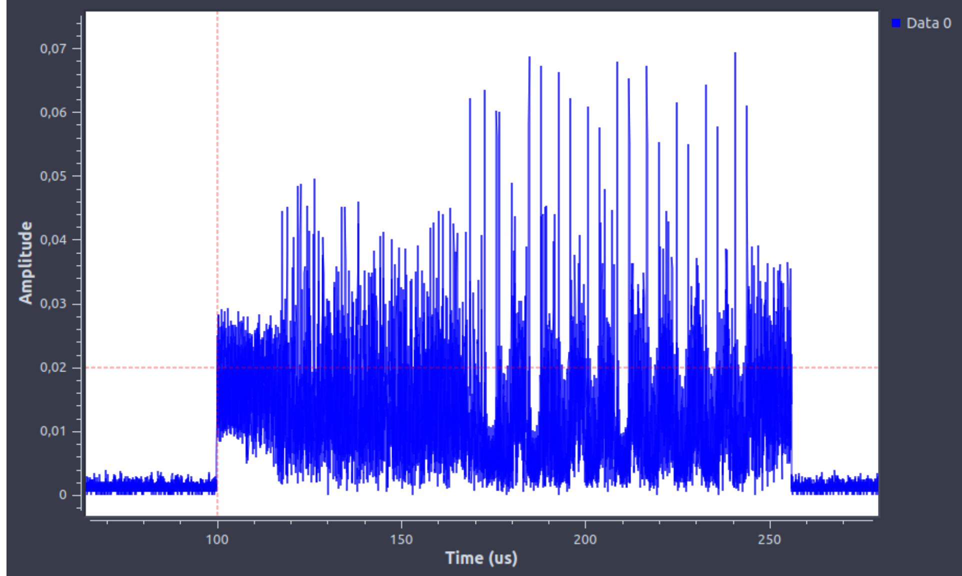


Figure 3.21: The envelope of a WuS generated by a WuTx with Linksys WUSB54GCv1, using the encoding the binary value "0101010101010101010" captured using a GNU Radio scope with a USRP B200 SDR.

Further detail concerning the Python implementation of the WuTx, including snippets of source code is contained in Appendix B.1.

3.8.3 Implementation of the Peak-Flat WuTx in an embedded device

After the success of the Linux-based WuTx implementation, the implementation of the Peak-Flat WuTx was attempted in an embedded platform, the ESP-32. This low-cost System on a Chip (SOC) is relevant to current IoT developments. Therefore, it is closer to the target application environment for IEEE 802.11ba.

Although there are Python-based SDKs for ESP-32 [100], the implementation of the Peak-Flat WuTx requires access to low-level functionality that is not available from those. Consequently, the development started in Section 3.8.2 was ported to the ESP-32 native SDK: ESP-IDF, v3.2 [101]. The previously developed Linux WuTx implementations was ported to C/C++, and the specific WLAN interaction model supported by ESP-IDF.

This framework enables low-level interaction with the WLAN hardware, however, in the same way, as in Linux, only access to the MSDU is allowed. Therefore, the same solutions taken in the previous section to allow for the insertion of the WuS in the MSDU also were applied here. Nonetheless, not all combinations of data rates and symbols were implemented for this platform. Those implemented were the ones

showing more promise, BPSK with the -1 symbol and 16-QAM with the +1 symbol.

Further detail concerning the C implementation of the ESP-32 based WuTx, including snippets of source code is contained in Appendix B.2.

3.9 Conclusion

This chapter has presented a WuR system alternative to IEEE 802.11ba, that is compatible with legacy IEEE 802.11 devices. With it, WuR can be extended to legacy-compatible IEEE 802.11 devices. Thus, proving that a legacy-compatible WuR implementation, alternative to IEEE 802.11ba, is feasible. This implementation is based on an amplitude-based modulation compatible with low-power WuRx, which, additionally, can also be received by OOK receivers with a sensitivity penalty. The proposed Peak-Flat modulation used by the WuTx is generated with a software-based method particularized for IEEE 802.11g. Consequently, it can be generated by any standard-compliant IEEE 802.11g OFDM PHY transmitter without requiring any hardware addition. Nonetheless, the proposed method requires that the scrambler seed used by the IEEE 802.11 transmitter is known in advance. At the moment, support for this feature is not found in public IEEE 802.11 driver interaction models, however, the possibility of using WuR can drive more manufacturers to expose this functionality at the driver level or, alternatively, document the seed generation procedure so the scrambler seed can be effectively predicted. It has also been shown that Peak-Flat can coexist with other IEEE 802.11 stations, as well as follow the spectrum masks mandated by the IEEE 802.11 standard after accounting for non-linear amplifier models.

The performance of Peak-Flat has been evaluated using detailed MATLAB simulations of an IEEE 802.11 transmitter and two different WuRx architectures, all based on off-the-shelf components. With this evaluation, it has been shown that Peak-Flat can operate with a comparable performance against state-of-the-art WuR modulations, when evaluated under AWGN and fading channel models. Finally, Peak-Flat has been implemented using commodity IEEE 802.11 hardware in embedded and non-embedded devices. This chapter concludes with the implementation of the Peak-Flat WuTx, therefore, to obtain a complete WuR system, the construction of a low-power WuRx is needed. Thus, the next step is to design a receiver based on any of the previously evaluated architectures.

This chapter was published as an article in *IEEE Access* (Q1, Computer Science, Information Systems 35/156 [102]) on the 9th of April 2019:

M.Cervià, A.Calveras, E.López, E.Garcia, I.Demirkol and J.Paradells. An Alternative to IEEE 802.11ba: Wake-up Radio with Legacy IEEE 802.11 Transmitters. *IEEE Access [103]*

Additionally, the software-based method to generate WuS with Peak-Flat is in process of being patented.

M.Cervià, A.Calveras, E.López, E.Garcia, I.Demirkol and J.Paradells. EP3824607 - A METHOD AND A DEVICE TO GENERATE AN AMPLITUDE-BASED MODULATION WIRELESS SIGNAL USING OFDM TO BE RECEIVED BY A LOW-POWER NON-COHERENT RECEIVER. ***Patent*****[104]**

Chapter 4

A microcontroller-based WuRx

As previously discussed, the performance of WuR as a MAC-level energy-saving technique depends strongly on the power consumption of the WuRx. To achieve low latency, the WuRx needs to be close to always-on, continuously waiting for incoming WuS. To illustrate this, Fig.4.1 presents a WuRx turning on a device after receiving a WuS.

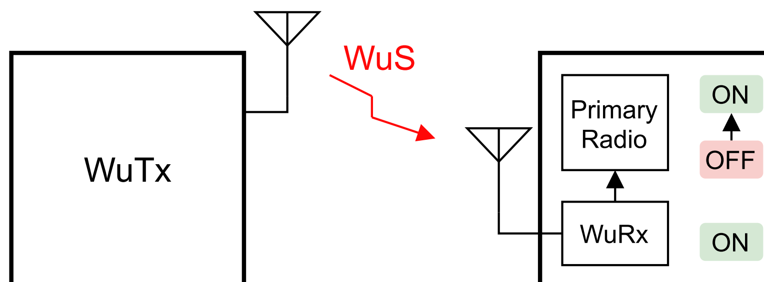


Figure 4.1: A WuRx receiving a WuS and turning on its associated primary radio.

Low-power WuRx are built using state-of-the-art CMOS manufacturing processes [105]. This is a barrier of entry to WuR research, both in terms of cost and expertise. Consequently, WuRx development is a research topic that is commonly outside the reach of most network research groups, which lack expertise in microelectronics and, generally, rarely can access the investment required. Thus, research in WuR networks is done mainly by simulation [106][69][17]. The field could benefit from an accessible WuRx design, which facilitated the implementation and evaluation of WuR networks in testbed form. Such a tool would allow networking research groups to improve their output to the WuR state-of-the-art with real-life testbeds.

Thus, this chapter presents an accessible WuRx design based on off-the-shelf components, aimed to lower the barriers of entry to WuR research. Therefore, allowing other researchers to replicate this WuRx implementation, as well as extending it, without requiring a significant investment. Following the state of the art, the

WuRx proposed in this chapter includes a high rate baseband capable of supporting a 250 kbps OOK data rate, equivalent to the maximum bit rate supported by IEEE 802.11ba [68]. Moreover, besides OOK, it is compatible with the legacy IEEE 802.11 WuTx introduced in Chapter 3. Thus, the proposed WuRx, with the legacy-compatible WuTx implementation presented in that chapter, provide a complete WuR system. Thanks to its high bit rate, this WuR system can be applied to other more generic uses, such as transmitting arbitrary information between devices.

The work in this chapter was done with the collaboration of Maison Hussein, which presented part of the results herein contained in her Master's Thesis [107]. Her role included the software implementation of the WuRx, as well as the realization and validation of the experimental results. The work developed in the context of this Ph.D. thesis included the protocol design, the design of the software and hardware elements, and the procedures for their validation. All these developments are presented in this chapter.

4.1 Design and Implementation of a WuRx using off-the-shelf parts

The proposed WuRx must decode frames sent with a 250 kbps OOK modulation. For its implementation, the WuRx [17] was separated into two main subsections: the first, the RF front-end, which performs signal conditioning and outputs a binarized waveform; the second, the baseband, which decodes the binarized waveform output by the first. The baseband detects the start of frames. Afterward, after a frame has been detected, it samples an address and, if it matches the expected, wakes up the device connected to the WuRx. This design is featured in Fig. 4.2.

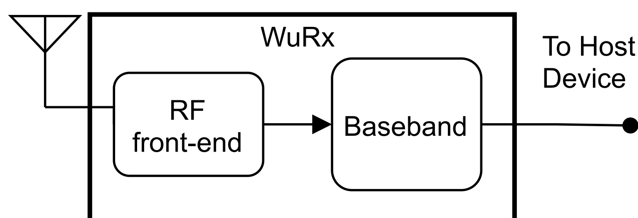


Figure 4.2: Structure of the proposed WuRx.

The development of the WuRx was separated into two different projects, each one concerning one of the aforementioned parts. The first, discussed here [107] covers the implementation of a low-power baseband, while the second, [108] covers the development of a low-power RF front-end.

4.1.1 WuRx RF front-end implementation

An RF front-end composed of off-the-shelf non-low-power components that were commercially available was used to validate and integrate the baseband section of the WuRx. Therefore, allowing to development of the baseband and the RF front-end to advance independently from each other.

The front-end implementation, which follows an active WuR design, includes a signal conditioning chain with a Low-Noise Amplifier (LNA), a band-pass filter (BPF), an envelope detector, and a comparator. This minimal set of components allows the WuRx to operate in an indoor environment. Thus, enabling the evaluation of the baseband with real WuS. The off-the-shelf components used for the design are listed below in Table 4.1.

Table 4.1: Component parameters.

Component	Parameters	
LNA	Gain[1]	31 dB
	Noise factor	2.8 dB
	Supply voltage	5 V
	Supply current[1]	200 mA
BPF	Center frequency	2.45 GHz
	Bandwidth[1]	150 MHz
	Insertion loss	2 dB
LTC5508[109]	Supply Voltage	4 V
	Active current	550 μA
	Standby current	2 μA
CA3140[110]	Supply Voltage	4 V
	Supply current[1]	2 mA

[1] Value obtained experimentally.

The components included are connected as described in Fig.4.3. A detailed description of each component is available below, with the same numeration used in Fig.4.3.

1. A LNA with a nominal gain of 40 dB. This device, provided by the low-volume manufacturer GPIO-labs, covers the 2.4 GHz ISM band used by IEEE 802.11 signals. However, this component consumes 200 mA at a supply voltage of 5 V. Consequently, this LNA is not usable in a low-power WuRx design. Its gain goes from 30 to 40 dB over its operating bandwidth, which encompasses from

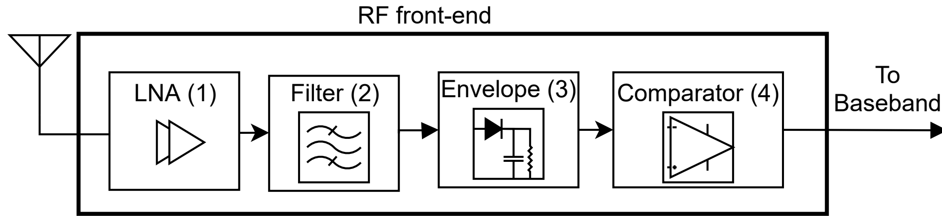


Figure 4.3: Components of the WuRx RF front-end.

100 MHz to 8 GHz. Specifically, at the 2.4 GHz band used by IEEE 802.11a/g signals, the gain was characterized to be 31 dB.

2. A band-pass filter for the 2.4 GHz ISM band. This filter, also sourced from GPIO labs, presents a center frequency of 2.45 GHz with 150 MHz of bandwidth. A filter with lower bandwidth could provide better Signal to Noise Ratio (SNR) figures to the WuRx. Optimally, the filter bandwidth should match a single IEEE 802.11 channel, which is the bandwidth of legacy-compatible IEEE 802.11 signals. However, there was not any readily available part with these characteristics. Additionally, its insertion loss, at 2 dB, is relatively low.

3. An envelope detector. This function is covered by the LTC5508 [109] power detector. Its operating range goes from 300 MHz to 7 Ghz, covers the 2.4 GHz band. Additionally, this device is available in the form of low-cost demonstration boards, which facilitates its integration with the rest of the WuRx. In contrast with the LNA, the LTC5508 consumes only 550 μA in active mode, and 2 μA in standby. Therefore, with a moderate duty cycle, it is compatible with a low-power WuRx design.

4. A comparator. This component was implemented using a general-purpose Operational Amplifier (OP-amp), the CA3140 [110]. It compares the incoming demodulated signal with a fixed 250 mV voltage threshold, obtained with a voltage divider. This threshold corresponds to an input signal of -24 dBm at the entry of the LTC5508, which, after amplification, is easily achieved indoors with commodity transmitters operating at 20 dBm in the 2.4 GHz band.

This front-end design was implemented in a rapid prototype. It can be seen paired with an antenna and the corresponding baseband microcontroller using a protoboard in Fig 4.4.

4.1.2 WuRx baseband implementation

The WuRx baseband is implemented in software, using a low-power microcontroller. This microcontroller samples the signal output by the RF front-end at 250 ksps. To

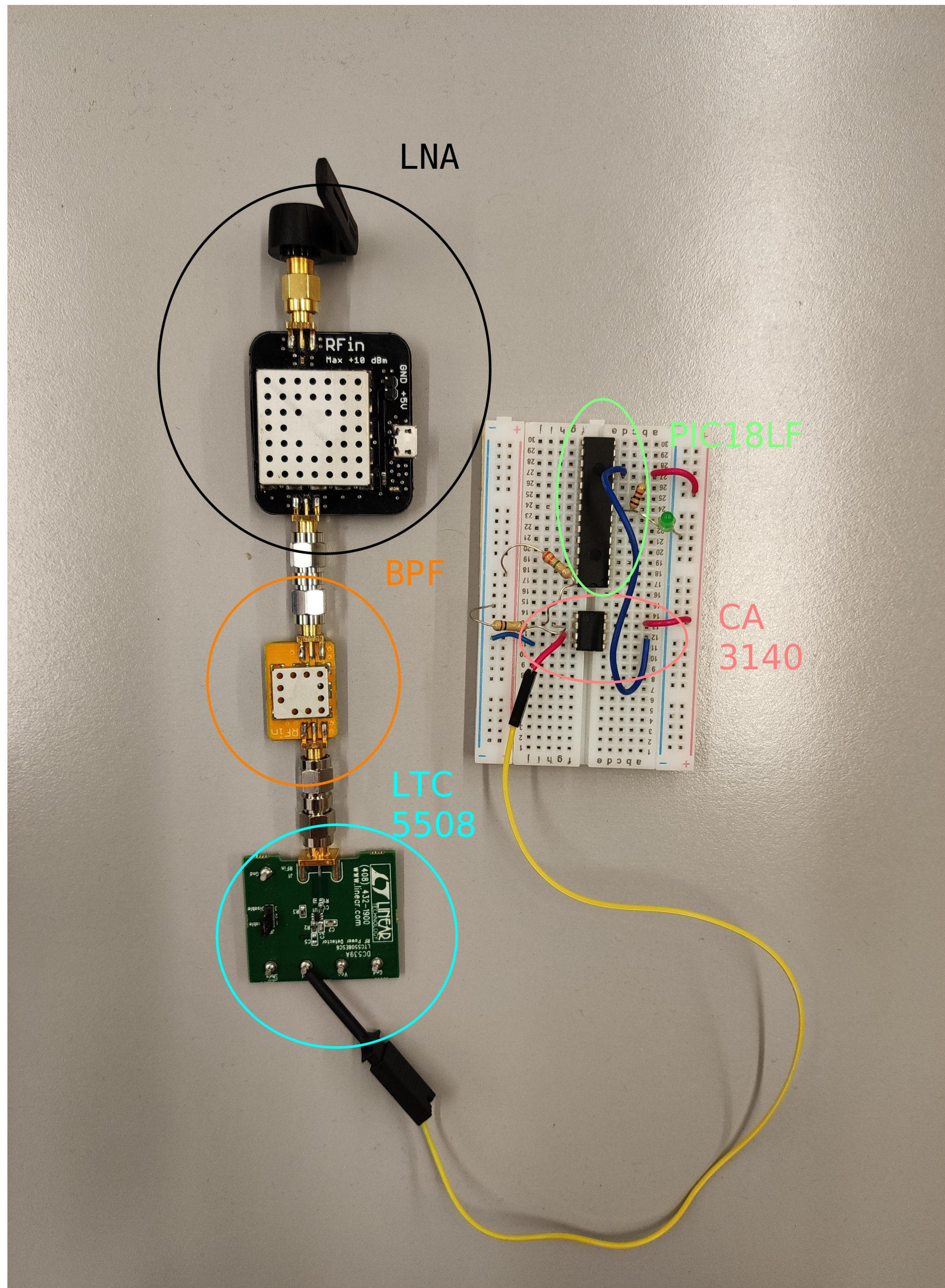


Figure 4.4: Final prototype of the WuRx, with the RF front-end.

correctly receive a WuS, it detects the start of frames bearing a possible WuS and correlates the bits received to the address of the WuRx. Finally, if the address from the WuS matches the device, the WuRx triggers the wake-up of the main radio.

The microcontroller for the baseband implementation is the PIC18LF23K22 [111], manufactured by Microchip. This 8-bit microcontroller has a remarkably low power consumption, consuming only 10 nA in sleep mode, and 3 mA when operating at

its maximum instruction clock, 4 MHz. Additionally, it does not require external oscillator circuits that could increase power consumption above the aforementioned figures. The rest of the characteristics of the PIC18LF23K22 are summarized in Table 4.2.

Table 4.2: Microcontroller parameters.

PIC18LF23K22[111]	Supply Voltage	3 V
	System Clock	16 MHz
	Instruction Clock	4 MHz
	Active current	3 mA
	Standby current	10 nA

However, even the low-power PIC18LF23K22 microcontroller consumes too much energy while operating in active mode. To conserve power the PIC18LF23K22 remains in sleep mode most of the time. When a WuS arrives, a transition to a high level occurs at the output of the RF front-end, this triggers the awakening of the microcontroller.

Nonetheless, the WuS frame structure is designed to allow this mode of operation.

WuR frame format with addressing for a low-power microcontroller

To support compatibility with IEEE 802.11 frames, the WuS payload must start 72 μs after the start of the frame. This equals the length of the shortest backward-compatible PLCP preamble included in the IEEE 802.11g release, which is considered for the WuTx presented in Chapter 3. This 72 μs preamble is received by the WuRx as a single high-level pulse, as the transmission power remains roughly constant. Additionally, the preamble duration is long enough to allow the local oscillator of the PIC18LF23K22 to stabilize after waking up. A stable instruction clock is required for the address sampling and correlation.

After waking up, the microcontroller needs to synchronize itself with the symbol period. For this purpose, a frame delimiter sequence is used. It begins with two "0" OOK symbols after the previous 72 μs pulse ends. The transition to a low level signals the microcontroller the start of the synchronization preamble. Although using only one "0" pulse would be optimal, two "0" pulses are used instead due to processing time constraints in the microcontroller. If this transition is not found, the microcontroller assumes that the frame is not a WuS and returns to sleep. After these, the frame delimiter is completed by a "1" OOK symbol, which causes a transition to a high level. The microcontroller starts sampling for OOK symbols encoding its address exactly 6 μs after this transition. Additionally, this last "1" OOK symbol allows the microcontroller to distinguish a WuS from shorter interfering

frames. The total preamble and the frame delimiter length is 84 μs. These are shown together in Fig.4.5.

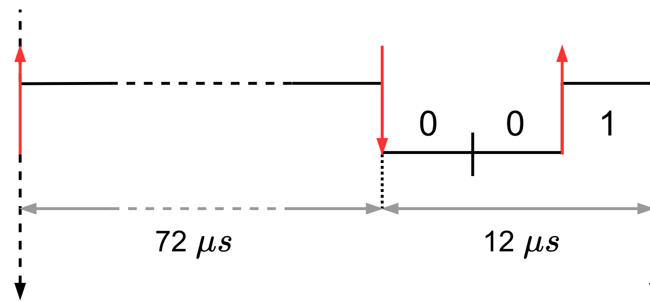


Figure 4.5: WuS headers. Red arrows highlight signal transitions.

The WuS frame is completed with a 16-bit address which is used to identify each WuRx in the local network. The duration of the address is 64 μs, being each address bit coded by a 4 μs OOK symbol. Therefore, the WuS length, adding the preamble and address fields, is 148 μs. The complete frame structure is shown in Fig.4.6.

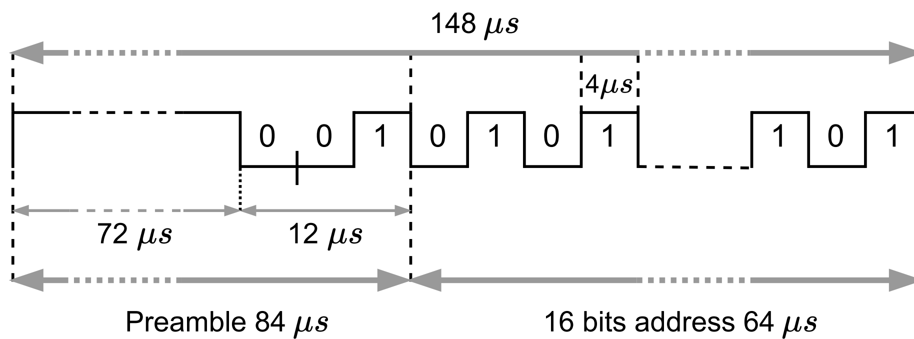


Figure 4.6: WuS frame structure.

Software implementation of the WuRx

As mentioned before, the WuRx microcontroller remains sleeping most of the time, awakening only if a possible WuS is detected. Consequently, the PIC18LF23K22 is configured to wake on a rising flank interruption. The behavior of the microcontroller was defined in accordance with the frame structure presented before, shown in Fig.4.7.

The microcontroller starts all frame WuS receptions in the Sleep state (state 1). After waking up, the microcontroller enters the Preamble Analysis state (state 2). Here the microcontroller spends 40 μs waiting for its clock to stabilize, and, afterward, it awaits a low-flank interrupt, signaling the start of the frame delimiter. If the frame delimiter is not successfully received, the microcontroller returns to the Sleep

state. If the delimiter is received correctly, the microcontroller enters the Sampling Bit 0 state (state 3), and, if the received bits can be matched to the WuRx address, it advances up to the Sampling Bit 15 state, and triggers the wake up of the main radio, afterward, it returns to the Sleep state (state 18). If any bit address bit does not match the expected value, the WuRx stops the correlation process and returns to sleep. Before returning to sleep, the microcontroller enters the Hold Sensing state (state 19) and waits an amount of time. this is done to avoid successive wake-ups on the same received frame.

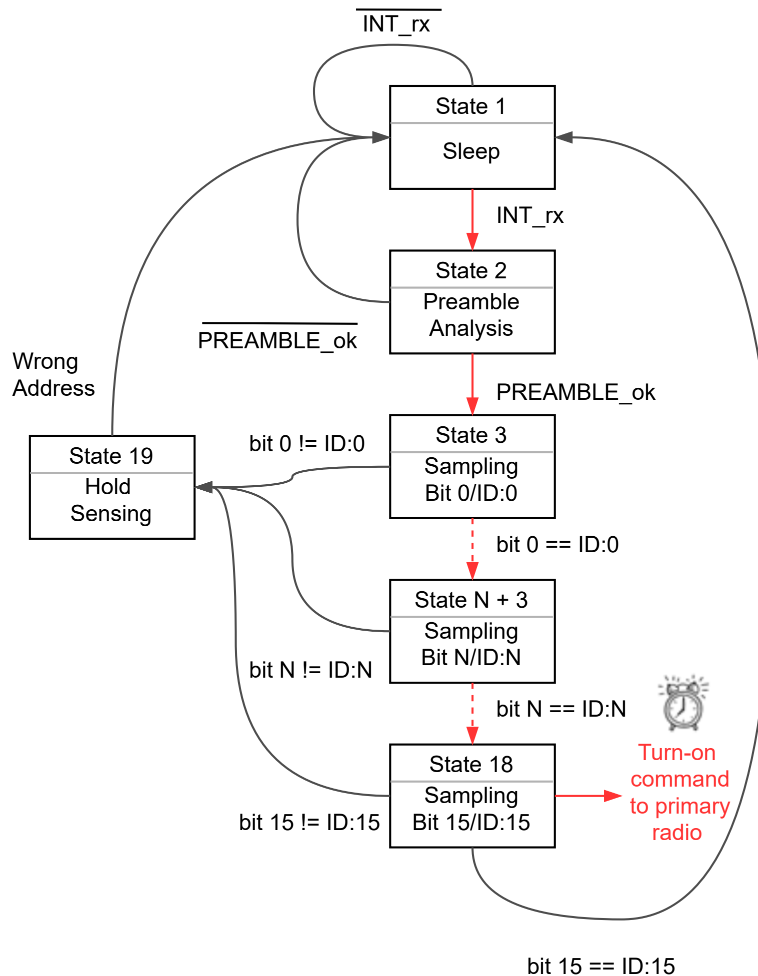


Figure 4.7: State diagram for the WuS processing.

The microcontroller needs to be fast enough to process the WuS in real-time. Fortunately, the PIC18LF23K22 supports various clock frequencies using its embedded clock sources. These go from 31 kHz to 16 MHz. However, the instruction rates are 4-fold lower than the clock rates, being the highest instruction rate attainable 4 MHz.

To minimize power consumption, the minimum clock rate capable of operating the WuRx in real-time is required. The most computationally intensive process, i.e., the address correlation procedure, is used to derive the lowest instruction clock compatible with real-time operation. The main loop of this process must be executed at least once for each symbol; therefore, its run time must be lower than 4 μs. It is as follows:

$addressArray \leftarrow WuRxAddress$
$i \leftarrow 0$
while $i <$length($address$) **do**
  $bit \leftarrow readPinValue$
  if $bit \neq address(i)$ **then**
    return $false$
  else
    $i = i + 1$
  end if
end while
return $true$

Thus, the microcontroller must sample the incoming bit, compare its value with the corresponding address bit obtained from memory, increment the loop counter, and finally, compare the counter with the length bound. These 4 operations, when compiled to the machine code, cost 18 instruction cycles. At the highest possible instruction rate, 4 MHz, the execution time of the loop is 4.5 μs. Therefore, it is not possible to meet the timing requirement of 4 μs with this design. The last operation, the comparison with the counter value, can be suppressed to save some valuable instruction cycles. By using manual loop unrolling, the loop was unrolled into 16 successive iterations. This allowed reducing the cost of each correlation cycle to 15 instructions, with a run time of 3.75 μs, below the 4 MHz limit.

This hard limit in the instructions available per symbol period made it impossible to synchronize the sampling instants using clock-derived interruptions. To align the sampling time with the symbol period, an additional NOP instruction was added (a placeholder instruction that does nothing and consumes 1 instruction cycle) to each sampling operation to lengthen the run time to 4 μs. In the same way, to meet the strict timing constraints, the state machine described in Fig.4.7 was implemented with a sequence of *if-else* statements, instead of using a traditional *do-while* loop with a switch statement.

4.2 Performance of the WuRx

The WuRx was evaluated using two different signal sources. First, one using an ideal 250 kbps OOK modulation, and, second, using the legacy-compatible IEEE WuTx described in Chapter 3, which also operates at 250 kbps. The correct reception of WuS from both WuTx was evaluated at 2 meters distance, in an indoor environment.

4.2.1 An SDR-based OOK WuTx

The implementation of the OOK WuTx was performed in an SDR using the GNU Radio SDR framework. For this design, two GNU radio blocks were created using the GNU Radio C++ API. These generate the frame format required by the WuRx at bit level. The first block adds bits corresponding to the address of the target WuRx. The second block adds the physical preambles. The first preamble is a long 72 μs pulse, which is represented by 18 “1” bits. The second preamble added is the frame delimiter sequence, comprising two “0” bits followed by one “1” bit.

Afterward, the resulting sequence is coded into OOK symbols, each one represented by a 4 μ OOK pulse. Finally, the spectrum of each pulse is widened to 20 MHz to emulate a WuS embedded into an IEEE 802.11 frame. This bandwidth is equal to the WuS sent by the legacy-compatible WuTx presented in Chapter 3. Although IEEE 802.11ba does also use a high-bandwidth OOK signal, it is only 4 MHz wide. The IEEE 802.11ba drafts propose a mechanism for signal generation that only use 16 subcarriers [85] [68]. In contrast, the legacy-compatible mechanism from Chapter 3 uses the 64 subcarriers present in the IEEE 802.11g OFDM PHY specification. Thus, the spectrum of the OOK WuS is widened to 20 MHz by multiplying each of the 4 μs pulses by a chirp sequence of 80 additional 5 ns pulses with an alternating “1” to “0” pattern.

Finally, the resulting GNU Radio flow diagram was configured to strobe a WuS directed to an address configured through the GNU Radio Companion GUI. This way, a WuS could be sent each 100 ms, emulating a wake-up procedure synchronized with IEEE 802.11 beacons. Transmission of a WuS as sent by the OOK WuTx is shown in Fig.4.8.

4.2.2 A legacy IEEE 802.11-based WuTx

The Linux-based legacy IEEE 802.11 WuTx described in Section 3.8.2 was adjusted to follow the framing specified in this chapter. To respect the preambles defined in Section 4.1.2, the WuTx was configured to use exclusively IEEE 802.11g OFDM. By adding extra padding to the IEEE 802.11 headers, which include the OFDM PLCP,

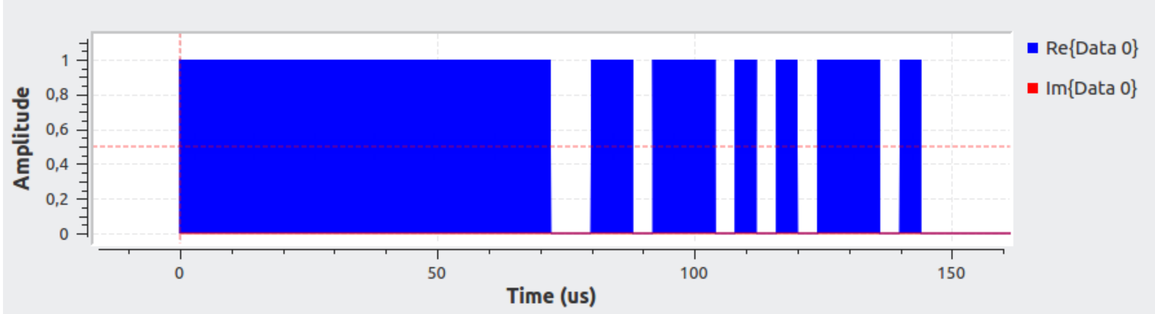


Figure 4.8: OOK WuS

and the MAC headers, it was possible to configure the WuTx to emit a single 72 μs pulse at the start of each frame. Following, the implementation was configured to append the frame delimiter, followed by a 16-bit WuRx address. Despite the modifications, the underlying IEEE 802.11 stack adds an additional footer. This corresponds to the checksum and padding defined by the legacy IEEE 802.11 MAC framing [84]. Nonetheless, this addition does not interfere with the reception of the WuS, which ends just after the last address bit. A WuS as sent by the legacy IEEE WuTx is shown in Fig.4.9.

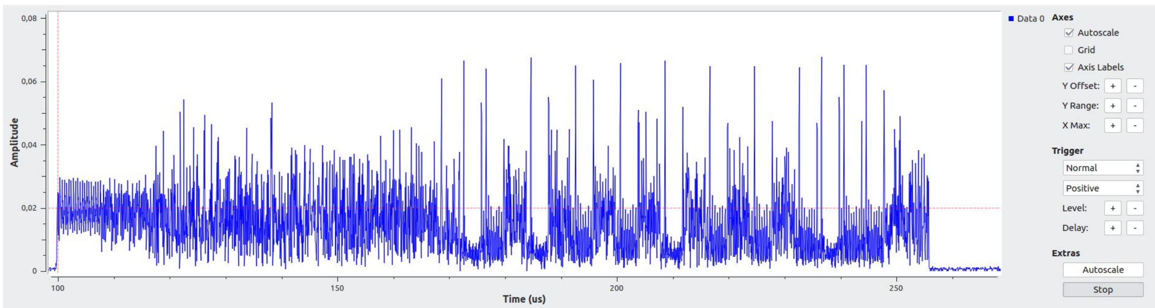


Figure 4.9: Legacy-compatible WuS to the WuRx address “0101010101010000”, as received with a USRP B200.

4.2.3 Results

The WuRx baseband was evaluated in an indoor environment to not introduce relevant propagation loss. To address the efficiency of the WuRx baseband, the power consumption of the WuRx was evaluated using three different scenarios. The WuTx used for evaluation on all scenarios was the OOK WuTx explained in Section 4.2.1. Nonetheless, the Linux legacy-based IEEE 802.11g WuTx was also evaluated to work with the WuRx.

The first scenario measures the energy consumption of the WuRx for correct WuS reception. For this purpose, this scenario includes a correct WuS sent to the expected WuRx address. The second scenario measures the energy consumption of

the WuRx when it receives a non-WuS frame, i.e., a non-WuR IEEE 802.11 frame. Consequently, it includes a frame with a mismatched frame delimiter. Finally, a third scenario evaluates the energy consumption of the WuRx when it receives a WuS addressed to another WuRx. This scenario explores a relatively optimistic case where the first bit of the address is mismatched. Nonetheless, due to the length of the hold sensing state, power consumption of addresses mismatched on later bits cause similar power consumption profiles. For each scenario, the resulting energy consumption was evaluated with a DC power analyzer, obtaining the duration of the wake-up event and the average current consumed by the WuRx during the wake-up. These results are evaluated at a voltage supply level of 3 V for the microcontroller to obtain the energy consumed.

Receiving a correct WuS, as evaluated in the first scenario, consumed an average of 982.29 μA during 230.4 μs, obtaining a total of 679 nJ. When the incoming frame fails to follow the frame delimiter, the detection of the WuS stops at an earlier time point, thus, reducing energy consumption. Moreover, the WuRx consumes less average current as it is performing less energy-intensive NOP operations in the hold sensing state. This way, in the second scenario, the average current consumption is 793.9 μA, with only 163.8 μs of wake-up time, resulting in 390 nJ of energy consumption. Finally, an address mismatch is a costly scenario in terms of energy consumption. The wake-up time includes the preamble sensing, as well as the time spent in the hold sensing state. However, the inclusion of the hold sensing state is required to avoid consecutive wakeups that would further increment the energy cost of these events. The energy consumption for the reception of a mismatched WuS is 753 nJ. All these results are showcased in Table 4.3.

Table 4.3: WuRx baseband energy consumption.

Scenario	Avg Current (μA)	Avg Wake-up time (μs)	Energy consumed (nJ)
Correct WuS	982.29	230.4	679
Incorrect frame delimiter	793.9	163.8	390
WuS with mismatched address	1090	230.4	753

4.3 Conclusion

This chapter proposes implementing an affordable WuRx design with off-the-shelf components. This way, a low-power WuRx baseband capable of receiving OOK signals at a 250 kbps data rate was designed and implemented. The WuRx baseband was evaluated using a provisional non-low-power RF front end. Nonetheless, the baseband is designed to operate in low-power environments. The microcontroller-based baseband implements preamble detection and address matching using an ex-

tremely low-power 8-bit microcontroller to its highest potential. It was found capable of achieving low-power operation, with an idle current consumption of 10 nA. Additionally, the energy expenditure per WuS triggered wake-up was found low, at 679 nJ per WuS received. Moreover, the energy consumption per false wake-up was found lower, at 390 nJ, due to the preamble detection procedure implemented in the baseband.

The results of this chapter point in the direction that a high bitrate WuRx baseband constructed with off-the-shelf parts can perform well in low-power environments. Additionally, the low-power WuRx baseband presented was found robust to interference, being able to operate with non-WuS frame transmissions occurring in the same site. Nonetheless, evaluating the performance of the baseband under non-WuS traffic profiles requires further study. The WuRx developed for this chapter validates experimentally the WuTx presented in Chapter 3 with a physical WuRx, thus, completing a WuR system compatible with devices implementing legacy versions of the IEEE 802.11 specification.

Thus, the results from this chapter can be used jointly with those of Chapter 3 to provide a high bit rate WuR system. Such a system allows the transmission of more complex signals that can go beyond simple wake-up calls. For example, WuR frames bearing data can be used to provide efficient data transmission between heterogeneous devices. Moreover, such a WuR system can also enable direct communications between WSN devices and non-energy-constrained devices. Thus, enabling gateway-less interaction between non-compatible WSN devices and contributing to the achievement of convergence between most technologies applied to WSN developments.

This chapter was published as an article to *IEEE Wireless Communications Letters* (Q1, Telecommunications 17/90 [102]) on the 1st of July 2019.

M.Hussein, E.López, M.Cervià and A.Calveras. Design and Implementation of a Wake-up Radio Receiver for Fast 250 kbps Bit Rate. *IEEE Wireless Communications Letters[73]*

Chapter 5

Enabling Wireless Convergence with WuR

Nowadays, several wireless communication solutions targeted at low-power devices (e.g. sensors, actuators) compete for market dominance. As a consequence, fragmentation slows the deployment of the Internet of Things. However, there is a window of opportunity to use a WuR to bridge the gap between incompatible technologies.

The previous chapters present a high rate WuR solution, that is capable of sending arbitrary binary payloads. In addition to using this capability to address WuS, devices can use high-rate WuR hardware as a secondary means of communication. Since WuR compatibility is unrelated to main radio hardware compatibility, communication among heterogeneous devices can be another application area for high-rate WuR. Thus, WuR offers a way to bridge the gap between heterogeneous devices, allowing them to interact directly without requiring a gateway.

Therefore, WuR can contribute to the already existing landscape of CTC solutions. In this chapter, a WuR-CTC solution will be presented and evaluated in a physical testbed, using WuTx and WuRx solutions derived from those already presented in Chapters 3 and 4. Importantly, the addition of WuR to devices enables asynchronous communications and, in a clear improvement over existing CTC solutions, provides energy savings.

The content presented in this chapter is structured as follows: Section 5.1 exposes the rationale behind this development and motivates WuR-CTC; Section 5.2 explains the communication scenario envisioned, as well as the characteristics of the desired WuR-CTC solution; Sections 5.3, 5.4, and 5.5 detail the WuR solution adopted and its implementation in the testbed; Section 5.6 presents the experimental results obtained from the testbed operation; finally, Section 5.7 concludes the study and points out next research issues.

5.1 Contribution of WuR to the CTC state-of-the-art

As previously stated in Section 2.2, WSN have motivated the extension and development of a considerable number of solutions that suit their set of requirements (e.g., low-power consumption, low-cost and increased range). However, these heterogeneous solutions are not interoperable, most of them defining incompatible physical layer implementations. This precludes direct communications between heterogeneous devices, becoming one prominent cause of the often-cited IoT fragmentation problem.

The issue of compatibility is mitigated through the use of gateways, devices that can translate between incompatible protocol stacks. Nonetheless, as discussed in Section 1.1, the use of gateways does not come without caveats [11]. Those include reliability and performance issues. Therefore, there is a research opportunity in investigating ways to achieve direct communications between heterogeneous devices. This area is CTC research, which concentrates on finding techniques that allow direct interaction between non-compatible devices. Fig.5.1 shows the comparison between CTC and gateway use on an example network with IEEE 802.11 and IEEE 802.15.4 devices.

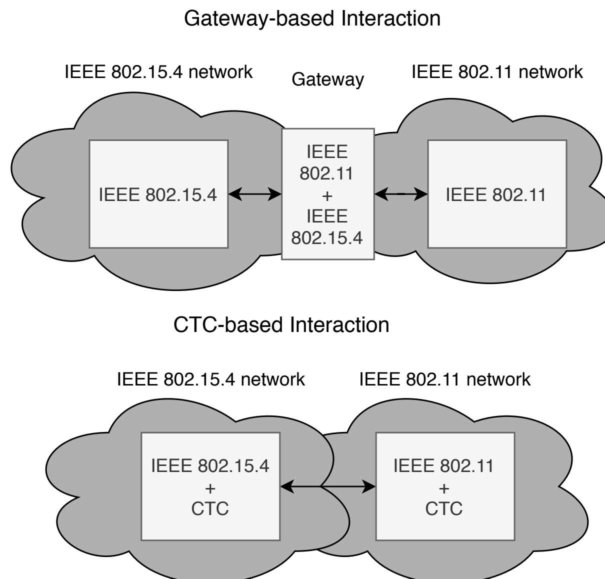


Figure 5.1: Examples of communication scenarios comparing gateway-based interaction with CTC-based interaction

Two approaches have been previously evaluated in the CTC literature to provide direct communications between non-compatible devices: first, re-purposing CSMA mechanisms and, second, signal emulation. Following, the state-of-the-art of each CTC approach is presented, along with the advantages and disadvantages of various CTC approaches.

CSMA-based CTC approach

This approach repurposes the coexistence mechanisms that WPAN and WLAN protocols implement for its use with CTC. The most widely deployed WLAN and WPAN technologies that have appeared in the last decades operate on the 2.4 GHz ISM band [13, 23, 112] using CSMA with Collision Avoidance (CSMA/CA) as their MAC approach. CSMA-enabled stations avoid collisions by sensing the channel before starting a transmission. Thus, CSMA- equipped stations can detect transmission bursts initiated by non-physically compatible devices. Moreover, they can also determine the duration of an incoming transmission burst by measuring its length. Therefore, CSMA/CA-based CTC encodes binary data using signal features that can be received by any CSMA-enabled station, such as the frame length and the inter-frame interval. This approach to CTC was first proposed in [113], which presents a proof-of-concept implementation supporting unidirectional communications between IEEE 802.11 based transmitters and IEEE 802.15.4 receivers. [113] uses several IEEE 802.11 frame lengths as codewords to build a dictionary for CTC, reaching a throughput of 1.6 kbps in a best-case scenario. Later, the idea of using frames to encode information was generalized to use the interval between frames to code information in [114] and [115].

Low spectral efficiency and throughput limit the usability of CSMA-based CTC. To maximize spectral efficiency, authors of [114] proposed piggybacking CTC communications on existing data fluxes. In [114], binary data is modulated by changes in the transmission intervals of frames from pre-existing TCP/UDP fluxes sent by an IEEE 802.11 station. Although [114] still presents a relatively low bitrate, it reuses frames from already present data streams with minimal disruption. However, it also requires the presence of an existing data flow. If frames need to be sent explicitly to convey a CTC message, the spectral efficiency advantage is lost.

To send CTC data without depending on other data flows to piggyback messages, [115] uses slight variations of the interval between beacon frames to send information. This allows [115] to enable bidirectional CTC between IEEE 802.11 and IEEE 802.15.4. Moreover, [115] it is also capable of sending information unidirectionally from Bluetooth transmitters to IEEE 802.15.4 and IEEE 802.11 receivers. The bitrate of [115] is determined by the rate beacon frames are sent, which is relatively low in most technologies, for example, the default IEEE 802.11 beacon rate is 100 ms. Moreover, to ensure reliability, several beacon transmissions are required to encode a symbol. As a consequence, the throughput achieved a [115] data flow is around 30 bps. Nonetheless, several parallel data flows can coexist by sending beacons in a non-overlapping manner. This way, the aggregated throughput of the system can be increased to up to 1400 bps.

Despite improvements in spectral efficiency, the main drawback of CSMA-based CTC is that it uses entire frames as symbols. Consequently, CSMA-based CTC only

encodes a few bits of information in each of them. As a result, CSMA-based CTC throughput will always be low when compared to the solution used by the respective main radio. Moreover, the symbols used by CTC (frame lengths, or inter-frame intervals) can be spontaneously generated by non-CTC network activity, causing false detection events. The interference caused by those needs to be mitigated either by using codification or by performing retransmissions, further reducing CSMA-based CTC throughput. In a realistic scenario with the presence of interference, the maximum throughput achieved by CSMA-based CTC is close to 3.1 kbps, with unidirectional CTC from BLE to IEEE 802.11 [116]. Nonetheless, as [116] reuses existing Bluetooth data fluxes, its spectral efficiency remains high.

Table 5.1: Characteristics of CSMA-based proposals referenced

Reference	Method	CTC	Maximum Throughput
[113]	Burst length	326bps	Unidirectional, IEEE 802.11 to IEEE 802.15.4
[115]	Burst interval	30 bps [a]	Bidirectional, IEEE 802.11 and IEEE 802.15.4. Unidirectional, Bluetooth to IEEE 802.11 and IEEE 802.15.4
[114]	Burst interval	760bps [b]	Bidirectional, IEEE 802.11 to IEEE 802.15.4
[116]	Burst power	3.1 kbps [b]	Unidirectional Bluetooth to Wifi

[a] Possibility of parallel streams. [b] Embedded into non-CTC traffic.

Emulation-based CTC approach

The emulation-based CTC approach can overcome the low throughput associated with CSMA-based CTC. Emulation tries to reproduce the signaling defined by a wireless solution using a transmitter implementing another wireless solution. This approach appeared in the literature with WEBee [117], which defines a procedure to emulate IEEE 802.15.4 signals with an OFDM transmitter following the IEEE

802.11 standard. WEBee provides a unidirectional link between a transmitting IEEE 802.11 station and one or more receiving IEEE 802.15.4 stations at IEEE 802.15.4 full data rate, 250 kbps. Nevertheless, inaccuracies on the emulation of IEEE 802.15.4 cause degraded range and a frame error rate close to 40%, which the authors mitigate with a repetition code. However, coding limits the effective data rate to 63 kbps. Subsequent proposals [118, 119] present incremental improvements over [117]. The first, [118] proposes modifications to [117], enabling it to use the IEEE 802.11 5 MHz down-clocked mode to reduce the bandwidth disparity between the IEEE 802.11 transmitter and the IEEE 802.15.4 receiver. As a consequence, IEEE 802.15.4 receiver SNR is increased since the emulated signal falls mostly inside of its filtering bandwidth. This, coupled with optimizations in preamble coding, reduce the frame error rate and increase range. The second, [119] uses physical-layer coding to further reduce the frame error rate. The code works by effectively splitting an emulated symbol into two symbols, using symmetry between IEEE 802.15.4 chip sequences. The solution ensures that, between the two transmitted symbols, the chips encoding the expected symbol avoid boundary errors. As a result, [119] can achieve a throughput 3 times as high as [117]. With CTC by emulation has been also introduced to other technologies, including Bluetooth to IEEE 802.15.4 [120] and Bluetooth/IEEE 802.15.4 to LoRa [121].

However, in the implementations found in the literature, emulation works only one way. For example, while an IEEE 802.11 can emulate the signaling used by IEEE 802.15.4, the reverse link is not possible. This also appears in [120], for IEEE 802.15.4 and Bluetooth, and, finally, in [121], a reverse link, from LoRa to either IEEE 802.15.4 or Bluetooth is not found. Therefore, emulation is not able to provide a truly bidirectional link, thus, limiting its usefulness in most communication scenarios. Nonetheless, [122] introduced support for reliable communications by using confirmation messages. In this proposal, the authors use emulation-based CTC to send high-rate data streams from IEEE 802.11 to IEEE 802.15.4 using WEBee [117] with low-rate confirmation messages sent from the IEEE 802.15.4 nodes. These confirmation frames are received by the IEEE 802.11 nodes using raw access to the channel samples, a feature that is not accessible in commodity IEEE 802.11 devices. Still, the lack of symmetric bidirectional connectivity relegates emulation-based CTC to one-way communications or data dissemination via broadcast and multicast.

5.1.1 Overcoming CTC limitations with WuR-CTC

Due to the aforementioned limitations, current CTC developments offer either one-way communication, or a data rate several orders of magnitude lower than state-of-the-art WSN solutions. As a consequence, CTC in WSN is relegated to those use cases enabled by one-way, low-rate communications, such as data dissemination through broadcast.

Table 5.2: Characteristics of Emulation-based proposals referenced

Reference	CTC	Data Rate	Throughput
[117]	Unidirectional, IEEE 802.11 to IEEE 802.15.4	n/a	63 kbps[a]
[118]	Unidirectional, IEEE 802.11 to IEEE 802.15.4	n/a	63 kbps[b]
[119]	Unidirectional, IEEE 802.11 to IEEE 802.15.4	n/a	3 times higher than [117][c]
[120]	Unidirectional, Bluetooth to IEEE 802.15.4	n/a	220 kbps
[121]	Unidirectional, Bluetooth and IEEE 802.15.4 to a non-standard LoRa receiver	3.2 kbps	n/a
[122]	Unidirectional, IEEE 802.11 to IEEE 802.15.4 [d]	n/a	10.67 kbps

[a] [117] and [118] use a repetition code to achieve reliability. [b] [118] offer longer range than [117], through physical layer optimizations and preamble coding. [c] [119] manuscript refers its throughput to be 3 times higher than WeeBee, however, WeeBee throughput is only displayed in Fig.34 of [117]. No exact figure is provided in the manuscript. [d] [122] supports reliable communications through acknowledgment messages sent from IEEE 802.11 stations through CSMA-based CTC.

At the moment, CTC cannot contribute to enabling heterogeneous mesh networks for WSN. To allow that, CTC must be able to support bidirectional communications and, preferably, at a rate that is close to those offered by the main radio of popular WSN solutions. Thus, providing high-rate bidirectional CTC could open up at least three new use cases for CTC while, at the same time, contribute to reducing fragmentation in the WSN space. The first of those use cases is direct interaction between end-user devices and a WPAN network. For example, a mobile IEEE 802.11 terminal could configure, actuate, and obtain sensor data from an IoT network composed exclusively of IEEE 802.15.4 WuR-enabled devices. Another relevant use case

concerns heterogeneous network architectures. These appear as a consequence of the rapid pace of innovation in the IoT space, where, after the initial deployment of an IoT network, subsequent extensions may use other wireless solutions due to improvements in the state-of-the-art. With bidirectional CTC, the devices forming these heterogeneous networks can freely communicate, regardless of their respective wireless implementations, without requiring a gateway device translating. For example, bidirectional CTC could enable the expansion of an IEEE 802.15.4 network composed of specialized low-power devices with higher throughput IEEE 802.11ba devices, which could go on to form the backbone of the network. Thus, improving the overall performance of the resulting network. Simultaneously, bidirectional CTC can be used to improve coexistence between the different wireless solutions present in the network. With it, a heterogeneous network can adapt to changes in the environment by coordinately switching the operating frequencies of all device radios. Of course, these examples can be generalized to any arbitrary combination of non-compatible wireless solutions.

This contribution proposes the application of WuR to CTC to enable high-rate bidirectional CTC. WuR can provide CTC that is high-throughput, as well as bidirectional, through the assistance of the dedicated WuR hardware elements. These can be compatible, even if deployed in devices with divergent main radio implementations. Additionally, WuR-enabled devices can use those to provide higher throughput digital communications. For example, WuR implementations proposed for IEEE 802.11 devices offer up to 250 kbps of data rate [103, 105]. This rate is more than 50 times faster than the fastest CSMA-based CTC development [116] and equivalent to the data rates enabled by emulation-based CTC. Moreover, WuR also enables low-power operation, which is a requirement on most WSN use cases. Thus, with these advantages in mind, this contribution presents the WuR-CTC concept to provide direct communications among heterogeneous devices.

Nonetheless, the current state of WuR research is relatively concerning in regards to compatibility among WuR implementations. Instead of standardizing a single WuR solution compatible with a wide range of IoT applications, the current industry trend is to develop specific WuR solutions, related to existing WPAN and WLAN solutions. The first example of this approach is IEEE 802.11ba [68], which defines a WuR solution exclusively aimed at IEEE 802.11 devices. These specific WuR solutions could become non-compatible with each other. As a consequence, it is critical to highlight the harmonization opportunity that new WuR developments bring. There is a window of opportunity for reducing fragmentation in the IoT environment with WuR, however, only if the actors in the field harmonize nascent WuR implementations.

5.2 A proof-of-concept WuR-CTC testbed

To demonstrate the feasibility of WuR-CTC, its implementation in a testbed is proposed. Nonetheless, to showcase CTC, the testbed must feature a set of characteristics, which are determined according to a WSN-based communication scenario.

5.2.1 Testbed Characteristics

The communication scenario considered for the testbed involves symmetric and reliable point-to-point communications, the mode of communication required for WSN mesh networks. This showcases the utility of bidirectional CTC over unidirectional CTC solutions.

Most WSN devices operate in shared frequency bands, i.e., both IEEE 802.15.4, IEEE 802.11, and Bluetooth can operate in the 2.4 GHz ISM band. Therefore, to show its applicability, the testbed needs to consider coexistence with other solutions operating at the same frequency band.

Furthermore, to present the added value of WuR-CTC, and to better support low-latency mesh networks, time-sensitive interactions will be considered. Point-to-point communications should occur with a latency lower than other bidirectional CTC solutions. For this purpose, a target latency lower than of 100 ms is set.

Current WSN developments allow nodes to communicate over IP. Both between them, and with the wider Internet. Consequently, the WuR-CTC testbed envisioned should to be capable of supporting an IP-based communications stack (e.g., an IP stack with a 6Lo adaptation layer[123]).

Moreover, the testbed needs to feature devices that are representative of the current trends in WSN hardware, and that the fulfill requirements of WSN applications. In this way, prospective WuR-CTC devices are:

- Heterogeneous, operating at the 2.4 GHz ISM band which is supported by the physical layers of most WPAN and WLAN standards.
- Battery-friendly. Low-power, paired with low activity ratio. This capability can be enabled or supported by WuR.
- Limited computational capability, i.e., microcontroller hardware.
- Off-the-shelf available. Both in the devices themselves and their components.

Derived from the above described, jointly with the device profile, the following requirements for the WuR-CTC testbed were identified.

- To reduce the power consumption of the device, the WuRx must be implemented by separate low-power hardware that is available off the shelf. To implement WuR with off-the-shelf parts, the WuS needs to be coded with OOK. Moreover, OOK is used throughout most of the WuR implementations in the literature[124].
- To achieve bidirectional and reliable communications in a point-to-point scenario, the testbed needs to use an Automatic Repeat Request (ARQ) mechanism and two-way addressing.
- A CSMA-CA implementation is required to coexist with other WPAN and WLAN standards operating on the same unlicensed frequency band.
- To provide compatibility with IP communications, the testbed needs to provide a payload size that is sufficient to achieve low fragmentation overhead with a prospective 6Lo implementation. Payload size should be close to those defined by other WPAN solutions with existing 6Lo implementations such as IEEE 802.15.4.

5.2.2 Standards implemented

This chapter presents the WuR-CTC concept as standard agnostic. In this way, the WuR-CTC implementation presented here is designed to showcase communications between different WPAN and WLAN solutions by the means of a compatible WuR implementation. Nonetheless, to allow for the prompt evaluation of WuR-CTC, the testbed incorporates devices implementing two wireless communication standards: IEEE 802.11 and IEEE 802.15.4.

Both IEEE 802.15.4 and IEEE 802.11 standards are massively deployed for industrial and domestic use cases, however, they are not interoperable. Their PHY are defined with different modulations, channel assignments, physical framing, bandwidth, and transmission rates. Despite these differences, both can operate at the 2.4 GHz ISM frequency band. Moreover, commodity devices implementing both of these standards are available for development. Furthermore, it is possible to generate OOK signaling at a high symbol rate (250 kBd) with the main radios of devices implementing both standards.

A pseudo-OOK modulation can be generated with IEEE 802.11g/a transmitters by using the technique presented in Chapter 3. Although IEEE 802.11ba also defines a WuTx implementation, the specification is still to be ratified and, at the time of writing of this document, there are no commercially available IEEE 802.11ba embedded devices that could be incorporated in the testbed. Therefore, using the legacy compatible IEEE 802.11 WuTx implementation enables the evaluation of

WuR-CTC with a testbed based on commercially available commodity IEEE 802.11 hardware.

The legacy-compatible WuTx implementation requires the transmitter PHY to use a predictable scrambler sequence (a condition that occurs in several IEEE 802.11 devices [117, 125]). Moreover, the requirements presented in Section 5.2 call for a low-power embedded device. The ESP-32 WuTx implementation, presented in Section 3.8.3 fulfills these requirements.

The generation of OOK is not possible with all standard-compliant IEEE 802.15.4 devices. However, several devices implementing IEEE 802.15.4 include reconfigurable radio hardware that can generate OOK signals while maintaining its operation as a standard-compliant IEEE 802.15.4 device. One of these is the EFR-32MG12 [126], which can transition its main radio between IEEE 802.15.4 and OOK in less than 100 μs. This reduced switching time allows for time-sensitive communications, as required in Section 5.2. Therefore, an EFR-32MG12 can operate as a part of an IEEE 802.15.4 network, reconfigure its radio to transmit with OOK, send a frame and, finally, reconfigure its radio back to IEEE 802.15.4. The same can indeed be done in some Bluetooth devices [36] incorporating a proprietary radio mode, which can also generate OOK. Nonetheless, IEEE 802.15.4 was selected for this purpose due to familiarity with the underlying protocol and previous experience with the development environment.

Thus, the WuR-CTC testbed will feature an IEEE 802.11-enabled device, the ESP-32, and an IEEE 802.15.4-enabled device, the EFR32MG12.

5.3 PHY layer specification for the WuR-CTC testbed

As in previous chapters, the PHY specification for the WuR-CTC testbed proposes using the main radio as WuTx and a separate dedicated low-power receiver as WuRx. The advantage of this approach is that it reduces device complexity and implementation cost since the only required addition is the WuRx.

First, the testbed WuR-CTC PHY uses a single transmission rate of 250 kbps with OOK symbols. This rate is supported both in the previously presented legacy IEEE 802.11 WuTx, and the IEEE 802.15.4-compatible WuTx [35]. This relatively high transmission rate is lower than the minimum 1 Mbps bitrate defined by IEEE 802.11 or Bluetooth. However, it matches the data rate offered by the popular IEEE 802.15.4 WPAN standard.

To aid in synchronization and allow the WuRx to retrieve the WuR-CTC PSDU, as in most wireless protocol specifications [68], [112], [23] the WuR-CTC Physical

Protocol Data Unit (PPDU) includes a PHY header that is added before the WuR-CTC PSDU.

5.3.1 WuR-CTC PPDU structure

The WuR-CTC PPDU, partially based on the format presented in Chapter 4, includes two components: a preamble sequence and a frame delimiter. The complete structure of the WuR-CTC PPDU, including both of these fields, and the PSDU is shown in Fig.5.2.

The preamble sequence consists of 10 symbols, alternating "1" symbols and "0" symbols (5.1). The preamble also maintains a constant average amplitude that the WuRx uses to calibrate its OOK symbol detection level. Additionally, transitions from high to low can be used to synchronize the WuRx with the symbol period. This uncomplicated design allows very simple WuRx based on low-power microcontroller hardware without dedicated correlators.

$$\{1, 0, 1, 0, 1, 0, 1, 0, 1, 0\} \tag{5.1}$$

After it, a frame delimiter field composed of two "1" symbols (5.2) is appended.

$$\{1, 1\} \tag{5.2}$$

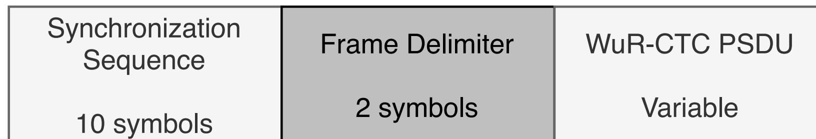


Figure 5.2: WuR-CTC PPDU structure.

5.3.2 Encapsulation of the WuR-CTC PPDU

The WuR-CTC solution is designed to operate with commodity transmitters. These are not fully reconfigurable since they already implement other wireless solutions. As a consequence, complete control over the framing of the WuR-CTC PPDU cannot be achieved. For example, the IEEE 802.11 transmitter adds its own PPDU, including training sequences and preamble fields before the WuS PPDU can be sent. Therefore, the WuR-CTC PHY needs an encapsulation mechanism to embed the WuR-CTC PPDU inside the frame structure followed by the radio operating as

WuTx. This encapsulation mechanism is designed to allow any WuRx to retrieve the WuR-CTC PPDU regardless of which encapsulation format has been used.

Therefore, for the testbed, two encapsulation formats, compatible at reception, are defined. The first is used by IEEE 802.11-based WuTx [125] and the second is used by IEEE 802.15.4-compatible WuTx based on a reconfigurable radio [126].

IEEE 802.11-compatible encapsulation format

On an IEEE 802.11-based WuTx the WuR-CTC PPDU is encapsulated in the MAC Service Data Unit (MSDU) of a standard IEEE 802.11g frame. This differs from the first mechanism introduced in Chapter 3, which describes the encapsulation of the WuR-CTC PPDU directly in the IEEE 802.11g PSDU. However, as seen in Sections 3.8.3 and 3.8.2, using the MSDU provides wider compatibility to the method since most IEEE 802.11 implementations allow for raw byte access to the MSDU from software.

As in Chapters 3 and 4, the WuR-CTC PPDU is coded with OOK symbols at one bit per symbol, for a data rate of 250 kbps, being each of the WuR-CTC symbols a standard-compliant IEEE 802.11g OFDM symbol. Consequently, the bandwidth of the symbols comprising IEEE 802.11-encapsulated WuR-CTC PPDU is 20 MHz [125].

To reduce overhead, the IEEE 802.11 encapsulation uses the ERP-OFDM PLCP format, which enables frames sent by the IEEE 802.11-based WuTx to coexist with other non-WuR-CTC IEEE 802.11g OFDM stations. Thus, reducing the overhead of the encapsulation, in comparison with the solution presented in Chapter 4. The legacy-compatible PLCP used there is at least 72 μs, while the OFDM PLCP is only 16 μs long. As a drawback, the WuR-CTC IEEE 802.11 encapsulation is not compatible with the backward-compatible DSSS-OFDM preambles. However, this can be mitigated by using IEEE 802.11-defined mechanisms, as the CTS-to-self, in mixed networks. The resulting encapsulation of the WuR-CTC PPDU inside an IEEE 802.11g OFDM frame is represented in Fig.5.3.

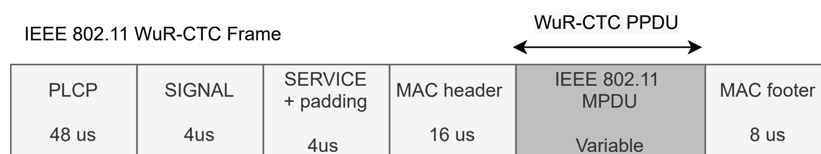


Figure 5.3: WuR-CTC PPDU encapsulation format for the IEEE 802.11g WuTx.

IEEE 802.15-4-compatible encapsulation format

The IEEE 802.15.4-compatible WuTx switches between IEEE 802.15.4 operation to OOK on demand. Before sending a WuR-CTC PPDU, the device must configure its reconfigurable radio from IEEE 802.15.4 to OOK symbols at 250 kbps. The bandwidth of the OOK symbols is slightly lower than 1 MHz, defined with a 20 dB attenuation, equivalent criteria that the one used to define the IEEE 802.11 20 MHz bandwidth.

To encapsulate the WuR-CTC PPDU, an OOK preamble including 18 OOK symbols with an alternating “1” to “0” pattern is prepended. This preamble delays the start of the WuR-CTC PPDU, thus allowing feature-limited WuRx to wake up and stabilize their local oscillators before the WuR-CTC PPDU starts. The encapsulation of the WuR-CTC PPDU inside a frame sent by a reconfigurable radio is represented in Fig.5.4.

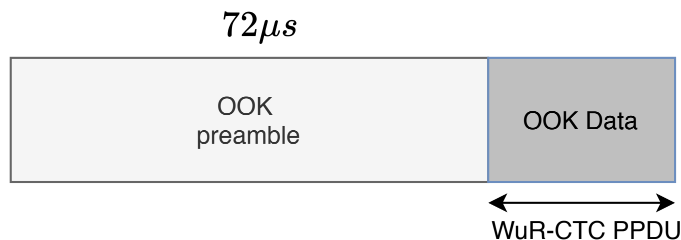


Figure 5.4: WuR-CTC PPDU encapsulation format for the IEEE 802.15.4 WuTx.

5.3.3 Determining the maximum WuR-CTC PSDU size

The maximum size of data that can be carried in the WuR-CTC PSDU is limited by the encapsulation format used. To support a 6Lo-based stack running on top of WuR-CTC, this figure must be made as high as possible. As mentioned before, WuR-CTC uses the OFDM symbols of the IEEE 802.11g MSDU as OOK symbols, each of those introducing a given amount of bytes that count to the IEEE 802.11 MTU of 2304 bytes. Consequently, IEEE 802.11-based encapsulation provides the most limiting scenario when compared to EFR-32MG12 [126].

This way, the maximum number of OFDM symbols that the IEEE 802.11g MSDU can carry is limited by the number of octets jointly coded in each OFDM symbol. This number, in turn, depends on the transmission rate used by the IEEE 802.11g radio. Lower transmission rates use a lower number of WLAN level bytes encoded per OFDM symbol. Consequently, these yield a higher number of available OFDM symbols for encoding WuR-CTC data since they require a higher number of OFDM symbols to reach the 2304 byte MTU. However, higher transmission rates yield a

lower maximum WuR-CTC MTU.

To find the available WuR-CTC PSDU size, the following formula can be used. The ($WuRCTC_{PSDU}$) is calculated as a function of the IEEE 802.11 maximum PSDU ($MSDU_{max}$) size in octets, the bits per OFDM symbol ($ODFM_{bits}$), and the overhead bits introduced by the WuR-CTC PPDU headers ($PPDU_{header}$), which corresponds to 12 bits (5.3).

$$WuRCTC_{PSDU} = \frac{ODFM_{bits}}{8 \cdot MSDU_{max}} - PPDU_{header} \tag{5.3}$$

The results of applying (5.3) for each transmission rate supported by IEEE 802.11g are displayed in Table 5.3. It lists the available WuR-CTC PSDU in function of the data rate used and the corresponding bit loading used by OFDM symbols. According to the results, to maximize the WuR-CTC PSDU size, the testbed uses the 6 Mbps transmission rate, which yields a maximum PSDU size of 94 bytes. However, using an alternative transmission rate of 24 Mbps (corresponding to the 16-QAM modulation) could provide increased range, as per the simulation results shown in Section 3.6. Nonetheless, the WuR-CTC PSDU would need to be reduced to 22 bytes.

A 96 byte frame size is similar to other WPAN standards, such as IEEE 802.15.4, that support IP-based stacks. This facilitates the implementation of 6-Lo in WuR-CTC, fulfilling one of the requirements formulated in the previous section.

Table 5.3: Maximum WuR-CTC PSDU Size

Throughput (Mbps)	OFDM symbol loading (bits)	WuR PSDU size (bytes)[a]
6	24	94
9	36	62
12	48	46
18	72	30
24	96	22
36	144	14
48	192	10
54	216	9

[a] Obtained by rounding the number of bytes to the lowest integer.

5.4 Link layer specification for the WuR-CTC testbed

The WuR-CTC link layer defines both MAC and Logical Link Control (LLC) sublayers, providing three services:

- Transmission medium sharing
- Addressing
- Reliability

The first is provided by the MAC sublayer, being the following responsibility of the LLC sublayer.

5.4.1 MAC sublayer

In most wireless regulatory domains, devices using the 2.4 GHz band are required to implement strategies to share the medium with other stations. In the 2.4 GHz ISM band regulations require either the use of a variant of the CSMA MAC mechanism or to severely limit the radio duty cycle of the station. To improve coexistence with other solutions and to comply with these regulations, WuR-CTC uses CSMA/CA as a coexistence method. The WuR-CTC WuTx implementations are heterogeneous and each of those implements CSMA using specific methods to ensure coexistence. Two CSMA implementations are used by the WuTx supported by the current WuR-CTC testbed.

1. IEEE 802.11 based WuTx encapsulate the WuR-CTC PPDU inside the MSDU of an IEEE 802.11g compliant frame, which is sent with a standard complying transmitter. As a consequence, no additional CSMA mechanisms are used, since the CSMA/CA coexistence mechanisms defined by IEEE 802.11 already apply.
2. IEEE 802.15.4 compatible WuTx send the WuR-CTC PPDU using the CSMA/CA method provided by the reconfigurable radio of the device [126]. Nonetheless, the configurable parameters of the CSMA/CA implementation are limited to a given set of values. Those are tuned to be as close as possible to those defined by IEEE 802.11 for mixed b/g networks[127] to optimize coexistence with IEEE 802.11 stations (see Table 5.4).

Table 5.4: WuR CSMA/CA Parameters for the reconfigurable radio

CSMA/CA parameter	Value	Unit
Slot Time	20	μs
Minimum Backoff Exponent	4	Slots
Maximum Backoff Exponent[a]	8	Slots
Retries	7	n/a
Detection Threshold[b]	-82	dBm
CCA Interval	14	μs

[a] Reduced from the 10 defined by the IEEE 802.11-2003 spec [127] to 8 due to hardware limitations on the EFR-32MG12 reconfigurable radio[126].
[b] The minimum allowed by EFR-32MG12 reconfigurable radio [126].

Latency and coexistence considerations

The WuTx is separated physically from the WuRx, which is implemented as a separate peripheral that is interfaced through a bus. This procedure adds additional latency to communication since there is one additional jump that received frames must travel. Low-power microcontroller hardware exposes feature-limited buses, such as I2C and SPI. Those provide low data rates, making it impossible to acknowledge frames with the same speed as other wireless solutions where the transmitter is integrated with the receiver. Those include the solutions used by the main radio of the devices featured in the testbed, IEEE 802.15.4 and IEEE 802.11. As a consequence, devices implementing these other solutions can send frames between a WuR-CTC frame is send and it is acknowledged.

For example, WuR-CTC implementations send ACK frames with a higher delay than the Short Interframe Space (SIFS) interval defined by the IEEE 802.11 standard. Therefore, the WuR-CTC ACKs do not gain priority over other frames sent by coexisting IEEE 802.11 stations and WuR-CTC ACKs need to contend with IEEE 802.11 traffic. This coexistence issue is also present in other wireless solutions such as Bluetooth [13] and IEEE 802.15.4 [23], which also use heterogeneous inter-frame spaces for their confirmation messages. For example, these two technologies use inter-frame spaces that are higher than the SIFS defined by IEEE 802.11 releases.

Nonetheless, with the IEEE 802.11-based WuTx this issue can be mitigated. The

IEEE 802.11 NAV can be used to reserve the channel with enough time to protect the WuR-CTC frame, as well as its acknowledgment. Protection can be achieved by using a CTS-to-self frame before starting a WuR-CTC exchange. However, this mechanism cannot protect transactions initiated by IEEE 802.15.4 WuTx. These do not include an IEEE 802.11 compatible radio capable of transmitting the frame fields required to set the NAV of IEEE 802.11 stations.

5.4.2 LLC sublayer

To provide LLC functionality, WuR-CTC implements a stop-and-wait ARQ mechanism that supports ACK piggybacking and uses 10-bit unicast addresses. With an address space capable of supporting up to 1024 devices per network, the protocol can fulfill its role as WuR demonstrator for relatively large networks. Addressing scope is local, thus device addresses could be reused in other networks, as long as there is no coverage overlap between them. If desired, devices can be segmented in sub-networks in an energy efficient manner by using a common prefix at the start of the address field.

As any WuR protocol, the WuR-CTC LLC protocol defines control messages necessary to awaken and sleep other stations. Nonetheless, in contrast to most WuR implementations, this protocol adds additional header fields that are used to support reliable general-purpose communications. Additionally, the protocol must enable WuR functionality, i.e., to wake up or put to sleep other devices. The header fields used are the minimum required to support the mode of communication chosen for the testbed, point-to-point reliable communication.

LLC headers

To support the aforementioned use cases, the WuR-CTC MAC Protocol Data Unit (MPDU) includes the header fields shown in Fig.5.5.

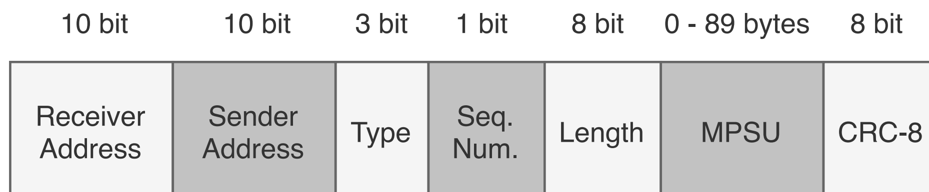


Figure 5.5: WuR-CTC MPDU.

The included fields are:

- Receiver Address: A 10-bit address that identifies the destination WuR-CTC node.
- Sender Address: A 10-bit address that identifies the source WuR-CTC node.
- Type: A 3-bit field with the following flags, ordered from the most significant bit to less significant bit, in Big Endian:

  1. Data flag.
  2. ACK flag.
  3. WuS flag.

  Using these flags, the protocol defines 6 types of frames:

  - "100": **DATA frame**. Includes data.
  - "010": **ACK frame**. Acknowledges a previously transmitted frame.
  - "001": **WAKE frame**. Awakens/sleeps other stations.
  - "110": **DATA+ACK frame**. Includes data and acknowledges a previous frame.
  - "101": **WuS+DATA frame**. Includes data for a sleepy station. This frame type does not keep the station awoken after the reception.

  The two remaining frame types are not used in the testbed and remain reserved for future use.

- Sequence Number: A 1-bit sequence number. The **ACK frame** sequence number must match the sequence number of the acknowledged frame.
- Length: An 8-bit length indicator. Contains the WuR-CTC MSDU length in bytes. Although, the maximum MSDU length supported is only 89, the length of this field is set to 8 bit due to implementation constraints on the EFR32MG12 reconfigurable radio [126].
- MSDU: The payload, which can range from 0 to 89 bytes. The maximum length corresponds to the maximum allowable WuR-CTC PSDU obtained from Table 5.3 minus 5 bytes, which account for the length of the LLC header and footer.
- CRC-8: A 8-bit length CRC field. It is calculated on the rest of the MPDU, including headers and payload, and appended at the end of the payload. The CRC-8 polynomial is used (5.4):

  $$p(x) = (x^8) + x^2 + x + 1 \tag{5.4}$$

  The use of a CRC-8 provides a minimal overhead, and, most importantly, it is fast on most micro-controllers, including ultra-low-power 8-bit ones, with an uncomplicated implementation based in a look-up table. While it is true that the selection of the CRC-8 polynomial could be optimized [128], this decision requires prior knowledge of the traffic profile.

LLC message flow

All frame types previously defined, except **ACK frames**, must be acknowledged by the receiver. No other frame can be sent before receiving the corresponding **ACK frame** or the expiration of the reception timeout. Moreover, all frame types must feature all the fields defined for the WuR-CTC MPDU as shown in 5.5.

Both IEEE 802.15.4 and IEEE 802.11 use a bounded ACK latency. The bounds are fixed to be lower than the standard inter-frame space, so, as a consequence, no other frames can be sent before a frame is acknowledged by its corresponding ACK. This allows the LLC sublayers of these protocols to skip the source address in the ACK since no other stations can send a frame at that moment. However, due to the latency added by interfacing, a tight bound on ACK latency cannot be applied to WuR-CTC.

It is important to notice that, in contrast to these wireless protocols, WuR-CTC **ACK frames** include source and destination addresses, as well as a sequence flag. Thus, WuR-CTC frames can be acknowledged unambiguously without strict timing constraints. This is not the case in other protocols, which omit the sender's address from the ACK frame. As a result, these protocols need to guarantee strict timing constraints to ensure that the acknowledgment frame is sent before any other frame. The SIFS used before an acknowledgment frame in IEEE 802.11 is an example of such a constraint.

WAKE frame payload data length must be 1 byte, and its value encodes the intent of the frame:

- A payload of **0xFF** is a **WAKE frame**, which indicates that the device should be woken up.
- A payload of **0x00** is a **SLEEP frame**, which indicates that the device needs to be returned to sleep.

Sending a **SLEEP frame** is recommended. However, a previously woken device will return to sleep automatically after a timeout passes. An exchange featuring the wake-up of a target device and communications using the primary radio can be found in Fig.5.6. The interaction starts with a **WAKE frame** (1), which is acknowledged by an **ACK frame** (2) when the receiver fully wakes up. After this, both devices can interact with their main radios. The exchange is finished by a **SLEEP frame** (3), which must also be acknowledged by an ACK frame (4).

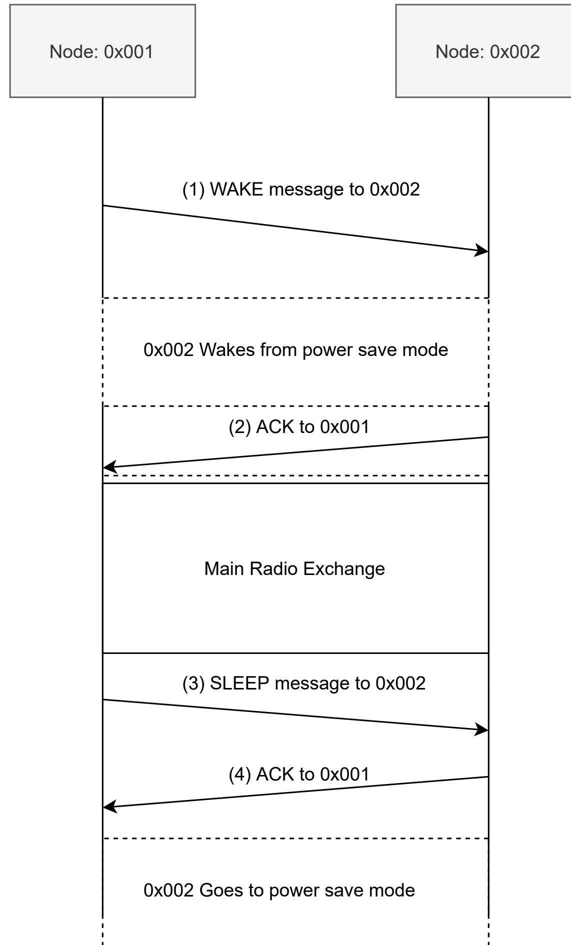


Figure 5.6: Traditional WuS interaction between two sleepy nodes.

However, WuR-CTC is not limited to just coordinating wake-ups. A station can send data to another with a single **WuS+DATA frame**, without neither a prior **WAKE frame** nor a **SLEEP frame**. This exchange conveys information but does not initiate any exchange with the main radio, nor force the receiver device to remain awake for any given time, as is the case with an interaction that starts with a **WAKE frame**. Fig.5.7 features this scenario, with a **WuS+DATA frame** (1), acknowledged with an **ACK frame** after the receiving device processes the incoming data.

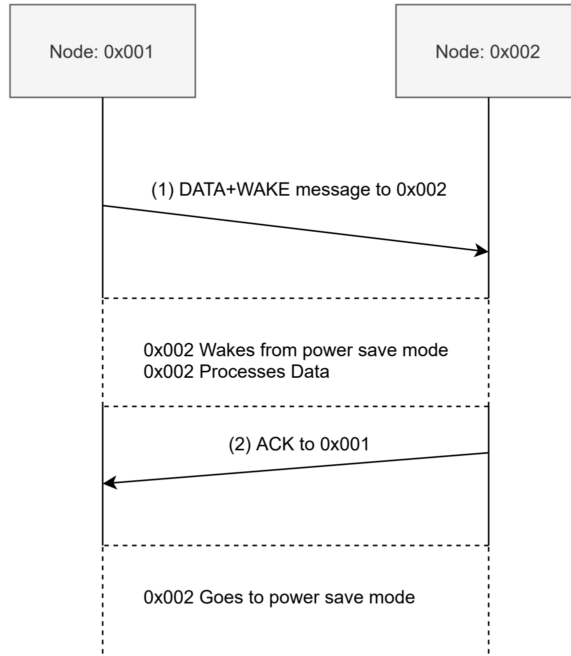


Figure 5.7: Sending data to a sleepy node.

Additionally, a Request/Response interaction between two sleepy WuR-CTC devices using their WuR hardware can also occur. This type of exchange is shown in Fig.5.8 for devices with assigned WuR addresses 0x001 and 0x002 on a specific situation featuring a request/response interaction using piggybacking. The interaction includes a **WAKE frame** (1) to awake the receiver device, one or more **DATA frames** (3), or **DATA+ACK frames** (4). Finally, the interaction finishes with a **SLEEP frame** (6) to indicate to the receiver that it can return to sleep.

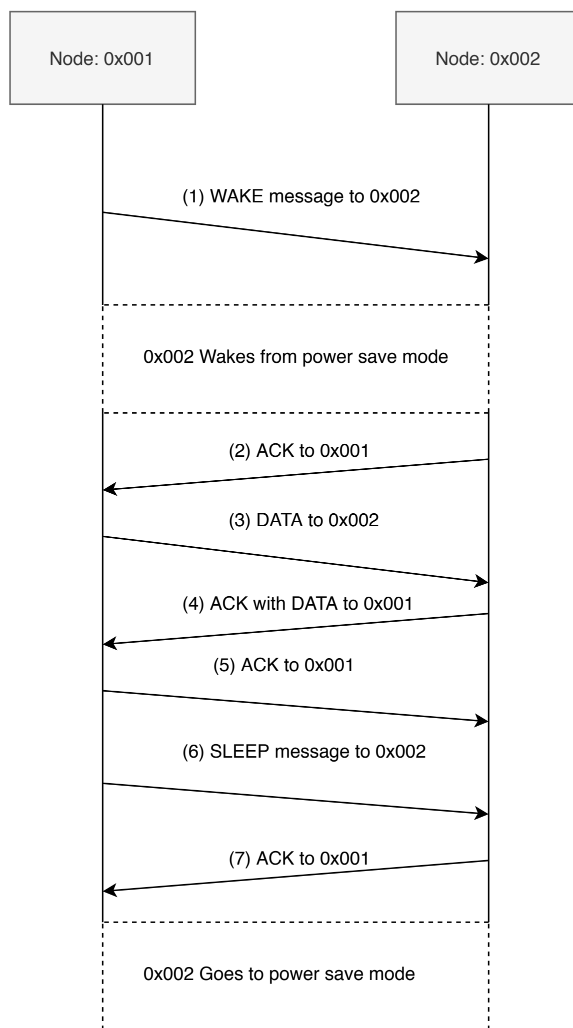


Figure 5.8: Request/response WuR-CTC communications between two sleepy nodes.

5.5 Device implementation for the WuR-CTC testbed

Devices from the testbed implement the aforementioned protocol layers, as well as a demo application to evaluate the achievable throughput of WuR-CTC. To obtain an effective testbed with efficiency, the developments previously produced in chapters 3 and 4 were modified and applied as necessary.

5.5.1 WuTx implementation for the WuR-CTC testbed

The WuTx implementation is done in software in both the ESP-32 and the EFR32MG12 platforms, allowing the reuse of the main radio of the device as WuTx. To support

both devices, the WuTx implementation differs according to the different approaches taken for OOK signal generation in both platforms. However, to allow for code reusability, both implementations present the same programming API, defined in the `ook_com.h` header file.

```c

#include "lib_conf.h"

/* initializes the WuTx*/
wur_errors_t ook_wur_init_context(void);

/*
 * Sends a WLAN frame containing a WuS data_bytes_len. It
 * requires that ook_wur_init_context has been called
 * previously to initialize the WuTx.
 */
wur_errors_t ook_wur_transmit_frame(uint8_t* data_bytes, uint8_t ...
    data_bytes_len);
```

Consequently, the WuTx defines an Application Programming Interface (API) that is used in the same way in both platforms WuTx implementations. First, `ook_wur_init_context` must set up the platform-specific facilities to start using the WuTx, and, afterward, `ook_wur_transmit_frame` is called to send frames embedding WuR-CTC PPDU. Both functions use the `wur_errors` enumeration to inform the caller of the result.

```c
typedef enum wur_errors{
    WUR_OK = 0,
    WUR_KO = 1
}wur_errors_t;
```

IEEE 802.11g WuTx

The generation of the OOK signal in the ESP-32 platform is based on previous work, performed in Chapter 3. Consequently, the implementation of Section 3.8.3 was particularized for the transmission rate of 6 Mbps, which provides the maximum WuR-CTC MSDU length, as can be seen in Table 5.3. Additional logic was added to introduce the headers and footers defined in Sections 5.3.

The WuTx implementation of the functions exposed by `ook_wur_init_context` comprises a single C source file, providing the required functions to set up the WuTx, as well as using it to send WLAN frames embedding a WuR-CTC signal. Additionally, the implementation also provides functionality to send WuR-CTC signals over a GPIO port output signal instead of the WLAN radio. The GPIO output consists of pulses as produced by OOK signals at the output of the RF front-end, thus, emulating the WuR-CTC protocol for WuRx baseband validation. This was

used as a debug tool for the development of the library.

The ook_wur_init_context implementation sets up the WLAN for the ESP-32 connecting it to an unsecured AP and configuring the data rate to the required. Following, the scrambler sequence is generated according to the data rate to be used and the seed configured to the library. Currently, both 6 Mbps, featuring BPSK, and 24 Mbps, featuring 16-QAM, are supported. The ook_wur_transmit_frame implementation embeds a WuR-CTC frame containing the incoming byte buffer inside a standard-compliant IEEE 802.11g data frame, directed at broadcast. Directing the frame to non-existing, or existing stations would produce either ACKs frames and/or retransmissions after the WuR frame. Choosing a broadcast frame limits further retransmissions of the frame undertaken by the IEEE 802.11 transmitter, as those are best-effort and no ACK frame is expected. Moreover, those are protected by the NAV mechanism [112]. First, the PHY WuR-CTC preamble, as well as its corresponding frame delimiter, are prepended to the aforementioned buffer. Afterward, the aforementioned byte buffer is translated into a sequence of bytes that produces a Peak-Flat modulated waveform when sent by the IEEE 802.11g PHY, according to the procedure explained in Section 3.4. Finally, the resulting bytes are sent through the IEEE 802.11 interface. Further detail about this implementation can be found in Annex B.2.

IEEE 802.15.4-compatible WuTx

The WuTx used by IEEE 802.15.4-compatible devices is based on the reconfigurable radio incorporated in the EFR-32 Mighty Gecko (EFR-32MG12) [35], which supports sending OOK symbols and obtain a data rate of 250 kbps. It does so through the Flex-Radio [126] library, which allows defining proprietary radio profiles that can be switched on and off on-demand.

The implementation of the WuTx also comprises a single source file, which implements the interface between the Flex-Radio API and the corresponding WuR-CTC functionality. The implementation is complemented by the rail_config.h header file, containing the configuration according to the Flex-Radio APIs. The frame format defined in that header uses OOK at 250 kbps and includes the WuR-CTC PHY headers and the CRC-8 LLC footer. The ook_wur_init_context uses the information in rail_config.h to add the WuTx profile to those supported by the EFR32MG12 reconfigurable radio. This configuration also includes the CSMA/CA parameters defined in Table 5.4.

The ook_wur_transmit_frame schedules the transmission of a WuR-CTC frame, and waits for the operation to finish. As mentioned previously, the WuR-CTC WuTx shares the same physical transceiver with IEEE 802.15.4, being the default state of the hardware. Therefore, after a WuR-CTC frame is scheduled, the transceiver is configured from receiving IEEE 802.15.4 to transmitting WuR-CTC. This operation

takes around 100 μs, afterward, the WuR-CTC frame is sent. Finally, the transceiver is configured back into its default state, IEEE 802.15.4 receiver.

This allows WuR-CTC IEEE 802.15.4 devices to perform as a standard-compliant IEEE 802.15.4 radio. In this testbed, the EFR-32MG12 is running a full IEEE 802.15.4 based protocol stack (the Thread stack [129]) concurrently with the WuR-CTC solution proposed in this article.

5.5.2 WuRx implementation for the WuR-CTC testbed

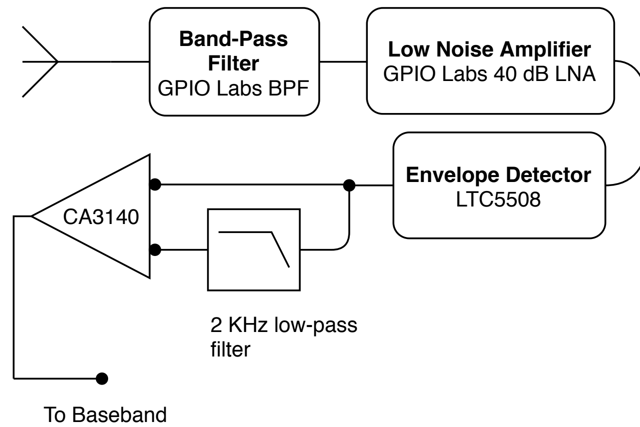
The WuRx is based on the implementation presented in Chapter 4. It is divided into two parts: the RF Front-End that amplifies, demodulates, and normalizes the incoming signal; and the baseband, which processes incoming signals into a binary stream and parses them according to the protocol. The block structure of the WuRx is presented in Fig.5.9

RF front-end

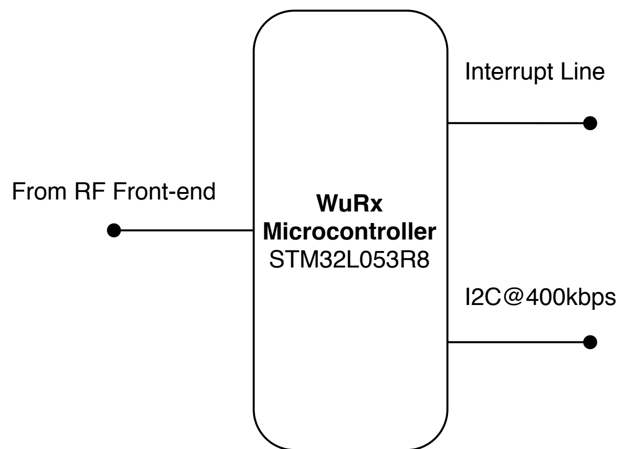
In the same way as in Chapter 4, the RF front-end developed for this testbed is composed of non-low-power, off-the-shelf RF components. Its block structure is shown in Fig. 5.9a, and it is analogous to the structure presented in Chapter 4.

Indeed, using a monolithic WuRx implementation including the front-end and the baseband using an integrated circuit would be ideal for the reduction in power consumption [105, 130]. Nevertheless, the off-the-shelf approach used produces a testbed for the evaluation of WuR-CTC that is both low-cost and flexible. The main characteristics of the current non-low-power RF front-end are summarized in Table 5.5. All components, except the LNA, which is USB powered, are powered from the STM32L0R8 3.3V voltage regulator output.

The RF front-end uses one band-pass filter that covers the complete 2.4 GHz ISM band instead of a single IEEE 802.11 channel filter covering 20 MHz. As a result, the RF front-end output is derived from the overlap of all channels operating in the 2.4 GHz band. This design increments the interference received by the WuRx versus a single-channel filter. Nonetheless, the WuRx design remains fit to demonstrate the viability of WuR-CTC. Not selecting a single-channel filter increments the robustness of the results presented, as these include a pessimistic scenario concerning received interference. Moreover, using this type of filter makes the testbed design more accessible to other researchers since, at the moment of writing of this document, single-channel IEEE 802.11 filters are not available off-the-shelf on major component providers. The consequences of the increased interference, both for frame error rate and false wake-ups are studied and discussed in depth in Section 3.6.



(a)



(b)

Figure 5.9: Block diagrams of the two WuRx components. (a) Structure of the RF front-end. (b) Structure of the baseband.

The filter is followed by an LNA and an envelope detector to demodulate the incoming signal down to baseband. The envelope detector has a low-pass filter behavior that can be tuned according to the load at its output. The input signals considered are both the 20 MHz bandwidth pseudo-OOK used by the IEEE 802.11 WuTx, and the 1 MHz bandwidth OOK used by the IEEE 802.15.4 compatible WuTx. Therefore, the envelope detector bandwidth cannot be matched specifically to any of the two signals without compromising the reception of the other. Instead, the detector output bandwidth was tuned to 2 MHz, a compromise in reception performance for the two input signal types. Although 2 MHz is lower than the 10 MHz baseband bandwidth of the IEEE 802.11 WuTx signal, in agreement with the WuRx simulation results in Chapter 3, the receiver exhibited better performance with lower bandwidth. Moreover, a 2 MHz receiver bandwidth also favors the reception of OOK signals sent by the IEEE 802.15.4 compatible WuTx by matching more closely their 1 MHz bandwidth. Nevertheless, such a design is sub-optimal to a matched filter

implementation, such as those that can be implemented in IEEE 802.11ba receivers, which only need to be matched to the 4 MHz bandwidth of the MC-OOK symbols.

To translate the OOK pulses into a binary waveform, the front-end uses a comparator, which matches the instantaneous signal level with a reference. In the prototype, this role is fulfilled by a CA3140 operational amplifier. The reference signal for the comparator. In contrast to the implementation of Chapter 4, the reference is obtained via a passive RC first-order low-pass filter with a cutoff frequency of 2 kHz.

Table 5.5: Components of the WuRx RF front-end.

Component	Parameters	Values
LNA	Gain[1]	31 dB
	Noise Factor	2.8 dB
	Supply Voltage	5 V
	Supply Current	200 mA
BPF	Center Frequency	2.45 GHz
	Bandwidth[1]	150 MHz
	Insertion Loss	2 dB
LTC5508	Active Current	550 μA
	Standby Current	2 μA
CA3140	Supply Current[1]	2 mA

[1] Obtained through component characterization.

Baseband

The baseband implementation is shown in Fig. 5.9b. It uses an STM-32L053R8 as the WuRx microcontroller instead of the PIC-8 used in Chapter 4. With an instruction clock of 16 MHz, this powerful 32-bit microcontroller is capable of processing the more complex WuR-CTC protocol in real–time. Nonetheless, its standby power consumption is higher (410 nA with the required peripherals active, compared to the 10 nA of the PIC-8 implementation). However, it is still a sub-μA current consumption, fit for low-power operation with a suitable low-power RF front-end. In addition, the active current consumption of the STM-32L053R8 is lower than with the PIC-8 (2.82 mA on the STM-32L053R8 vs 3 mA on the PIC-8).

Table 5.6: Components of the WuRx Baseband.

Component	Parameters	Values
STM32L053R8	Instruction Clock	16 MHz
	Active Current	2.82 mA
	Standby Current	410 nA

To allow for WuR-CTC, the WuRx needs to send the received WuR-CTC payload to the host device, in addition to taking it up when a WuS is received. Therefore, in addition to an interruption line to signal wake-ups, a bus is required to send data from received WuR-CTC frames to the host. Consequently, communications between the host device and the WuRx microcontroller occur through an I2C bus at 400 kbps.

WuRx software implementation

Analogously to the WuRx presented before, in Chapter 4, the WuRx microcontroller firmware handles the reception of incoming frames. Although the new WuRx microcontroller is relatively powerful, the synchronization procedure is not based on an optimal correlation mechanism [131], such a procedure still cannot be implemented using the current hardware. As in the previous work, WuRx synchronization is achieved by edge detection and a frame delimiter. Nonetheless, the extra computing power of the current microcontroller platform allows for a more precise implementation. The wake-up procedure is shown in Fig.5.10, explained below. First, the microcontroller is woken up from standby mode (0) with a rising edge from the comparator output, i.e., at the start of an incoming frame on the band of interest. After waking up, the WuRx microcontroller waits for a transition to a low level on the comparator output. If the WuRx does not find any transition to 0 after 40 samples sampled at 1.25 Msps, it discards the event as a false wake-up (1). After detecting the transition, the WuRx starts sampling preamble symbols at 250 ksps. At this point, if the WuRx finds more than 3 consecutive "0" symbols it discards the frame as a false wake-up (2). Otherwise, it keeps sampling the preamble until it encounters the frame delimiter or the preamble finishes. After the delimiter, the WuRx starts to match the address (3). If it fails to do so, it enters the hold sensing state, returning to sleep after a minor delay to avoid multiple wake-ups per frame (6).

Once a valid address is detected, the rest of the WuR-CTC frame is received and its CRC-8 checksum is calculated (4). If it matches the checksum received from the

frame, the WuRx microcontroller saves the frame and generates a pulse on the interrupt line to wake up the device connected to the WuRx (5). Otherwise, the frame is silently discarded, and the microcontroller returns to hold sensing (6). Afterward, the WuRx microcontroller returns to sleep directly (0). After being woken up, the host device can retrieve the frame from the WuRx via an I2C command.

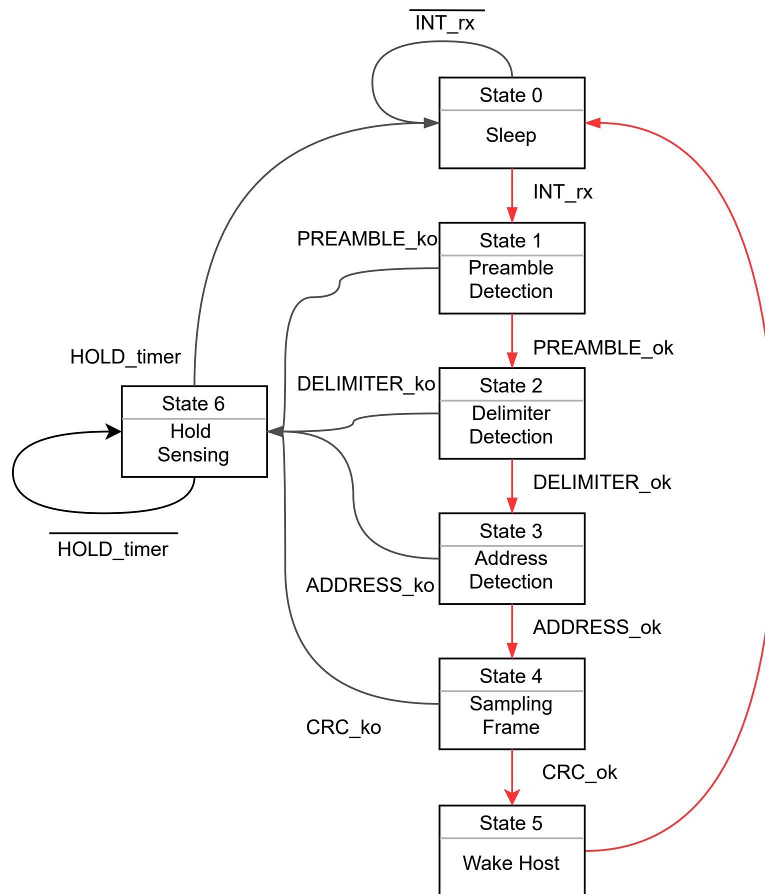


Figure 5.10: State machine representing the frame parsing procedure implemented in the WuRx microcontroller. The main reception path is highlighted in red.

To allow high transmission rates while keeping the power consumption of the baseband low, the WuRx microcontroller uses two clock sources. These are switched on and off mirroring the state of the host microcontroller:

1. An internal low-power RC oscillator with a clock accuracy of 0.25%. With this clock source, the WuRx can receive a frame of up to 20 bytes without losing synchronization. The internal RC is used for receiving **WuS** and **WuS+DATA frames** when in low-power mode.

2. An external oscillator circuit that incorporates a crystal resonator rated to 30 ppm accuracy. It has a power consumption of up to 100μW. However, with

this more precise clock source, the WuRx can receive **DATA frames** with the maximum length allowed by the WuR-CTC protocol. This oscillator is activated once the device is woken up to activity with a **WAKE frame**. It is deactivated after receiving a **SLEEP frame**.

The complete source code for the WuRx firmware can be found in Github [132] and further details about the implementation can be found in Appendix D.1.

5.5.3 Software implementation of the WuR-CTC solution

The only entity capable of interacting with the WuTx is the host device connected to the WuRx, as a consequence, most of the WuR-CTC LLC protocol implementation (as seen in Section 5.4.1) is found there. Said protocol implementation is programmed in C and strives to be portable, with a low resource footprint. Currently, it supports both the ESP-32 and EFR-32MG12 platform libraries. The implementation can be configured to be OS aware and run efficiently in a separate task or thread. Moreover, it can be executed in an event loop in platforms where no OS is present. To aid in the reproduction of the results and extension of the WuR-CTC solution presented the sources for this implementation, which include the WuTx are published in Github [133]. The implementation of the library is further discussed in Appendix D.1.

To evaluate the WuR-CTC testbed a demonstration scenario was prepared, including a data transmission from an IEEE 802.11 device to an IEEE 802.15.4 device, both equipped with WuRx. The code used for the demonstration application for both the ESP-32 and EFR-32MG12 can also be found in Github [134][135]. This code is further documented in Appendix D.2.

5.6 Operation of the WuR-CTC testbed

Devices for the testbed implement a demo application to evaluate the achievable throughput of WuR-CTC. The testbed was used to obtain tangible results that allow evaluating the feasibility of WuR-CTC. Some of the results were obtained directly from the testbed operation while others were obtained through the emulation of network traffic using parameters derived from the testbed operation.

The results obtained directly from the testbed operation include throughput, frame error rate, latency, and WuS loss. These are used to compare the WuR-CTC testbed to other state-of-the-art CTC systems found in the literature, as well as to validate the performance of the WuR system proposed in this chapter.

The results obtained from emulation allow assessing the viability of the low-power

of our off-the-shelf WuRx solution under realistic conditions. Although the RF front-end used in this testbed is not low-power, the microcontroller-based baseband is designed to operate in low-power applications and needs to be assessed in such scenarios.

5.6.1 Direct results of testbed operation

Scenario

The testbed includes a WuRx equipped IEEE 802.11 device, the ESP-32, and a WuRx equipped IEEE 802.15.4 device, the EFR-32MG12. Since the objective of the tests is not to profile the RF front-end of the WuRx, the layout of the devices is set up to not introduce relevant propagation losses. The two WuR enabled devices are separated by 1 meter and placed in an indoor residential environment. A diagram with the testbed layout is shown in Fig.5.11. This layout is also used in other CTC developments [122], thus, providing a valid comparator for the results.

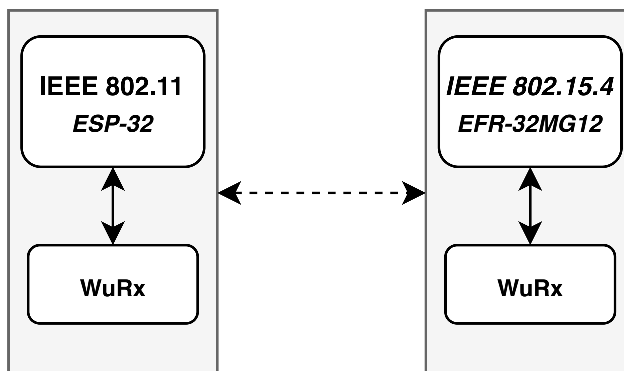


Figure 5.11: Diagram of testbed experimental setup.

To measure accurately the throughput and frame error rate, each device performs a transmission of 6.4 kbytes spread over 100 frames, each containing 64 bytes of randomly obtained payload bytes in its WuR-CTC MSDU. In addition to the 100 frames bearing data, and their corresponding **ACK frames**, the total transmission time accounts for the complete WuR interaction, including the **WAKE** and **SLEEP** frames required to wake up the receiving station. This procedure is repeated 25 times for each of the devices in the testbed, averaging the effect produced by bursts of interference. Additionally, throughput results include the effect of frame errors and retransmissions caused by received interference.

The transaction time is measured using a **WAKE+DATA frame** with a 1-byte payload. This is the shortest transaction bearing data that is viable to perform on a sleepy node. The transaction time takes into account the time required to complete the whole transaction, including the generation and transmission of the

WuS+DATA frame and the reception of the **ACK frame** generated by the receiver device. The test is performed 10 times to reduce the variability in delays. These can be caused by jitter in the host device event processing time and OS introduced delays.

Additionally, the probability of a missed wake-up transaction was measured to characterize the performance of the WuR mechanism implemented for the WuR-CTC testbed. This probability was evaluated using 500 wake-up transactions initiated by each of the WuTx. A transaction was counted as valid only if both, the **WAKE frame** and its respective **ACK frame**, were correctly received. Each wake-up transaction was spaced at least 100 ms to minimize bias introduced by bursts of interference. All **WAKE frames** were received with the WuRx in the low-power state.

Throughput and Frame Error Rate Results

The throughput results of the test, classified by WuTx device technology, are shown in Fig.5.12. Fig.5.13 shows the frame error rate from test sessions. These figures show the data with a box and whisker plot visualization with median and interquartile ranges, with the results from each of the test realizations overlayed.

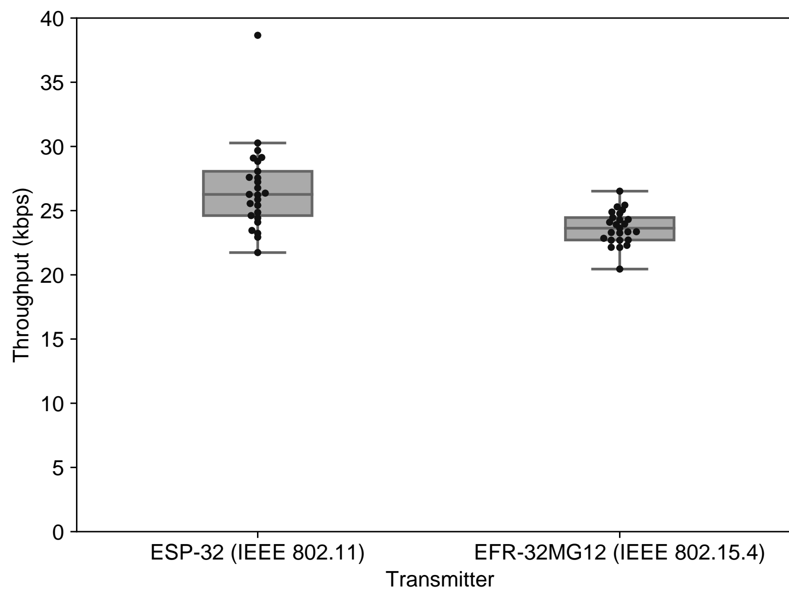


Figure 5.12: Throughput obtained from testbed operation. Results are shown for each of the WuTx.

The mean throughput is 26.737 kbps for the ESP-32 (IEEE 802.11) and 23.647 kbps for EFR-32MG12 (IEEE 802.15.4). These differ by roughly 3 kbps. This reduced

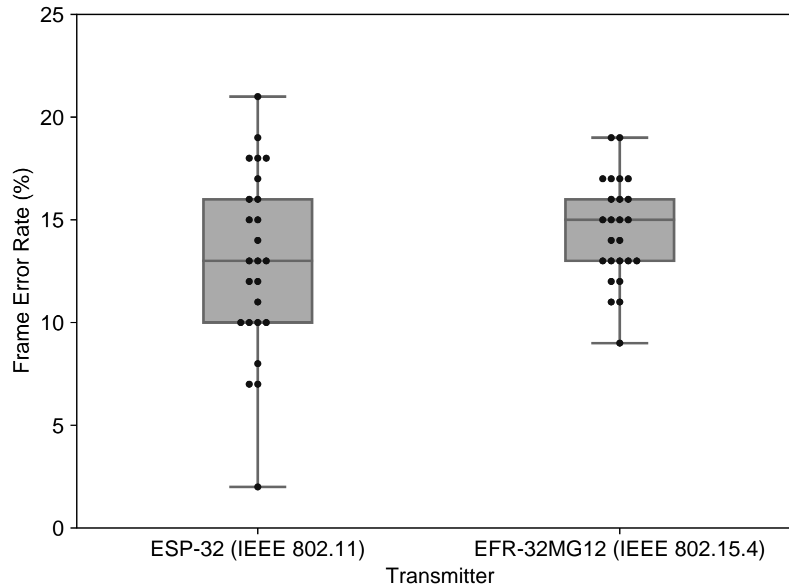


Figure 5.13: Frame error rate obtained from testbed operation. Results are shown for each of the WuTx.

difference is caused by device-specific frame processing times, including differences in the speed and latency of I2C communications with the WuRx microcontroller.

The WuR-CTC devices achieved a multiple of 7 throughput improvement over the CSMA-based CTC state of the art (3.1 kbps in [116]). The mean frame error rate observed was 13% and 14.48%, for ESP-32 and EFR-32MG12, respectively. A relatively high frame error rate is expected due to the nature of the experimental setup. The WuRx receives interference over the span of the 2.4 GHz ISM band while the WuTx only uses CSMA/CA on the channel where the main radio operates. Therefore, the testbed is vulnerable to collisions that will affect the WuRx but that neither the WuTx nor interfering stations can detect. In addition, the current WuRx implementation is more prone to synchronization errors than a WuRx based on correlation. However, the frame error rate measured is a three-fold improvement over non-channel-coded, emulation-based CTC between IEEE 802.15.4 and IEEE 802.11 [117] (40% in [117]). As a result, the throughput obtained with the WuR-CTC testbed, which includes the loss of throughput due to retransmissions caused by frame errors, more than doubles the maximum throughput found on the literature for emulation-based CTC with reliability (10.67 kbps in [122], also evaluated at 1 meter distance).

Latency Results

The transaction time results obtained are shown in Fig.5.14 with the same box plot format as the two preceding figures. When the ESP-32 acts as WuTx the transaction is completed in a mean of 10.63 ms. When the EFR-32MG12 acts as the transmitter, the transaction takes a mean of 12.83 ms.

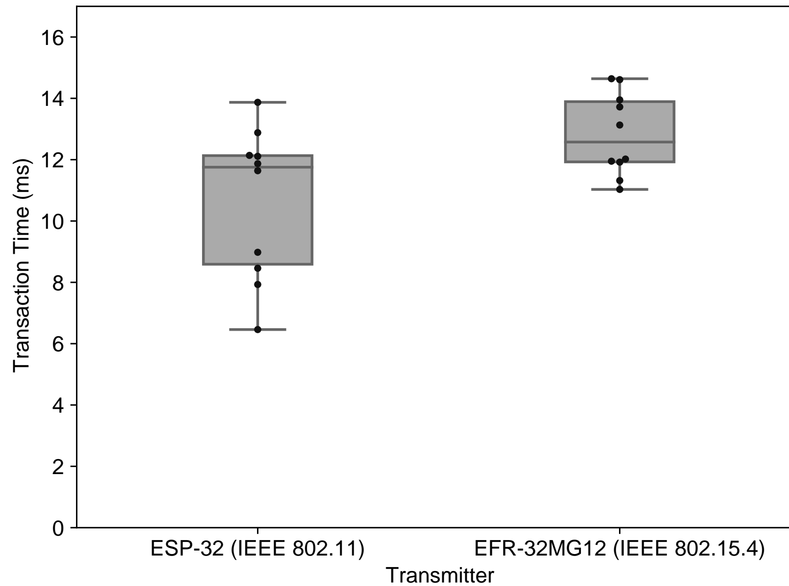


Figure 5.14: Transmission time obtained from testbed operation. Results are shown for each of the WuTx

Missed Wake-up Signals

With the ESP-32 WuTx, the missed wake-up transaction probability is 3.8%, while with the EFR-32MG12 WuTx it is 3.2%. Both rates are lower than the 13,74% frame error rate calculated for the 64 byte data frames evaluated for throughput. Nonetheless, the effect of a missed wake-up transaction is subsequently mitigated by retransmissions, which will occur after the receiving station fails to acknowledge the **WAKE frame** before a 20 millisecond timeout.

5.6.2 Emulation results

Although the prototype WuRx presented in this testbed has a non-low-power RF front-end, its baseband is designed to operate under restrictive power constraints. The WuRx microcontroller, which implements the WuR-CTC baseband, awakens to decode incoming frames and returns to sleep after processing them.

Unfortunately, the WuRx microcontroller needs to wake up to discriminate WuR-CTC frames from non-WuR-CTC frames. Therefore, non-WuR-CTC network activity increases WuRx power consumption. For this purpose, the length of the WuRx microcontroller wake-ups caused by non-WuR-CTC network frames was measured. Results of the measurements are shown in (5.5) show the dependence between the length of the non-WuR-CTC frame received and the WuRx wake-up length, according to the preamble sampling mechanism explained in Section 5.5.2.

This mechanism separates (5.5) into three regions, which can be mapped to the states shown in Fig.5.10. On the first, the non-WuR-CTC frame ends before the WuRx wakes up and is discarded after sampling three "0" symbols (1). On the second, the WuRx finds a transition to "0" in the expected range, but, in the same way as before, discards the frame after reading three "0" symbols (1). On the last, a transition to "0" and the comparator output remains high, causing the WuRx to discard the frame (2). Fig.5.15, shows graphically how the WuRx reacts to non-WuR frames according to (5.5).

$$f(Frm_{len}) = \begin{cases} 64\mu s & : Frm_{len} < 42\mu s \\ Frm_{len} + 22\mu s & : 42 \leq Frm_{len} < 74\mu s \\ 78\mu s & : Frm_{len} \geq 74\mu s \end{cases} \tag{5.5}$$

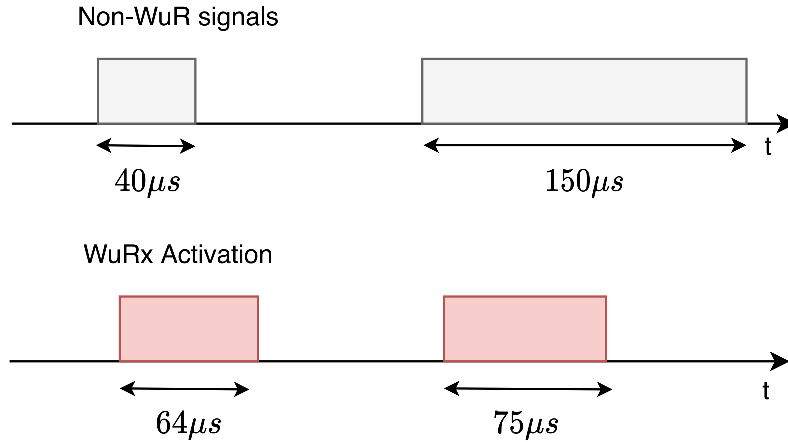


Figure 5.15: Example of non-WuR signals and the corresponding WuRx activation length according to (5.5).

Emulation framework

To help quantify the activity of the WuRx microcontroller, the WuRx activity rate is defined as the time fraction that the WuRx microcontroller spends on a high

power state. The lower the WuRx activity rate is, the longer the battery life of a WuR-CTC device will be. This figure resembles the duty cycle used in traditional synchronous power-saving mechanisms. To assess the viability of the baseband, the WuRx activity rate is measured under two different emulated scenarios comprising non-WuR interference.

The first scenario models a residential scenario. A location with low IEEE 802.11 network activity coming from a limited number of devices. The second scenario models a crowded location with a public WLAN and, consequently, features high network activity coming from a considerable number of devices. As presented in Section 5.5.2, the WuRx uses a band-pass filter covering the entire 2.4 GHz ISM band. The filter output is then fed to an envelope detector. As a consequence, the signal that reaches the WuRx microcontroller contains a binary version of the IEEE 802.11 network activity aggregated from the 2.4 GHz band. This signal reaches a high level when a frame is being transmitted on one or more channels and a low level when no transmission is occurring in any of them. To the author's best knowledge, there is no public dataset describing such a signal generated by IEEE 802.11 network activity at the time of writing of this document.

As a solution, one or more single-channel captures of IEEE 802.11 frames are combined to generate an overlapped occupation capture, which emulates the signal received by the WuRx microcontroller. This capture is derived from overlapping IEEE 802.11 frame sequences at the same time axis. These sequences are obtained from one or more IEEE 802.11 network captures. The overlapping process is the following: First, an occupation capture is derived from each source capture. The capture is generated by registering the intervals when a frame transmission occupies the channel. Second, the occupation captures are overlapped in an iterative process, as shown in Fig.5.16 to produce the final overlapped occupation capture; Last of all, the length of the intervals of the overlapped occupation capture is used to calculate the activity rate of the WuRx, according to (5.5).

The use of IEEE 802.11 frames provides a more pessimistic scenario for the emulation of the WuRx activity rate than using other standards operating on the 2.4 GHz ISM band. IEEE 802.11 uses shorter frames than other prominent technologies operating in the same band (IEEE 802.15.4 [23] or Bluetooth [13]). This leads to a higher proportion of WuRx wake-up time for each frame and an increased WuRx activity rate. Moreover, the method used to overlap the occupation captures overestimates the time that the channel will be seen as occupied by the WuRx since it assumes that all frame interference events generate constructive interference that can be detected by the WuRx and trigger a wake-up event.

For the low traffic residential scenario, the overlapped occupation capture is generated by using as source data 11 captures registered ad-hoc. These captures were taken with a commodity IEEE 802.11bgn WLAN card on a Linux PC in a single residential location from all IEEE 802.11 channels where activity was detected in

the study period. The use of captures from different channels reflects the effect of different channel traffic profiles on the aggregated network activity.

A capture obtained from the VWave dataset [136] scenario “Pioneer” was used to construct the overlapped occupation capture for the high network activity scenario. It features traffic obtained from a public WLAN network with a high number of devices. However, it does only cover a single channel. Therefore, in this case, the original “Pioneer” capture is separated into 11 equal-length consecutive segments, which are used to generate the overlapped occupation capture. An iteration of this procedure is shown in Fig.5.16

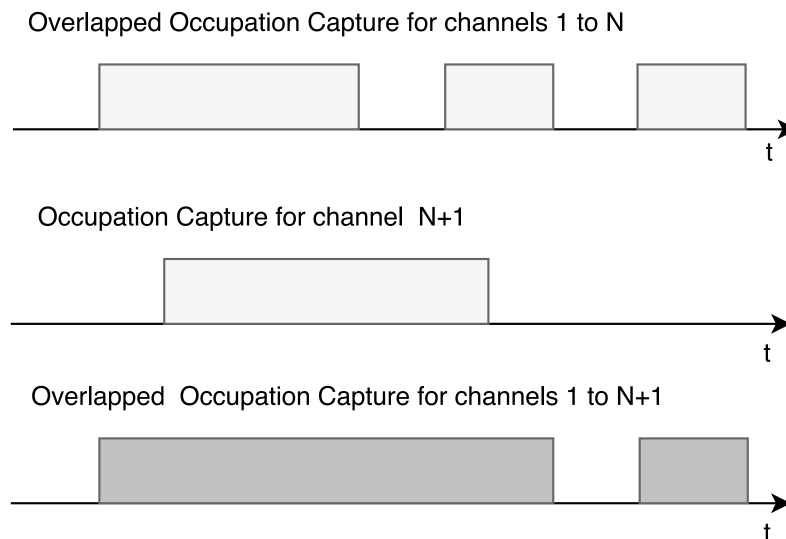


Figure 5.16: An iteration of the channel overlapping process.

Finally, to relate the network activity with the WuRx activity, the overall occupancy rate of the channel was defined as the rate of time that the channel remains occupied with non-WuR-CTC frames.

Emulation Results

As can be observed in Fig.5.17, the final activity rate for the WuRx is lower than the occupancy rate of the channel in all scenarios. In the high traffic scenario, the WuRx activity rate is 2.31%, while the channel occupancy rate is 19.33%. In the low traffic scenario, the activity rate is only 0.24%, while the channel occupancy rate is 1.17%. The ratio of WuRx activity compared to the overall channel occupancy rate is 0.21 in the low traffic scenario and 0.12 in the high traffic scenario. This reflects the influence of the distribution of frame lengths received.

With these activity rates, the WuRx baseband consumes an average of 65.54 μA and 7.18 μA in the high and low traffic scenarios, respectively. These results support

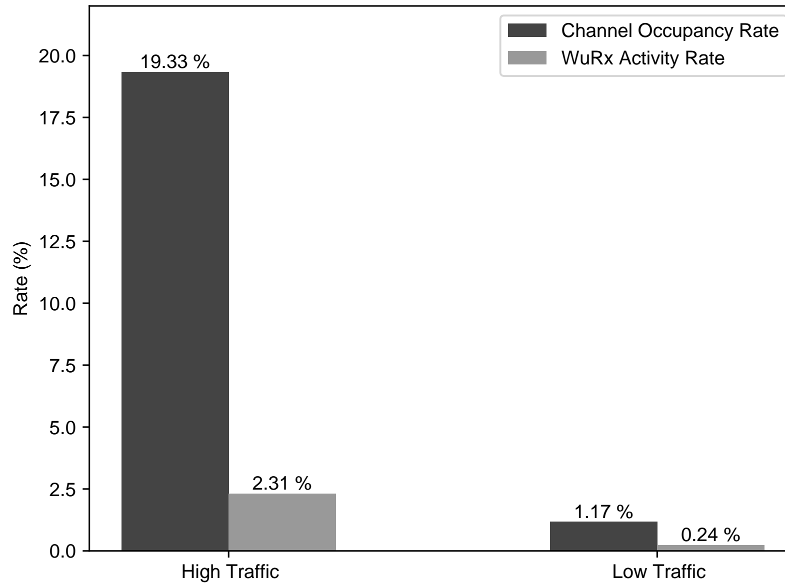


Figure 5.17: Channel occupancy rate and WuRx activity rate for both scenarios.

continuing the development of an off-the-shelf RF front-end to use with the current WuRx baseband. This combination would provide an accessible solution to evaluate WuR-CTC in its main application field: energy-restricted wireless networks.

Scenarios with even higher sustained traffic and, therefore, higher WuRx activity rate than the high traffic scenario emulated here are plausible. However, these scenarios take place in locations with a high device density. To support the high density of devices, such locations usually need to have a reliable power source available. As a result, these scenarios fall outside of the application scope of WuR-CTC as a technology to enable low-power device operation.

5.7 Conclusions

This chapter introduces WuR-CTC, a solution that helps to reduce IoT fragmentation by providing direct, reliable, and bidirectional communication between non-compatible devices. With WuR-CTC, it is demonstrated that WuR is not only a power-saving mechanism but a contribution to the CTC landscape. Thus, WuR-CTC defines a high-throughput CTC solution that can be added to existing WPAN and WLAN devices.

Instead of using a simulated platform, the viability of the WuR-CTC approach is validated with a physical testbed. The development of that testbed reused components previously introduced in chapters 3 and 4, involving both hardware and

software implementation. The resulting testbed implementation introduces a WuR-CTC solution covering the PHY and MAC layers. The devices on the testbed were capable of reliable and fully bidirectional CTC. Moreover, the presented implementation provides a MTU high enough to support a 6Lo stack, which would enable it to provide IP-based CTC. The testbed has shown improvements in throughput over the current CTC state of the art. This demonstrates the advantages of the WuR-CTC approach over other CTC solutions.

The testbed throughput, which includes the loss of throughput due to retransmissions caused by frame errors, more than doubles the maximum throughput found on the literature for emulation-based CTC with reliability (10.67 kbps in [122]). Moreover, it provides a 7-fold throughput improvement over the CSMA-based CTC state of the art (3.1 kbps in [116]).

Finally, it has been shown that a microcontroller-based WuRx base-band, as implemented using the off-the-shelf approach firstly presented in Section 4, and further developed in the testbed, can operate in power restricted environments, even those with high network activity.

Finally, and although the WuR-CTC implementation presented here might not be optimal in terms of power consumption, it serves in demonstrating the viability of the WuR-CTC concept. The testbed is implemented using easily available off-the-shelf parts, thus, enabling the reproduction and extension of the results presented therein. Additionally, the firmware of all devices used has been made open-source. The results obtained with the WuR-CTC testbed highlight what can be achieved by harmonizing the nascent WuR standards: enabling bidirectional communications between heterogeneous devices in a transparent, and low-power manner.

Finally, Chapter 5 was published in *IEEE Access* (Q1, Computer Science, Information Systems 35/156 [102]) on the 1st of January 2021.

M.Cervià, A.Calveras and J.Paradells. Wake-up Radio: an Enabler of Wireless Convergence. *IEEE Access [137]*

Additionally, the firmware developed for this chapter has been open-sourced and can be found in Github [132–135].

In future work, WuR-CTC could be implemented using more performant WuR hardware solutions, such as those being developed for IEEE 802.11ba. Moreover, a 6Lo layer can be developed on top of the proposed solution.

Chapter 6

Conclusion and future work

After the inception of WSN, several incompatible wireless specifications targeted at low-power applications appeared. Additionally, those solutions have rapid obsolescence cycles. Both of these issues difficult to maintain or upgrade existing WSN. Thus, fragmentation in the WSN space is one of the main factors hampering the adoption of WSN solutions. However, fragmentation is a multi-faceted problem. On one hand, there is an incompatibility between specialized WSN devices and general-use devices, which are used by end-user and that provide access points to the Internet. On the other, there is another incompatibility between the different solutions competing in the WSN space. To face both of these challenges, this contribution presents solutions that follow a common thread: WuR. WuR initially appeared as a power saving-mechanism. It was proposed to enable devices to save power when inactive and, at the same time, be responsive to incoming messages. This way, WuR offers improvements over classic power-saving schemes that present a trade-off between latency and efficiency.

First of all, this work has contributed to the enablement of general-use devices in low-power applications with WuR. The aim is to reduce the divide between WSN and those networks formed by general-use wireless devices. For this purpose, a WuR system for IEEE 802.11 devices was designed and developed. The system, in an improvement over IEEE 802.11ba, proposes a WuTx implementation that remains compatible with legacy devices. The legacy-compatible WuTx uses an amplitude-based modulation called Peak-Flat, which can be received by OOK receivers. The WuR system herein presented achieves a 250 kbps data rate and its WuTx implementation is purely software-based. Therefore, this proposal is easily deployable in legacy devices with a software update. To be completely WuR-enabled, those only require the addition of a WuRx peripheral. The viability of the legacy-based Peak-Flat modulation has been demonstrated using detailed simulations with two different WuRx architectures. In those simulations, the performance of Peak-Flat has been found competitive against the state of the art using realistic channel mod-

els. Moreover, the WuTx has been implemented in commodity IEEE 802.11 devices. First, on Linux-based computers and, afterward, on an embedded wireless device, the ESP-32. A journal article describing this work was accepted for publication in IEEE Access [125]. Additionally, the software-based WuS generation method derived from this work is in the process of being patented [104]. Finally, further outreach about this work has been made in IEEE 802.11ba work group meetings.

Next, to complement the previously presented WuTx, a matching WuRx has been designed and implemented. This device is completely based on easily available off-the-shelf parts. The development of the WuRx has been separated into two different projects. The first, covering the RF front-end, and the second, the baseband. This work features the development of the baseband. This component is based on an ultra-low-power microcontroller with an sleep mode consumption of 10 nA. Additionally, the baseband supports 16-bit addressing, with a data rate of 250 kbps. It is demonstrated that the development of a high rate and low-power WuRx baseband is feasible using off-the-shelf parts. Moreover, the WuRx design allowed to physically validate the use of the previously introduced backward-compatible WuTx. Finally, the off-the-shelf implementation broadens the reproducibility and applicability of the previous findings. The work related to this baseband was published in IEEE Wireless Communications Letters[73], and is being used in upcoming developments by the research group.

The previously developed WuRx and WuTx were applied to the last challenge remaining, connecting non-compatible wireless devices. High-rate WuR enables sending arbitrary data, instead of only device addresses. This way, the WuR-CTC concept was introduced to obtain CTC that is high-rate and bidirectional. Thus, WuR is applied not only as a power-saving mechanism but as an opportunity to connect previously non-compatible devices. Using the WuTx and WuRx designs based on those presented in the previous contributions, a physical testbed was implemented to showcase and validate direct general-purpose communications with WuR between IEEE 802.11 and IEEE 802.15.4 devices. The testbed demonstrated reliable bidirectional communications, with a throughput higher than previous CTC solutions. This result showcases the opportunity for harmonization that nascent WuR specifications bring to the WSN space. This research was accepted for publication in IEEE Access [137]. Additionally, the source code for all firmware implementation has been made available publicly [132–135].

The results produced by both of these research topics allow fulfilling both of the research objectives involving WuR. Nonetheless, other research topics, derived from those, can be undertaken as future work.

The first is an optimization in the WuRx design presented in Chapters 4 and 5, concerning the implementation and refinement of a low-power WuRx RF front-end. This development should complement the already low-power baseband developments presented in Chapters 4 and 5. For this purpose, a semi-passive WuRx can be devel-

oped with off-the-shelf parts that, while sub-optimal to monolithic implementations, can add to the low-power microcontroller baseband, while following the off-the-shelf design philosophy to broaden the accessibility of WuR research.

The second would be the implementation of an IP-based communications stack over the WuR-CTC solution presented in Chapter 5. The WuR-CTC solution link layer provides an MTU of 89 bytes, enabling the use of relatively unconstrained IP-based communications over a 6-Lo adaptation layer. This way, the utility of WuR-CTC can be maximized as it can be used to connect WSN devices directly to the internet. This can allow robust WSN that connect to the internet in a decentralized manner, without using gateway devices, as is common in current WSN deployments.

Finally, there is current interest in WuR in the cellular ecosystem. The 3GPP accepted the use of a duty-cycled WuS to signal stations incoming messages in Release 16 [138][139]. The methodology presented previously for WLAN could be applied here in order to add OOK-based WuR functionality in a way that is legacy-compatible with already released Long Term Evolution (LTE) stations.

Appendices

Appendix A

WuRx evaluation with MATLAB

Three key elements were implemented in MATLAB and Simulink to perform Peak-Flat WuR evaluation in a simulated environment. First, the WuTx, described next in Section A.1, second, the channel emulation, shown in Section A.2, and, last, the WuRx, presented next in Section A.3.

A.1 Implementing the WuTx

To implement the WuTx, several elements of the MATLAB WLAN Toolkit were used in conjunction with the software-based method to generate a WuS.

First, the WLAN Toolkit waveform generator object is initialized. The object is parametrized to use a 6 Mbps bitrate (MCS0 in the WLAN Toolkit library) with the non-HT header format, which is used by IEEE 802.11g. Additionally, its payload size is selected to fit the desired number of WuS bits.

```
1 % 6 mbps and adapt the payload length of the WuR PSDU accordingly
2 nonHT = wlanNonHTConfig;
3 nonHT.MCS = 0;
4 nonHT.PSDULength = 1 + (24*length(psdu)/8);
```

Code Listing A.1: Initialization of the wavefom generator.

Afterward, the bitstream that generates the WuS is created following the principles described in Section 3.4. For example, in the following snippet, the variable `one_sequence` is the 24-bit Flat Symbol generation sequence obtained in (3.17). The output of this procedure featured in the following snippet is `sequence`, which contains the WuS bitstream.

```
% initialize scrambler predicted state and data bit array
state = [1,0,1,1,1,0,1];
sequence = zeros(1, 24*length(psdu) + 8);

%prepare the packet with the desired symbols.
for i=1:1:(length(psdu)-1)
    %skip the rest of the first symbol, with the SERVICE field.
    index = ((i-1)*24) + 9;

    if psdu(i) == 1
        sequence(index:(index + 23)) = one_sequence;
    end
end

%this is the PSDU, so if the seed is 93, the state is not the same ...
    as the
%16 SERVICE bits have already been scrambled.
for i = 1:1:16
    res = mod(state(7) + state(4), 2);
    shifted = zeros(size(state));
    shifted(2:7) = state(1:6);
    shifted(1) = res;
    state = shifted;
end

%generate scrambler sequence for predistortion
for i = 1:1:(24*length(psdu))
    res = mod(state(7) + state(4), 2);
    shifted = zeros(size(state));
    shifted(2:7) = state(1:6);
    shifted(1) = res;
    state = shifted;
    sequence(i) = xor(res, sequence(i));
end
```

Code Listing A.2: Generation of the WuS bitstream.

Next, the waveform featuring the WuS is generated by feeding the bitstream contained in the `sequence` variable, along with the previously created waveform generator to the `wlanWaveformGenerator` function. This function outputs the complex waveform generated by the IEEE 802.11g PHY, according to the input bitstream, sampled at 20 Msps. Additionally, the scrambler seed previously used to generate the bitstream is passed.

Finally, the waveform is prepended with 100 '0' samples and a corresponding time axis is generated.

```
txWaveform = wlanWaveformGenerator(sequence, nonHT, ...
    'ScramblerInitialization', 93);

```

```
3 %prepend amplitude zero.
4 txWaveform = [zeros(100*20,1); txWaveform];
5 time = (0:length(txWaveform)-1)*(1/sample_rate);
```

Code Listing A.3: Generation of the WuS wavefom.

The variable `txWaveform` contains the WuS waveform, which is next passed to the channel model.

A.2 Implementing the transmission channel

Channel simulation was used to emulate the effect of the transmission medium on the WuS waveform. For this purpose, two channel models were used. First, an AWGN model was implemented, where only Gaussian noise was added to the WuS waveform. Second, a fading channel model, following IEEE 802.11 TGn Ch.B model was implemented.

First, the waveform output by the WuTx was resampled to 160 Msps. This is done to reduce the error on the subsequent filtering done by the WuRx implementation.

```
1 txWaveform = resample(txWaveform, oversampling_factor, 1);
```

Code Listing A.4: Resampling of the WuS waveform.

No additional signal processing is done in the AWGN implementation. However, after this step, the TGn. Ch.B model fading channel is incorporated before adding Gaussian noise.

```
1 tgnChan = wlanTGnChannel('SampleRate',160e6, ...
2 'CarrierFrequency', 2.45e9);
3 %apply TGnB channel
4 txWaveform = tgnChan(txWaveform);
5 txWaveform = awgn(txWaveform, snr_penalty, mean_pow);
```

Code Listing A.5: Addition of fading according to TG. Ch.B model.

To correctly calculate the resulting noise power, the additional bandwidth of the signal, versus the 20 MHz noise bandwidth must be compensated. Thus, a penalization factor is used to obtain the expected noise spectral density and maintain the spectral noise density that maintains the 20 MHz Bandwidth SNR previously defined. Following, the corrected SNR and the WuS waveform are input to the `awgn` function, which adds Gaussian noise with the expected spectral density.

```
1 snr_penalty =  snr - 10*log10(oversampling_factor)
```

```
2 txWaveform = awgn(txWaveform, snr_penalty, mean_pow);
```
Code Listing A.6: Addition of Gaussian noise.

Now, `txWaveform` is ready to be passed to the WuRx implementation.

A.3 Implementing the WuRx

The WuRx implementation follows the block diagram shown in Fig.3.16. It includes a band-pass filter, envelope detection, low-pass filtering, a bit decoder implementation, sampling, and, finally, symbol decision.

First, the `txWaveform`, which is described in baseband, is filtered by a Butterworth low-pass filter with 2.4 MHz bandwidth. This action emulates the band-pass filtering that would be applied to an actual band-pass WuR signal.

```
1 %filter at bw 2.4 MHz just as specified on ba guideline
2 order_LPF=2;
3 Fcut = 2.4e6; %3dB cut-off frequency at 2.5MHz
4 Fs = oversampled_rate; % the high sampling frequency is used such ...
    that the Matlab
5 function "freqz" is approximately correct within 6MHz.
6 [num_LPF,den_LPF] = butter(order_LPF, Fcut/(0.5*Fs), 'low');
7 %save as timeseries for simulink
8 txWaveform = filter(num_LPF,den_LPF,txWaveform);
```
Code Listing A.7: Bandpass filtering.

Afterward, the waveform is transformed back to baseband, and its envelope is taken out. This is modeled by the application of an absolute value function to the complex input waveform.

```
1 txWaveform = abs(txWaveform);
```
Code Listing A.8: Obtaining the signal envelope and .

Afterward, a time axis is generated, and a Timeseries object is generated from the incoming waveform and the related time axis.

```
1 timeWaveform = (0:length(txWaveform)-1)*(1/oversampled_rate);
2 wlanTimeSeries = timeseries(txWaveform, timeWaveform);
```
Code Listing A.9: Timeseries object creation.

The Timeseries object is then passed to a Simulink model implementing low-pass filtering and bit decoding. Simulink models are developed according to the architectures presented in Sections 3.5.2 and 3.5.3, which contain circuital elements. Simulating those elements with rigor would require a more complex implementation than what would be feasible to develop in the framework of this contribution. Fortunately, Simulink provides detailed models of circuital elements and tight integration with MATLAB, therefore, it was used to facilitate this implementation. The previously obtained Timeseries object, containing the WuS waveform is fed as an environment variable to the model using the `sim` API. The model, in turn, generates an output Timeseries object, that is received by the script also as an environment variable.

```
1 wlanTimeSeries = timeseries(txWaveform, timeWaveform);
2
3 %simulate the result
4 r = sim('lowpass.slx');
5 val_out_sliced_packet
6 res_time = val_out_sliced_packet.Time;
7 res_val = val_out_sliced_packet.Data;
```

Code Listing A.10: Calling the Simulink model.

The waveforms generated by a model of a OOK bit decoder implementation, as described in Section 3.5.2 can be observed in Fig.A.1.

Finally, the bits contained in the Simulink output waveform are sampled. The sampling time of the first bit is obtained using the delay from the start of the waveform, to the ideal sampling time of the first WuS bit. Subsequent sampling times are obtained by adding the bit period to the previous sampling time.

Subsequently, after a bit is sampled, it is compared with its expected value. This operation provides the number of bit errors, which at the end of the simulation is used to estimate the BER.

```
1 samp_conversion = oversampled_rate;
2 samp_offset = 3.8e-6 * samp_conversion;
3 init_sample = 124e-6 * samp_conversion;
4 init_sample = init_sample + samp_offset;
5
6 bits_ko = zeros(1,length(psdu));
7 bits_detected = zeros(1,length(psdu));
8
9 reference_value = mean(res_val)
10
11 %skip preabmble bits.
12 for k = 4:(length(psdu) - 1)
13     sample = uint32(init_sample + (k - 1)*4e-6* samp_conversion);
14     sample = res_val(sample);
```

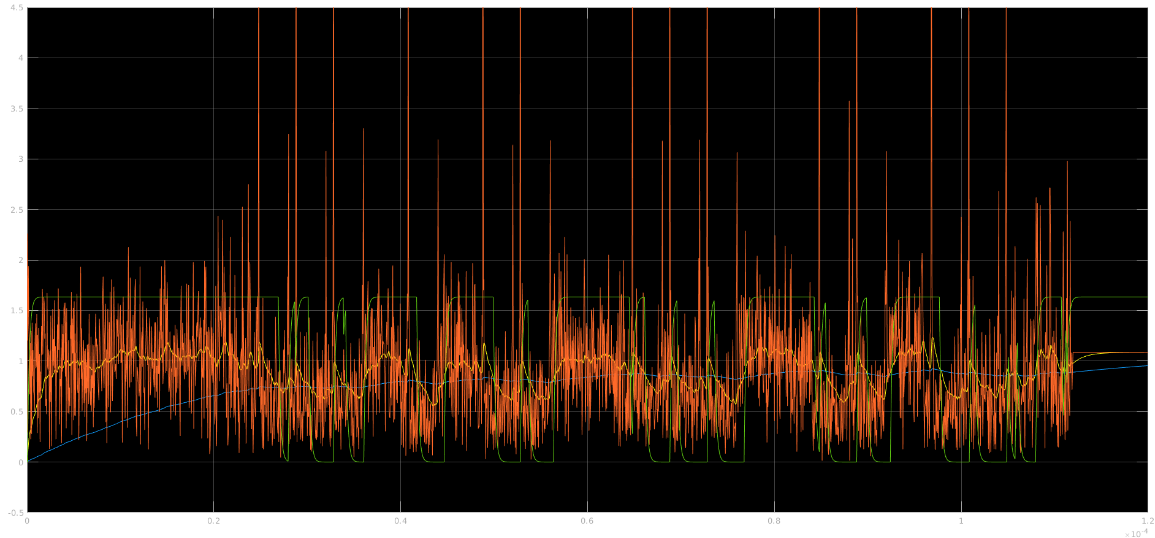


Figure A.1: Waveforms input and output by an OOK detector Simulink model. The orange waveform corresponds to the WuS waveform contaminated with noise by an AWGN channel model. The yellow waveform corresponds to the low-pass filtered input signal, fed to the positive terminal of the decoder comparator. The blue waveform corresponds to the detection threshold, fed to the negative terminal of the decoder comparator. Finally, the green waveform is the comparator output, which corresponds to the model output.

```
15
16      if(sample > 0.7)
17          sample = 1;
18      else
19          sample = 0;
20      end
21
22      if(sample ≠ psdu(k))
23          disp(strcat(["Error in bit " num2str(k) ". Sample is " ...
                num2str(sample) "
24  and bit is " psdu(k)]))
25          bits_ko(k) = 1;
26          bit_errors(k - 3) = bit_errors(k - 3) + 1;
27      end
28
29      bits_detected(k) = sample;
30  end
```

Code Listing A.11: Bit sampling and comparison.

If any of the bits in the frame is erroneous, the frame reception is considered failed towards the FER statistic.

Appendix B

Software-based legacy-compatible WuTx implementation

This appendix describes the implementation of the legacy-compatible WuTx, focusing on its software implementation. First, the implementation of a Python-based WuTx, compatible with Linux-based platforms, is presented in detail in Section B.1. Afterward, a C-based WuTx implementation, compatible with the embedded ESP-32 platform, is described in Section B.2.

B.1 Python WuTx implementation

The Python-based WuTx implementation is compatible with any Linux computer that supports Python 2, as long as it includes kernel support for the `nl80211` wireless driver interface. Currently, `nl80211` is the prevalent driver model, thus, the implementation is compatible with the vast majority of modern Linux distributions.

Access to the WuTx implementation is offered through a command-line interface. The interface accepts various configuration arguments, which support setting the desired WLAN output interface, the scrambler seed, the number of frames to transmit, the repetition interval, the target address, and the bitrate to be used by the PHY. The order of the command-line configuration arguments is the following:

```
test_packets.py -i <interface> -s <seed> -n <packageNumber (discard
or 0 for indefinite)> -f <sendInterval ( in ms)> -a <address> -m <useMcs
(from the 802.11g available MCS)>
```

Following, the implementation of the WuTx will be discussed in detail, using snippets from the source code. First, the implementation configures the radio interface with

the expected bit rate defined by the MCS argument and, subsequently opens a raw socket bound to the same interface.

```
1       if not pyw.iswireless(self.int):
2           raise InterfaceDoesNotExistError("Interface "
3           + self.int + " not found.")
4
5       #get info about this interface
6       card = pyw.getcard(self.int)
7       self.macAddr = pyw.macget(card)
8       link = pyw.link(card)
9       Printer.log("interface:"+ self.int)
10
11      if link != None:
12          for d in link:
13              Printer.log(str(d) + ":" + str(link[d]))
14      #try to set the correct tx MCS to this interface
15      # iw dev wlan0 set bitrates mcs-2.4 <mcs>
16      try:
17          subprocess.check_call(["iw", "dev", self.int, 'set', ...
                'bitrates', 'legacy-2.4',
18          str(BitrateConverter.get_rate_from_mcs(self.m))])
19
20      except subprocess.CalledProcessError:
21          Printer.log("Device does NOT accept speed:" +
22              str(BitrateConverter.get_rate_from_mcs(self.m))
23              +". Continuing at default rate.")
24
25
26      try:
27          self.injecter = socket.socket(socket.AF_PACKET,
28                      socket.SOCK_RAW, socket.htons(0x0003))
29          self.injecter.bind((self.int, 0x0003))
30      except Exception as e:
31          print e
32          raise InterfaceDoesNotAcceptPrepareTx("Could not bind the ...
                socket.")
```

Code Listing B.1: Implementation of the WuTx setup.

Afterward, the payload that generates the WuS specified through the command line is generated, ready to be sent over the raw socket.

The first step of this procedure consists of aligning the bytes that generate the WuS with the IEEE 802.11 framing, as explained in Section 3.4.1. However, due to platform limits, the WuS is introduced into the MSDU instead of the PSDU of the IEEE 802.11 frame. Hence, extra padding and additional configuration for the IEEE 802.11 MAC headers is added to the procedure.

```
1     #generate the bit array using the seed value after
2     #the signal field (the one extracted with gnu radio)
3     # without MAC header!
4     self.state = BitArray(7)
5     self.state.uint = self.seed
6     self.state.reverse()
7
8     # The rest of service filed, 9 bits, MAC header 24 bytes  and ...
         llc, which cannot
9     # be modified fully is 36 bytes long,
10    # Consequently, update the scrambler status this byte number so ...
         it does not
11    # desync. Also take into acconut the void signal byte.
12    #now generate a packet which needs a sufficient number of bits ...
         to send the
13    #address, after MAC symbol has been sent.
14
15    bits_headers = 36.0*8.0
16    symbols_headers = math.ceil(bits_headers/self.bitspersymbol)
17    bits_headers_padded = symbols_headers*self.bitspersymbol
18    bits_padding = int(bits_headers_padded - bits_headers)
19
20    #advance signal field symbol, headers plus signal symbol.
21    self.pos = int(symbols_headers) + 2
22
23    Printer.log("bits headers: " + str(bits_headers))
24    Printer.log( "header symbols : " + str(symbols_headers))
25    Printer.log( "bits_headers_padded : " + str(bits_headers_padded))
26    Printer.log( "bits_padding : " + str(bits_padding))
27
28    #two MAC and the last two bytes of LLC, 14 bytes --> 112 bits
29    settable_headers_bits = (14*8)
30    #skipping service!
31    self.bitArray = BitArray((self.length*self.bitspersymbol * 16) +
32    int(bits_headers_padded) + (self.length*self.bitspersymbol * ...
         36) - 16)
33    self.bitArrayIndex = 0
34    payload = BitArray((self.length*self.bitspersymbol * 16) + ...
         bits_padding +
35              settable_headers_bits + (self.length*self.bitspersymbol ...
                    * 36))
36    self.offset = 0
37    #advance the state of the scrambler an equal number of the ...
         symbols we're going to
38    # skip.
39    self._sendSymbol(False, None, int(bits_headers_padded - 16))
40
41    #set the settable MAC headers to the required values
42    #dst
43    payload.overwrite('0x010203040506', 0)
44    #src
45    payload.overwrite('0x010203040506', 48)
46    #next header
```

```
47      payload.overwrite('0xabcd', 96)
48
49      #leave the padding bytes as zeros
50
51      self.offset = bits_padding + settable_headers_bits
```

Code Listing B.2: Adding padding and alignment to the byte sequence generating the WuS.

The payload WuS uses basic framing. It starts with a 4-bit frame delimiter, followed by an alternating 16-bit sequence that can allow the receiver to synchronize itself with the signal. Afterward, the address of the target device is set in the following 16 bits.

```
1       #add a delimiter for the WuRx
2       for _ in range(self.length):
3           self._sendSymbol(False, payload)
4
5       for _ in range(self.length):
6           self._sendSymbol(False, payload)
7
8       for _ in range(self.length):
9           self._sendSymbol(False, payload)
10
11      for _ in range(self.length):
12          self._sendSymbol(True, payload)
13
14      #Send a sync preamble
15      for i in range(0,16):
16          self._sendSymbol(True, payload)
17          self._sendSymbol(False, payload)
18
19      #Generate the address bits
20      address = BitArray(16)
21      address.uint = self.address
22
23      for bit in address:
24          for _ in range(self.length):
25              self._sendSymbol(bit,payload)
26
27      print payload[(bits_padding + settable_headers_bits):]
28      print self.bitArray
29
30      #swap the bit endianness to match what is used on the wlan TX
31      self.message_bytes = bytearray(payload.tobytes())
32
33      #Append the payload of the WuS.
34      for i in range((bits_padding + settable_headers_bits)/8, ...
            len(self.message_bytes)):
35          self.message_bytes[i] = ...
```

```
BitReverseTable256[self.message_bytes[i]]
```

Code Listing B.3: Adding framing to the WuS.

This procedure uses the `_sendSymbol` callback, which is set according to the modulation used. It can send the two possible Peak-Flat Symbols. A Flat Symbol, coding a "1" bit, or a Peak Symbol, coding a "0" bit. It is implemented according to the sequence generating the Flat Symbol, for example, here can be seen the implementation for the 6 Mbps data rate, using the Flat sequence found in Section 3.3.2. For each of the payload bits coding a WuS bit, the implementation adds the expected scrambler value, and, if the WuS bit is to code a "1", XORs the corresponding Flat Symbol sequence bit.

```
1     def _sendSymbol_24(self, bit, payload=None, length=0):
2       good_bit_one = [1,1,0,0,1,1,1,0,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,0]
3       if length is 0:
4          length = self.bitspersymbol
5
6       for i in range(self.offset, self.offset + length):
7          res = self.state[6] ^ self.state[3]
8          #print "[" + str(i) +"]:"+ self.state.bin
9          self.bitArray.set(res, self.bitArrayIndex)
10         self.bitArrayIndex = self.bitArrayIndex + 1
11         if bit is True:
12            payload.set(good_bit_one[i - self.offset], i)
13         if payload is not None:
14            payload.set(payload[i] ^ res, i)
15         self.state.ror(1)
16         self.state[0] = res
```

Code Listing B.4: Add the payload to code for a WuS bit.

Finally, the obtained payload is sent through the raw socket a pre-specified number of times, with the configured interval between frames. Afterward, the raw socket is closed and the process exits successfully.

```
1     def send_wus(self):
2
3         if self.sent_packets >= self.num and self.num is not 0 :
4             raise SenderHasNoMorePacketsToSend("Successfully sent " +
5                 str(self.sent_packets) + " packets.")
6
7         time.sleep(self.period/1000.0)
8
9         self.injecter.send(self.message_bytes)
10
11        self.sent_packets = self.sent_packets + 1
12        Printer.log("Sent packet num " + str(self.sent_packets))
13
14    def send_frames(packet_sender):
```

```
15        try:
16            last_millis = millis()
17            while(True):
18                Printer.log("[" + str(millis()) + "]:" + ...
                      str(packet_sender.sent_packets)
19                          +":"+hex(packet_sender.seed))
20                packet_sender.send_wus()
21        except SenderHasNoMorePacketsToSend as e:
22            print e
23            Printer.log("Sent all packets.")
24
25        packet_sender.close()
26        sys.exit(0)
```

Code Listing B.5: Add the payload to code for a WuS bit.

B.2 WuTx implementation for the ESP-32

The C-based WuTx implementation is compatible with the API offered by the ESP-32 microcontroller, a widely available embedded microcontroller that incorporates an IEEE 802.11 transceiver. In contrast to the Linux-based implementation, this WuTx offers an API defined by two functions that implement the WuTx functions defined in the `ook_com.h` header file, as referenced in Section 5.5. This code presented here implements the WuTx described in Section 3.8.3, the WuR-CTC IEEE 802.11g WuTx.

First, `wlan_wur_init_context` sets up the device WLAN with the expected bit rate and sets up the local state variables, such as the scrambler values and the IEEE 802.11 headers. Additionally, if the `USE_GPIO` flag is set, the library is configured for transmission of simulated OOK signals through a GPIO port, instead of a WLAN interface. This feature was developed for debugging and integration purposes.

```
1  esp_err_t wlan_wur_init_context(wlan_wur_ctxt_t *wur_context,
2                                  uint8_t initial_state,
3                                  symbol_size_t symbol_size){
4  #ifndef USE_GPIO
5      memset(wur_context->frame_buffer, 0, WLAN_TOTAL_BYTES);
6
7      initial_state &= 0x7f;
8
9      wur_context->initial_scrambler_state = reverse(initial_state) ...
          >> 1;
10     wur_context->current_scrambler_state = ...
          wur_context->initial_scrambler_state;
11
12     switch(symbol_size){
13         case WUR_SIZE_6M:
```

```
14              wur_context->send_bit_fn = set_frame_bit_6_mbps;
15              break;
16          case WUR_SIZE_24M:
17              wur_context->send_bit_fn = set_frame_bit_24_mbps;
18              break;
19          default:
20              printf("Symbol size still not supported!\n");
21              return ESP_ERR_INVALID_ARG;
22      }
23
24      wur_context->symbol_len = symbol_size;
25      wur_context->current_len = 0;
26      switch(symbol_size){
27          case WUR_SIZE_6M:
28              set_wifi_fixed_rate(WIFI_PHY_RATE_6M);
29              esp_wifi_set_max_tx_power(ESP_CUSTOM_POWER_6M);
30              break;
31          case WUR_SIZE_24M:
32              set_wifi_fixed_rate(WIFI_PHY_RATE_24M);
33              esp_wifi_set_max_tx_power(ESP_CUSTOM_POWER_24M);
34              break;
35          default:
36              printf("Rate still not supported!\n");
37              return ESP_ERR_INVALID_ARG;
38      }
39
40      memset(wur_context->scrambler_buffer, 0, WLAN_TOTAL_BYTES);
41      wur_context->current_scrambler_state = ...
            wur_context->initial_scrambler_state;
42      printf("Innitializing frame with scrambler state 0x%02X.\n",
43  wur_context->current_scrambler_state);
44
45      uint8_t symbol_bytes = wur_context->symbol_len/8;
46
47      /* get the padding byte number to align payload to OFDM symbol ...
            start boundary*/
48      uint8_t padding_bytes = (symbol_bytes - ((SIGNAL_FIELD_BYTES + ...
            WLAN_HEADERS_BYTES) %
49  symbol_bytes)) % symbol_bytes;
50      printf("Using %d Padding bytes for a symbol size of %d ...
            bytes.\n", padding_bytes,
51  symbol_bytes);
52
53      uint16_t total_offset_bits = 9 + ((WLAN_HEADERS_BYTES + ...
            padding_bytes)*8);
54      /* advance the scrambler state to include the SIGNAL field and ...
            the non settable byts
55  of the header*/
56      uint8_t scrambler_state = wur_context->current_scrambler_state;
57
58      for(uint16_t i = 0; i < total_offset_bits; i++){
59          uint8_t feedback = 0;
60          feedback = ((!!(scrambler_state & 64))) ^ ...
```

```
            (!!(scrambler_state & 8));
61          scrambler_state = ((scrambler_state << 1) & 0x7e) | feedback;
62      }
63
64      wur_context->current_scrambler_state = scrambler_state;
65      printf("Current scrambler state is 0x%02X.\n", ...
            wur_context->current_scrambler_state);
66
67      /* Create the scrambler values for the whole frame*/
68      /* signal field bytes are scrambled but are not part of the MPDU*/
69      scramble_bytes(wur_context, WLAN_TOTAL_BYTES, ...
            wur_context->scrambler_buffer);
70      memset(&wur_context->frame_buffer[wur_context->current_len], 0, ...
            padding_bytes);
71
72      wur_context->padding_bytes = padding_bytes;
73  #else
74      gpio_pad_select_gpio(OUTPUT_GPIO);
75      /* Set the GPIO as a push/pull output */
76      gpio_set_direction(OUTPUT_GPIO, GPIO_MODE_OUTPUT);
77  #endif
78      gpio_pad_select_gpio(SIGNAL_OUTPUT_GPIO);
79      /* Set the GPIO as a push/pull output */
80      gpio_set_direction(SIGNAL_OUTPUT_GPIO, GPIO_MODE_OUTPUT);
81
82      return ESP_OK;
83  }
```

Code Listing B.6: Initialize the WuS library with the correct values.

Besides the operations directly implemented in its code, **wlan_wur_init_context** uses the **scramble_bytes** function, which prepares cache with pre-scrambled byte values. The scrambler values of any given frame remain constant, as the ESP-32 uses a constant scrambler seed. Thus, this optimization allows reusing those values and speeds up the execution of the library.

```
1   static void scramble_bytes(wlan_wur_ctxt_t *ctxt,
2                              uint16_t byte_len,
3                              uint8_t* payload){
4
5       uint8_t scrambler_state = ctxt->current_scrambler_state;
6
7       /* scramble each byte*/
8       for(uint16_t i = 0; i < byte_len; i++){
9           uint8_t scrambler_byte = 0x00;
10          /* advance scrambler state and XOR rtesult to byte array, ...
                if present*/
11          for(uint8_t j = 0; j < 8; j++){
12              uint8_t feedback = 0, bit = 0;
13              feedback = ((!!(scrambler_state & 64))) ^ ...
                    (!!(scrambler_state & 8));
14              bit = feedback ^ (payload[i] >> (j%8));
```

```
15              scrambler_state = ((scrambler_state << 1) & 0x7e) | ...
                    feedback;
16              /* scrambler sequencer is applied MSB first*/
17              scrambler_byte |= bit << (j%8);
18          }
19          if(payload != NULL){
20              payload[i] = scrambler_byte;
21          }
22      }
23
24      ctxt->current_scrambler_state = scrambler_state;
25  }
```

Code Listing B.7: Initialize a cache with the expected scrambler values.

After initialization, the library is ready to send WuS frames. For this purpose, the **wlan_wur_transmit_frame** is used. As **wlan_wur_init_context**, this function also supports sending frames through GPIO instead of WLAN by defining the `USE_GPIO` flag. The main flow, which contemplates sending WuS frames through the WLAN will be described.

```
1  esp_err_t IRAM_ATTR wlan_wur_transmit_frame(
2                              wlan_wur_ctxt_t *wur_context,
3                              uint8_t* data_bytes,
4                              uint8_t data_bytes_len){
5
6      esp_err_t esp_res;
7
8  #ifndef USE_GPIO
9      esp_res = wlan_wur_init_frame(wur_context);
10     if(esp_res != ESP_OK){
11         printf("Failed to prepare frame of len %d because of %d.\n",
12 wur_context->current_len, esp_res);
13         return ESP_FAIL;
14     }
15     _send_preamble_legacy_wlan(wur_context);
16 #else
17     portENTER_CRITICAL(&wlanGroupMux);
18     SET_OUTPUT;
19     ets_delay_us(20);
20     _send_preamble_legacy_gpio();
21 #endif
22
23     /* now send the actual frame, bit by bit*/
24     for(uint16_t i = 0; i < data_bytes_len; i++){
25         _send_byte_legacy_wlan(wur_context, data_bytes[i]);
26     }
27
28 #ifndef USE_GPIO
29     int32_t res = esp_wifi_internal_tx(ESP_IF_WIFI_STA, ...
           wur_context->frame_buffer,
30 wur_context->current_len);
```

```
31      if(res != ESP_OK){
32          printf("Failed to sand frame of len %d because of %d.\n",
33  wur_context->current_len, res);
34      }
35  #else
36      CLEAR_OUTPUT;
37      portEXIT_CRITICAL(&wlanGroupMux);
38  #endif
39      return ESP_OK;
40  }
```

Code Listing B.8: Send the contents of a byte buffer as a WuS through the WLAN interface.

First, `wlan_wur_transmit_frame` initializes the frame context with `wlan_wur_init_frame`. This auxiliary function re-initializes the frame transmission offset and buffer and, afterward, copies the WLAN MAC headers to the WLAN frame buffer. Finally, it advances the buffer offset past the recently set headers.

```
1   esp_err_t wlan_wur_init_frame(wlan_wur_ctxt_t *wur_context){
2       wur_context->current_len = 0;
3
4       switch(wur_context->symbol_len){
5           case WUR_SIZE_6M:
6               memset(wur_context->frame_buffer, 0, WLAN_TOTAL_BYTES);
7               break;
8           case WUR_SIZE_24M:
9               memset(wur_context->frame_buffer, 0xff, WLAN_TOTAL_BYTES);
10              break;
11          default:
12              printf("Rate still not supported!\n");
13              return ESP_ERR_INVALID_ARG;
14      }
15
16      /* set the header data apropiately*/
17      memcpy(wur_context->frame_buffer, standard_wlan_headers, ...
            WLAN_SETABLE_BYTES);
18      wur_context->current_len += WLAN_SETABLE_BYTES;
19      /* signal field bytes are scrambled but are not part of the MPDU*/
20      wur_context->current_len += wur_context->padding_bytes;
21
22      return ESP_OK;
23  }
```

Code Listing B.9: Reset frame state variables.

Afterward, the WLAN payload bytes encoding the bits encoding the WuR-CTC physical framing are added to the WLAN frame buffer. This is done by repeatedly calling the `send_bit_fn` callback, which is set according to the WLAN bit rate used.

```
1  static void _send_preamble_legacy_wlan(wlan_wur_ctxt_t *wur_ctxt){
2
3      wur_ctxt->send_bit_fn(wur_ctxt, 0);
4      wur_ctxt->send_bit_fn(wur_ctxt, 1);
5      wur_ctxt->send_bit_fn(wur_ctxt, 0);
6      wur_ctxt->send_bit_fn(wur_ctxt, 1);
7      wur_ctxt->send_bit_fn(wur_ctxt, 0);
8      wur_ctxt->send_bit_fn(wur_ctxt, 1);
9      wur_ctxt->send_bit_fn(wur_ctxt, 0);
10     wur_ctxt->send_bit_fn(wur_ctxt, 1);
11     wur_ctxt->send_bit_fn(wur_ctxt, 0);
12     wur_ctxt->send_bit_fn(wur_ctxt, 1);
13     wur_ctxt->send_bit_fn(wur_ctxt, 0);
14     wur_ctxt->send_bit_fn(wur_ctxt, 1);
15     wur_ctxt->send_bit_fn(wur_ctxt, 1);
16 }
```

Code Listing B.10: Send WuR-CTC Physical preambles.

The **send_bit_fn** callback adds the required bytes to send a Peak or Flat Symbol and, afterward, scrambles them with the corresponding pre-scrambled byte values taken from the previously generated cache.

```
1  static void _send_preamble_legacy_wlan(wlan_wur_ctxt_t *wur_ctxt){
2
3  static IRAM_ATTR esp_err_t set_frame_bit_6_mbps(wlan_wur_ctxt_t ...
       *ctxt, uint8_t value){
4
5      if(value){
6          memcpy(&ctxt->frame_buffer[ctxt->current_len], ...
               one_seq_6mbps, 3);
7      }
8
9      if(scramble_payload(ctxt, 3) != ESP_OK){
10         return ESP_FAIL;
11     }
12
13     return ESP_OK;
14
15 }
```

Code Listing B.11: Encode a WuS bit into the WLAN frame buffer.

After adding the preambles, the rest of the frame is added using the **_send_byte_legacy_wlan** function, which takes one byte from the buffer passed as an argument to **wlan_wur_transmit_frame** and codes it into the WLAN frame buffer. As before, this is done by calling the **send_bit_fn** callback for each of the bits contained in the byte passed to the function. Additionally, this function also allows sending the message through a GPIO port by generating 4 μs OOK pulses.

```c

static IRAM_ATTR void _send_byte_legacy_wlan(wlan_wur_ctxt_t ...
    *wur_ctxt, uint8_t byte){
    for(int16_t i = 7; i ≥ 0; i--){
        uint8_t bit;
        bit = (byte & (1 << i)) >> i;
#ifndef USE_GPIO
        wur_ctxt->send_bit_fn(wur_ctxt, bit);
#else
        if(bit){
            SET_OUTPUT;
        }else{
            CLEAR_OUTPUT;
        }
        cdelay(928);
#endif
    }
}
```

Code Listing B.12: Encode a payload byte into the WLAN frame buffer.

Finally, the WLAN frame buffer is sent through the WLAN interface by using the ESP-32 API function `esp_wifi_internal_tx`.

Appendix C

Software-based WuRx baseband implementation

The WuRx baseband used in Chapter 5 comprises a software-based implementation in an STM32L053R8 microcontroller. This piece of software is tasked with sampling and decoding incoming WuR-CTC frames. Afterwards it notifies the host device when a correct frame is received. The WuRx baseband only implements address correlation, therefore, the PSDU of received frames is read by the host device via an I2C bus, being processed there. This section covers the software implementation of the WuRx baseband in Section C.1. Additionally, the bus protocol and I2C implementation are covered in Section C.2.

C.1 WuR-CTC receiver

The WuRx baseband must manage its energy consumption. It awakens when a putative WuR-CTC frame is detected, and returns to sleep after finishing its reception.

The awakening status of the microcontroller follows the state diagram displayed in Fig.C.1. By default, the microcontroller sleeps in S.0, but there are two events that can trigger its wakening. First, a high level is detected in its signal input, which corresponds to a putative WuR-CTC frame, or a pulse in its wake-up line, which is used by the host device to wake the microcontroller before a bus interaction via I2C.

When awakened by a putative WuR-CTC frame reception, the microcontroller goes to S.1, and attempts the decoding of the possible WuR-CTC frame. If the frame fails to decode or decodes with an incorrect CRC-8, the microcontroller returns to S.0, sleeping. If a correct frame is received, the microcontroller will awaken the host

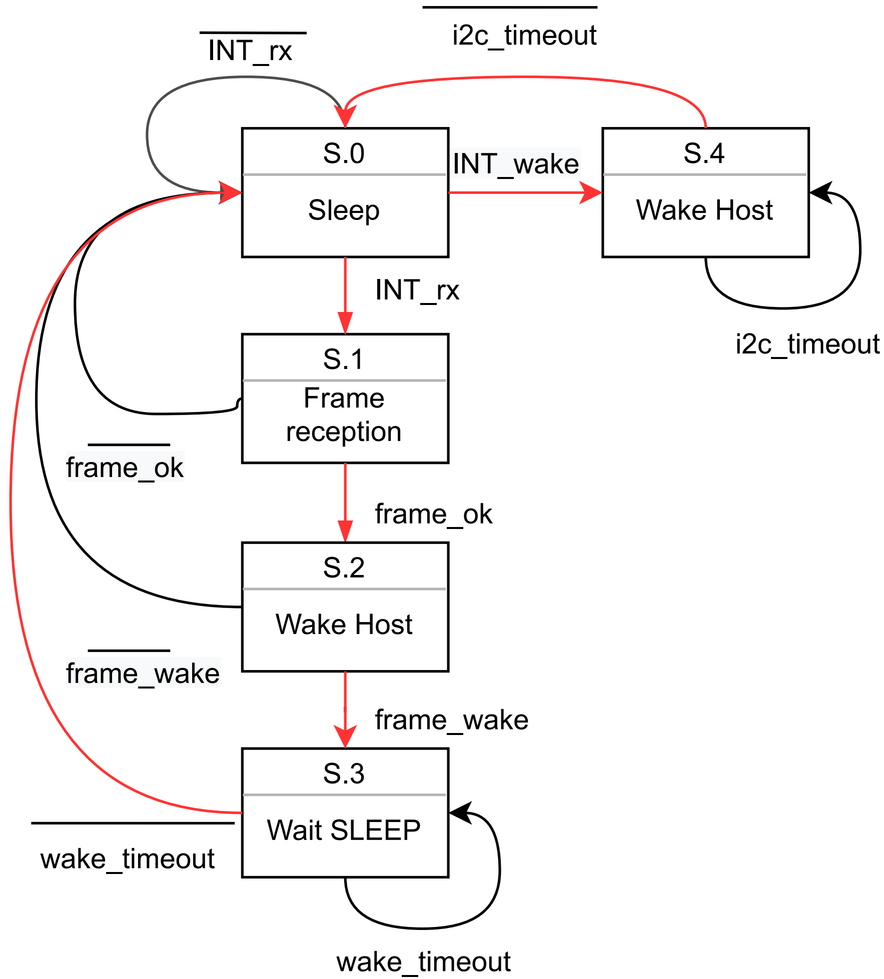


Figure C.1: State diagram of the WuRx baseband activation.

device via an interrupt line in S.2. Afterward, the microcontroller will match the expected state of the host device. If the received frame is not a **WAKE** frame, the microcontroller will return to sleep (S.0). However, if a **WAKE** is received, the microcontroller will remain awakened in S.3, waiting for larger data frames. Additionally, while in S.3, the microcontroller will maintain its precise oscillator circuit activated. While in S.3, the microcontroller does also answer messages sent by the host device via I2C. This state will be maintained until either a timeout is reached, or a **SLEEP** frame is received, returning to S.0.

When awakened by its wake-up line, the microcontroller will reach S.4, waiting for an incoming I2C request by the host device or a timeout. After any of these conditions are met, the device returns to sleep in S.0.

The behavior shown in Fig.C.1 is implemented by the firmware `loopMain` function in a state machine implementation using a switch clause. All the following code

fragments include signaling using the GPIO pin set as `WAKE_UP_FAST` that is used for debugging.

First, S.0 is implemented by the `WUR_SLEEPING` label, which handles sleeping the microcontroller and transitioning to either S.1, or S.4, depending on the input events. To solve possible conflicts, passible WuR-CTC frame reception events have priority over I2C wake-ups. If a WuR-CTC frame reception is started the function calls `WuR_process_frame`, which implements the reception, returning to sleep if the frame reception is not a **WAKE** frame, or fails. The logic switches the next state as `WUR_WAIT_DATA`, representing S.3 in the diagram is a **WAKE** frame is received. Analogously, if the event is an I2C transaction, the logic switches the next state to `WUR_WAIT_I2C`, implementing S.4. In both of these last cases, the length of a timeout is configured in millisecond scale. For a **WAKE** frame reception, the number of wake-up milliseconds specified in the frame is used. For an I2C transaction, a fixed timeout of 10 milliseconds is configured.

```
1  case WUR_SLEEPING:
2      I2C_operation = 0;
3      WuR_operation = 0;
4      WuR_go_sleep(context);
5      /* configure on wakeup for HSI use*/
6      SystemPower_sleep();
7      if(WuR_operation && !I2C_operation){
8          /* WAKE MS for a WAKE frame, will be != 0, it will be ...
               0xFFFF for a
9              SLEEP frame*/
10         wake_status = WuR_process_frame(context, 1);
11         WuR_operation = 0;
12         if(wake_status < 0){
13             break;
14         }
15         if(wake_status > 0){
16             app_wurx_ctxt.wurx_timestamp = wake_status;
17             app_wurx_ctxt.wurx_status = WUR_WAIT_DATA;
18         }
19     }else if(I2C_operation){
20         I2C_operation = 0;
21         app_wurx_ctxt.wurx_timestamp = 10;
22         app_wurx_ctxt.wurx_status = WUR_WAIT_I2C;
23     }else{
24         WuR_operation = 0;
25         I2C_operation = 0;
26     }
27     break;
```

Code Listing C.1: Switch clause controlling wake-up behaviour.

The more complex S.3 state is implemented by the `WAIT_DATA` clause. This clause handles arming the exit timeout at the number of milliseconds specified by the

previous **WAKE** frame. Next, the clause configures the high-speed and accurate oscillator circuit as the clock source. Following, the clause includes a while loop, running until the end of the timeout. This loop handles the same interruptions as `WUR_SLEEPING` clause, using the same priorities, where WuR-CTC frame receptions take over I2C frame reception events. However, if a `SLEEP` frame is received by `WuR_process_frame`, the timeout controlling the loop is set to exit in 10 ms. After the timeout ends, the clause deactivates the high-speed oscillator circuit, sets the state to `WUR_SLEEPING`, and returns the control to the main state machine loop.

```
1  case WUR_WAIT_DATA:
2      /* activate HSE for use. During this interval the ...
          microcontroller will be
3          irresponsive.*/
4      PIN_SET(GPIOA, WAKE_UP_FAST);
5      SystemPower_data();
6      HAL_SuspendTick();
7      /* notify host that we have a frame ready via interrupt and ...
          change state
8          accordingly*/
9      PIN_SET(GPIOA, ADDR_OK);
10     ADJUST_WITH_NOPS;
11     ADJUST_WITH_NOPS;
12     PIN_RESET(GPIOA, ADDR_OK);
13     /* prepare TIM6 to end at the timeout */
14     __TIM6_CLK_ENABLE();
15     TIMER_SET_PERIOD(TIM6, app_wurx_ctxt.wurx_timestamp);
16     TIMER_UIT_ENABLE(TIM6);
17     TIMER_COMMIT_UPDATE(TIM6);
18     TIMER_ENABLE(TIM6);
19     WuR_operation = 0;
20     I2C_operation = 0;
21     pinModeWaitFrame();
22     /* activate again the reception interrupt*/
23     /* loop used inside the state to minimize jitter*/
24     PIN_RESET(GPIOA, WAKE_UP_FAST);
25     while(!IS_TIMER_EXPIRED(TIM6))
26     {
27         if(WuR_operation && !I2C_operation){
28             PIN_SET(GPIOA, WAKE_UP_FAST);
29             pinModeFrameReceived();
30             WuR_operation = 0;
31             PIN_RESET(GPIOA, WAKE_UP_FAST);
32             /* WAKE MS for a WAKE frame, will be != 0, it will be ...
                  0x1 for a
33                 SLEEP frame*/
34             wake_status = WuR_process_frame(context, 0);
35             PIN_SET(GPIOA, WAKE_UP_FAST);
36             if(wake_status < 0){
37                 WuR_operation = 0;
38                 pinModeWaitFrame();
39                 continue;
```

```
40              }
41              else if(wake_status == 0x0001){
42                  /* if that, set the TIM6 to timeout in 10 ms*/
43                  TIMER_SET_PERIOD(TIM6, 10);
44                  TIMER_UIT_ENABLE(TIM6);
45                  TIMER_COMMIT_UPDATE(TIM6);
46              }
47              pinModeWaitFrame();
48              WuR_operation = 0;
49              PIN_SET(GPIOA, ADDR_OK);
50              ADJUST_WITH_NOPS;
51              ADJUST_WITH_NOPS;
52              PIN_RESET(GPIOA, ADDR_OK);
53          }else if(I2C_operation){
54              /* protect I2C transactions from frame interruptions*/
55              PIN_SET(GPIOA, WAKE_UP_FAST);
56              PIN_RESET(GPIOA, WAKE_UP_FAST);
57              pinModeFrameReceived();
58              while(i2Cbusy());
59              I2C_operation = 0;
60              pinModeWaitFrame();
61              PIN_SET(GPIOA, WAKE_UP_FAST);
62          }
63      }
64      CLEAR_TIMER_EXPIRED(TIM6);
65      TIMER_DISABLE(TIM6);
66      __TIM6_CLK_DISABLE();
67      /* deactivate HSE and return to the default clock config with HSI*/
68      SystemPower_wake();
69      app_wurx_ctxt.wurx_status = WUR_SLEEPING;
70      break;
```

Code Listing C.2: Switch clause controlling data frame reception.

Finally, I2C frame reception is controlled by the `WUR_WAIT_I2C` clause. This simpler handler runs to receive a single I2C request or timeout expiration. Once a request is received, the timeout is cut short to 2 milliseconds, which allows the interrupt routine handling I2C transfers to finish.

```
1
2   case WUR_WAIT_I2C:
3       pinModeWaitFrame();
4       /* prepare TIM6 to end at the timeout */
5       __TIM6_CLK_ENABLE();
6       TIMER_SET_PERIOD(TIM6, app_wurx_ctxt.wurx_timestamp);
7       TIMER_UIT_ENABLE(TIM6);
8       TIMER_COMMIT_UPDATE(TIM6);
9       TIMER_ENABLE(TIM6);
10      WuR_operation = 0;
11      I2C_operation = 0;
12      while(!IS_TIMER_EXPIRED(TIM6))
13      {
```

```
14          if(I2C_operation){
15              I2C_operation = 0;
16              TIMER_SET_PERIOD(TIM6, 2);
17              TIMER_COMMIT_UPDATE(TIM6);
18          }
19      }
20      TIMER_DISABLE(TIM6);
21      I2C_operation = 0;
22      reset_i2c_state(&I2cHandle);
23      app_wurx_ctxt.wurx_status = WUR_SLEEPING;
24      break;
```

Code Listing C.3: Switch clause controlling I2C request reception.

The function `WuR_process_frame` implements WuR-CTC frame reception, as defined previously in Fig.5.10. As the previous switch clauses of `loopMain`, this function uses strobes with the `WAKE_UP_FAST` pin for debug purposes.

First, after being called, the function arms a timer for the detection of the WuR-CTC PHY preambles. This function can be called after the device is woken up, or with the device already woken up, as it is done in the `WAIT_DATA` clause. Waking up from sleep takes more clock cycles, thus, when the execution has reached this sentence the window of opportunity for the preamble to start appearing is lower. Therefore, the timeout adjusted is lower if the `WuR_process_frame` is called after the device is woken up.

```
1
2  if(context->wurx_state == WURX_HAS_FRAME){
3      /* notify host via interrupt that a frame is still pending*/
4      return -1;
5  }
6  /*wait 64.25 us for operation completition */
7  context->wurx_state = WURX_DECODING_FRAME;
8
9  PIN_RESET(GPIOA, WAKE_UP_FAST);
10 PIN_SET(GPIOA, WAKE_UP_FAST);
11 PIN_RESET(GPIOA, WAKE_UP_FAST);
12
13 /* wait for preamble init.*/
14 __TIM2_CLK_ENABLE();
15 /* block for 60 us @ 16 ticks x us*/
16 if(from_sleep){
17     preamble_timeout = 650;
18 }
19 else{
20     preamble_timeout = 1300;
21 }
22 TIMER_SET_PERIOD(TIM2, preamble_timeout);
23 TIMER_COMMIT_UPDATE(TIM2);
24 CLEAR_TIMER_EXPIRED(TIM2);
```

```
TIMER_ENABLE(TIM2);
```

Code Listing C.4: Prepare the reception of the WuR-CTC preambles.

Next, the microcontroller begins sampling the input GPIO, waiting for a transition to a low level, which indicates the start of the WuR-CTC preambles. To reduce the possibility of false detection caused by noise, or a transient, the routine requires 5 consecutive samples to be 0 before detecting the start of a possible WuR-CTC preamble. If no low levels are detected, the frame reception is canceled and the function returns an error.

```

/* finish waiting for PHY encapsulation start */
while(!IS_TIMER_EXPIRED(TIM2)){
    PIN_SET(GPIOA, WAKE_UP_FAST);
    PIN_RESET(GPIOA, WAKE_UP_FAST);

    result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
    if(result != 0){
        continue;
    }

    PIN_SET(GPIOA, WAKE_UP_FAST);
    PIN_RESET(GPIOA, WAKE_UP_FAST);
    result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
    if(result != 0){
        continue;
    }

    PIN_SET(GPIOA, WAKE_UP_FAST);
    PIN_RESET(GPIOA, WAKE_UP_FAST);
    result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
    if(result != 0){
        continue;
    }

    PIN_SET(GPIOA, WAKE_UP_FAST);
    PIN_RESET(GPIOA, WAKE_UP_FAST);
    result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
    if(result != 0){
        continue;
    }

    PIN_SET(GPIOA, WAKE_UP_FAST);
    PIN_RESET(GPIOA, WAKE_UP_FAST);

    result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
    if(result == 0){
        preamble_detected = 1;
        if(from_sleep){
            ALIGN_WITH_AWAKE;
```

```
41          }
42          else{
43              ALIGN_WITH_SLEEP;
44          }
45          break;
46      }
47
48  }
49  if(!preamble_detected){
50      PIN_SET(GPIOA, WAKE_UP_FAST);
51      PIN_RESET(GPIOA, WAKE_UP_FAST);
52      PIN_SET(GPIOA, WAKE_UP_FAST);
53      PIN_RESET(GPIOA, WAKE_UP_FAST);
54      return -2;
55  }
```

Code Listing C.5: Detect the start of a WuR-CTC preamble.

Following, the routine starts sampling OOK symbols, searching for the end of the preamble, the frame delimiter. For this purpose, the timer is armed at 63 cycles, which correspond to 4 μs, the symbol period, and the sampling period. Afterward, the routine enters a sampling loop, which ends when the frame delimiter is found.

The delimiter is composed of two consecutive high-level symbols. To quickly discard non-WuR-CTC frames at this stage, the procedure ends with an error if three consecutive low-level symbols are sampled. Additionally, if no delimiter is found before the number of symbols contained in the WuR-CTC PHY preamble, the function returns with an error.

```
1
2   TIMER_SET_PERIOD(TIM2, 63);
3   TIMER_COMMIT_UPDATE(TIM2);
4
5   while(!IS_TIMER_EXPIRED(TIM2));
6   CLEAR_TIMER_EXPIRED(TIM2);
7   for(loop = 0; loop < PREAMBLE_MATCHING_LEN; loop++){
8       while(!IS_TIMER_EXPIRED(TIM2));
9       CLEAR_TIMER_EXPIRED(TIM2);
10      result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
11      PIN_SET(GPIOA, WAKE_UP_FAST);
12      PIN_RESET(GPIOA, WAKE_UP_FAST);
13
14      if(result && last_results[0]){
15          /* We have a preamble match! */
16          break;
17      }
18      else if(!result && !last_results[0] && !last_results[1] && ...
            !last_results[2]){
19          PIN_SET(GPIOA, WAKE_UP_FAST);
20          PIN_RESET(GPIOA, WAKE_UP_FAST);
21          PIN_SET(GPIOA, WAKE_UP_FAST);
```

```
22          PIN_RESET(GPIOA, WAKE_UP_FAST);
23          return -2;
24      }
25
26      last_results[2] = last_results[1];
27      last_results[1] = last_results[0];
28      last_results[0] = result;
29
30      if(loop == PREAMBLE_MATCHING_LEN -1){
31          PIN_SET(GPIOA, WAKE_UP_FAST);
32          PIN_RESET(GPIOA, WAKE_UP_FAST);
33          PIN_SET(GPIOA, WAKE_UP_FAST);
34          PIN_RESET(GPIOA, WAKE_UP_FAST);
35          return -2;
36      }
37  }
```

Code Listing C.6: Search for the frame delimiter.

After sampling the WuR-CTC PHY preambles, the WuR-CTC MPDU follows. It starts with the destination address field. The routine correlates the address field with the device own address, if any mismatch is detected, the function returns with error, detecting a frame not directed to the station. This and all subsequent fields that pertain to the WuR-CTC MPDU are stored into a frame buffer, which will be sent via I2C to the host device after successful frame detection.

```
1
2   /* match address!*/
3   //64 instructions loop at 16MHz
4   for(loop = 0; loop < ADDR_LEN; loop++){
5       while(!IS_TIMER_EXPIRED(TIM2));
6       CLEAR_TIMER_EXPIRED(TIM2);
7       result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
8
9       PIN_SET(GPIOA, WAKE_UP_FAST);
10      PIN_RESET(GPIOA, WAKE_UP_FAST);
11      if(result != context->wurx_address[loop]){
12          PIN_SET(GPIOA, WAKE_UP_FAST);
13          PIN_RESET(GPIOA, WAKE_UP_FAST);
14          PIN_SET(GPIOA, WAKE_UP_FAST);
15          PIN_RESET(GPIOA, WAKE_UP_FAST);
16          PIN_SET(GPIOA, WAKE_UP_FAST);
17          PIN_RESET(GPIOA, WAKE_UP_FAST);
18          WuR_clear_buffer(context);
19          return -3;
20      }
21
22      frame_buffer[offset] = (result != 0);
23      offset++;
24  }
```

Code Listing C.7: Address correlation.

Next, all the WuR-CTC MPDU fields, up to the length field, are read and stored. The received length is stored for the subsequent reception of the payload and CRC-8. If the received length is higher than the specified by the protocol, an error is returned.

```
1
2  /* store sender address*/
3  for(loop = 0; loop < ADDR_LEN; loop++){
4      while(!IS_TIMER_EXPIRED(TIM2));
5      CLEAR_TIMER_EXPIRED(TIM2);
6      result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
7
8      PIN_SET(GPIOA, WAKE_UP_FAST);
9      PIN_RESET(GPIOA, WAKE_UP_FAST);
10     frame_buffer[offset] = (result != 0);
11     offset++;
12 }
13
14
15 /* now decode frame type, 3 bits */
16 while(!IS_TIMER_EXPIRED(TIM2));
17 CLEAR_TIMER_EXPIRED(TIM2);
18 frame_buffer[offset] = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
19 PIN_SET(GPIOA, WAKE_UP_FAST);
20 PIN_RESET(GPIOA, WAKE_UP_FAST);
21 offset++;
22
23 while(!IS_TIMER_EXPIRED(TIM2));
24 CLEAR_TIMER_EXPIRED(TIM2);
25 frame_buffer[offset] = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
26 PIN_SET(GPIOA, WAKE_UP_FAST);
27 PIN_RESET(GPIOA, WAKE_UP_FAST);
28 offset++;
29
30 while(!IS_TIMER_EXPIRED(TIM2));
31 CLEAR_TIMER_EXPIRED(TIM2);
32 frame_buffer[offset] = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
33 PIN_SET(GPIOA, WAKE_UP_FAST);
34 PIN_RESET(GPIOA, WAKE_UP_FAST);
35 offset++;
36
37 /* now decode seq number, one bit */
38 while(!IS_TIMER_EXPIRED(TIM2));
39 CLEAR_TIMER_EXPIRED(TIM2);
40 frame_buffer[offset] = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
41 PIN_SET(GPIOA, WAKE_UP_FAST);
42 PIN_RESET(GPIOA, WAKE_UP_FAST);
43 offset++;
44
45 /* now decode length! */
46
47 for(loop = 0; loop < LENGTH_LEN; loop++){
```

```
48        while(!IS_TIMER_EXPIRED(TIM2));
49        CLEAR_TIMER_EXPIRED(TIM2);
50        result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
51        PIN_SET(GPIOA, WAKE_UP_FAST);
52        PIN_RESET(GPIOA, WAKE_UP_FAST);
53        if(result){
54            length |= 1 << (7 - loop);
55        }
56
57        frame_buffer[offset] = result;
58        offset++;
59    }
60
61    if(length > MAX_LEN_DATA_FRAME){
62        PIN_SET(GPIOA, WAKE_UP_FAST);
63        PIN_RESET(GPIOA, WAKE_UP_FAST);
64        PIN_SET(GPIOA, WAKE_UP_FAST);
65        PIN_RESET(GPIOA, WAKE_UP_FAST);
66        PIN_SET(GPIOA, WAKE_UP_FAST);
67        PIN_RESET(GPIOA, WAKE_UP_FAST);
68        PIN_SET(GPIOA, WAKE_UP_FAST);
69        PIN_RESET(GPIOA, WAKE_UP_FAST);
70        WuR_clear_buffer(context);
71        return -4;
72    }
```

Code Listing C.8: Reception of header fields.

Following, the payload contained in the MSDU is received jointly with the CRC-8. Afterward, the headers, which were stored as bits are converted to a byte array. Finally, the calculated CRC-8 is matched with the received. If a mismatch is detected the frame is discarded and an error is returned.

```
1   /* now we have length, read the rest of bits!*/
2
3   for(byte = 0; byte < length + 1; byte++){
4       uint8_t byte_res = 0;
5
6       for(loop = 0; loop < 8; loop++){
7           while(!IS_TIMER_EXPIRED(TIM2));
8           CLEAR_TIMER_EXPIRED(TIM2);
9           result = READ_PIN(GPIOA, INPUT_FAST, INPUT_FAST_NUM);
10          PIN_SET(GPIOA, WAKE_UP_FAST);
11          PIN_RESET(GPIOA, WAKE_UP_FAST);
12
13          if(result){
14              byte_res |= 1 << (7 - loop);
15          }
16
17          context->frame_buffer[WUR_DATA_OFFSET_BYTES + byte] = byte_res;
18      }
19  }
```

```
20
21 /* well, now the frame is over, let's just process it */
22
23 /* first, move from header bits to bytes! */
24 WuR_set_frame_buffer(context, frame_buffer, 32);
25 context->frame_len += length + 1;
26
27 /* is CRC ok? */
28 if(!WuR_is_CRC_good(context)){
29     PIN_SET(GPIOA, WAKE_UP_FAST);
30     PIN_RESET(GPIOA, WAKE_UP_FAST);
31     PIN_SET(GPIOA, WAKE_UP_FAST);
32     PIN_RESET(GPIOA, WAKE_UP_FAST);
33     PIN_SET(GPIOA, WAKE_UP_FAST);
34     PIN_RESET(GPIOA, WAKE_UP_FAST);
35     PIN_SET(GPIOA, WAKE_UP_FAST);
36     PIN_RESET(GPIOA, WAKE_UP_FAST);
37     WuR_clear_buffer(context);
38     return -4;
39 }
```

Code Listing C.9: Reception of payload and CRC-8.

Finally, the return value of the function is calculated. If the received frame type is a **WAKE/SLEEP** frame, the function returns the number of milliseconds contained in the frame payload, which allows the WuRx to set up the sleep timeout accordingly. However, if neither of the WuS frames is detected, the function returns 0.

```
1
2 /* a WuR WAKE/SLEEP frame is present*/
3 if(length == 2 && ((context->frame_buffer[WUR_FLAGS_OFFSET_BYTES] & ...
      0x0E) == 0b0010)){
4     memcpy(&wake_ms, &context->frame_buffer[WUR_DATA_OFFSET_BYTES], 2);
5     wake_ms = ntohs(wake_ms);
6 }
7
8 context->wurx_state = WURX_HAS_FRAME;
9
10 /* return wake_time to notify the WuRx the time in ms it must stay ...
      awake.*/
11 return wake_ms;
```

Code Listing C.10: Obtaining sleep timeout.

C.2 Communications with the host device

Communications over the I2C bus follow the request/response pattern, where the WuRx microcontroller is always the server, and the host device is the client. Both

devices operate in a dedicated I2C bus, and each of them has a pre-assigned 7-bit address.

To perform a request, the host device must first wake the microcontroller via a dedicated wake ping. Afterward, the host device can start a request. Sadly it was not possible to enable read/write transactions with the STM32L053R8 and host libraries. Specifically, no solution that supported this was found for the ESP-32 native SDK. As a consequence, every request is initiated by a write transaction, where the host specifies what operation it wants to perform. Following, the host can send the frame to perform the solicited operation, which can be contained by an I2C frame with a read or write flag.

The frame used to select operation is the following, as shown in FigC.3. It includes a 1 byte payload, with a 7-bit register identifier followed by a 1-bit write flag. Afterward, the host will send a write or read operation, and its format depends on the operation register used.

Figure C.2: I2C frame to perform register selection.

The microcontroller firmware supports 3 operations, which are identified with 3 registers. Each of them defines a frame format that is used to read or write the corresponding data. These are described by the following enumeration.

1. `I2C_STATUS_REGISTER`: A read-only register that enables reading the state of the WuR-CTC frame reception. The content includes 2 bytes. The first byte contains the frame reception state. It follows an enumeration with 3 states, enumerated by the following identifiers:

   (0) `WURX_NO_FRAME`

   (1) `WURX_DECODING_FRAME`

   (2) `WURX_FRAME_READY`

   The second byte contains, if any, the number of bytes that can be read from the received WuR-CTC frame buffer. The frame format is shown in Fig.C.3.

2. `I2C_ADDR_REGISTER`: A read/write register that allows reading and setting the WuR-CTC address of the microcontroller. The content is the 10-bit WuR-CTC address set to the device, which can be either read or written by the host. The frame format is shown in Fig.C.4.

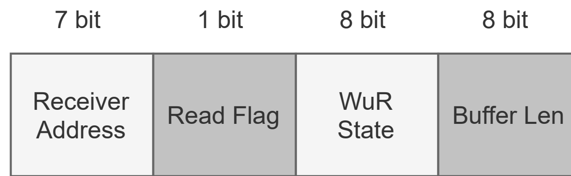


Figure C.3: I2C frame to read WuR-CTC frame reception status.

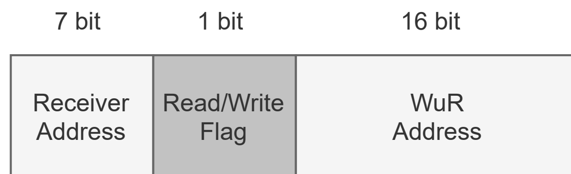


Figure C.4: I2C frame to read or write WuR-CTC address.

3. `FRAME_REGISTER`: A read-only register that includes a buffer storing the last received frame. Includes a number of bytes, limited to the maximum WuR-CTC PSDU. The frame format is shown in Fig.C.5.

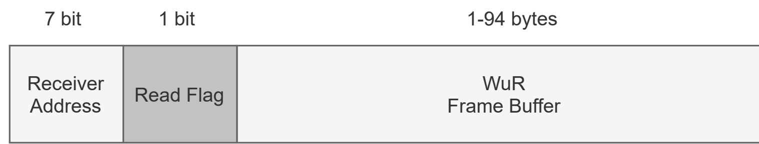


Figure C.5: I2C frame to read WuR-CTC frame buffer.

All the logic handling the I2C interaction is written in a state machine called by Interrupt Service Routine (ISR) handling I2C transactions. Although the state machine is relatively complex, it only reads or writes variables to, or from, device memory and does not include any blocking operation. Hence it is suitable for running inside an ISR without penalizing device performance.

```
1  static void i2c_state_machine(uint8_t i2c_operation, ...
       I2C_HandleTypeDef *I2cHandle){
2      uint8_t register_id;
3      uint8_t operation_id;
4      uint16_t address = 0;
5
6      if(i2c_operation == I2C_ERROR){
7          /* restore to the initial state!*/
8          reset_i2c_coms(I2cHandle);
9          return;
10     }else if(i2c_operation != I2C_SUCCESS_READ && i2c_operation != ...
           I2C_SUCCESS_WRITE){
11         reset_i2c_coms(I2cHandle);
```

```
12          return;
13      }
14
15      switch(i2c_context.i2c_state){
16          case I2C_IDLE:
17              reset_i2c_coms(I2cHandle);
18              break;
19          case I2C_WAITING_OPERATION:
20              register_id = i2c_context.i2c_frame_buffer[0] >> 1;
21              operation_id = i2c_context.i2c_frame_buffer[0] & 0x01;
22
23              if(operation_id){
24                  operation_id = I2C_WRITE_OP;
25              }else{
26                  operation_id = I2C_READ_OP;
27              }
28
29              switch(register_id){
30                  case I2C_STATUS_REGISTER:
31                      if(operation_id == I2C_WRITE_OP){
32                          //write is not supported in this register
33                          reset_i2c_coms(I2cHandle);
34                      }
35                      else{
36                          i2c_context.i2c_frame_buffer[0] = ...
                                wur_context->wurx_state;
37                          i2c_context.i2c_frame_buffer[1] = ...
                                wur_context->frame_len;
38
39                          if(HAL_I2C_Slave_Transmit_IT(I2cHandle, ...
                                (uint8_t*)
40                              i2c_context.i2c_frame_buffer, 2) != HAL_OK)
41                          {
42                          /* Transfer error in transmission process */
43                              reset_i2c_coms(I2cHandle);
44                          }
45                      }
46                      break;
47                  case I2C_ADDR_REGISTER:
48                      if(operation_id == I2C_WRITE_OP){
49                          if(HAL_I2C_Slave_Receive_IT(I2cHandle, ...
                                (uint8_t*)
50                              i2c_context.i2c_frame_buffer, 2) != HAL_OK)
51                          {
52                          /* Transfer error in transmission process */
53                              reset_i2c_coms(I2cHandle);
54                          }
55                      }
56                      else{
57                          address = WuR_get_hex_addr(wur_context);
58                          i2c_context.i2c_frame_buffer[0] = (address ...
                                & 0x0F00) >> 8;
59                          i2c_context.i2c_frame_buffer[1] = address & ...
```

```
                        0xFF;
60
61                    if(HAL_I2C_Slave_Transmit_IT(I2cHandle, ...
                        (uint8_t*)
62                      i2c_context.i2c_frame_buffer, 2) != HAL_OK)
63                    {
64                    /* Transfer error in reception process */
65                        reset_i2c_coms(I2cHandle);
66                    }
67                }
68                break;
69            case I2C_FRAME_REGISTER:
70                if(operation_id == I2C_WRITE_OP){
71                    //write is not supported in this register
72                    reset_i2c_coms(I2cHandle);
73                }
74                uint8_t frame_len;
75
76                if(wur_context->frame_len == 0){
77                    frame_len = 1;
78                }else{
79                    frame_len = wur_context->frame_len;
80                }
81
82                memcpy((uint8_t*)i2c_context.i2c_frame_buffer,
83                    (uint8_t*)wur_context->frame_buffer, ...
                        frame_len);
84                if(HAL_I2C_Slave_Transmit_IT(I2cHandle, (uint8_t*)
85                    i2c_context.i2c_frame_buffer, frame_len) != ...
                        HAL_OK)
86                {
87                /* Transfer error in transmission process */
88                    reset_i2c_coms(I2cHandle);
89                }
90                WuR_clear_context((wurx_context_t*)wur_context);
91                break;
92
93            default:
94                reset_i2c_coms(I2cHandle);
95                return;
96        }
97        /* successful start of read/write operation, save ...
              status for completition.*/
98        if(operation_id == I2C_WRITE_OP){
99            i2c_context.i2c_state = I2C_PERFORM_WRITE;
100       }else{
101           i2c_context.i2c_state = I2C_PERFORM_READ;
102       }
103
104       i2c_context.i2c_last_reg = register_id;
105       i2c_context.i2c_last_operation = operation_id;
106       break;
107   case I2C_PERFORM_WRITE:
```

```
108             if(i2c_operation != I2C_SUCCESS_READ){
109                 //wrong operation for the current state!
110                 reset_i2c_coms(I2cHandle);
111                 break;
112             }
113             if(i2c_context.i2c_last_reg == I2C_NONE_REGISTER){
114                 /* Should not happen,operation not started?!*/
115                 reset_i2c_coms(I2cHandle);
116                 break;
117             }
118             switch(i2c_context.i2c_last_reg){
119                 case I2C_ADDR_REGISTER:
120                     address = 0;
121                     address |= (i2c_context.i2c_frame_buffer[0] & ...
                            0x03) << 8;
122                     address |= i2c_context.i2c_frame_buffer[1];
123                     WuR_set_hex_addr(address, wur_context);
124                     break;
125                 default:
126                     //Operation not supported on write access
127                     break;
128             }
129             reset_i2c_state(I2cHandle);
130             break;
131         case I2C_PERFORM_READ:
132             if(i2c_operation != I2C_SUCCESS_WRITE){
133                 //wrong operation for the current state!
134                 reset_i2c_coms(I2cHandle);
135                 break;
136             }
137             if(i2c_context.i2c_last_reg == I2C_NONE_REGISTER){
138                 /* Should not happen,operation not started?!*/
139                 reset_i2c_coms(I2cHandle);
140                 break;
141             }
142             switch(i2c_context.i2c_last_reg){
143                 case I2C_FRAME_REGISTER:
144                     /* as frame has been read, we can flush it and ...
                            reset the start of the
145                     machine*/
146                     break;
147                 default:
148                     break;
149             }
150             reset_i2c_state(I2cHandle);
151             break;
152         default:
153             break;
154     }
155 }
```

Code Listing C.11: I2C state machine.

Appendix D

WuR-CTC host software implementation

The host devices used in the WuR-CTC use software to handle the LLC layer of the WuR-CTC protocol. The software implementing LLC was developed as a library that is hardware agnostic, maximizing code sharing between the heterogeneous host devices featured in the WuR-CTC testbed. Its implementation is discussed next, in Section D.1.

To produce results, a demonstration scenario of WuR-CTC was developed for both ESP-32 and EFR32MG12 devices. The scenario implements a data transfer over WuR-CTC and provides the means to control and validate the process. Its implementation details are discussed in Section D.2.

D.1 WuR-CTC stack implementation

To support the ESP-32 and EFR32MG12 the library uses a Hardware Abstraction Layer (HAL) that abstracts interactions with the platform software libraries. The HAL comprises a configuration header file, `lib_conf.h`, where a variable sets the target platform.

```
1
2 #define _LIB_CONF_H_
3
4 #define USE_ESP_VERSION
5 //#define USE_EFR_VERSION
6
7 #if (defined USE_ESP_VERSION) && defined(USE_EFR_VERSION)
8 #error "Only one target platform can be used."
```

```
9  #endif
10
11 #if !(defined USE_ESP_VERSION) && !defined(USE_EFR_VERSION)
12 #error "Please, chose one valid traget platform at the start of ...
       lib_conf.h."
13 #endif
14
15 /* add common ESP_32 includes*/
16 #ifdef USE_ESP_VERSION
17
18 #include "esp_system.h"
19 #include "driver/i2c.h"
20 #include "freertos/FreeRTOS.h"
21 #include "freertos/semphr.h"
22 #include "esp_wifi.h"
23
24 #define I2C_WAKEUP_GPIO GPIO_NUM_19
25
26 #define USE_FREERTOS
27 #endif /* USE_ESP_VERSION */
28
29 /* add common EFR_32 includes*/
30 #ifdef USE_EFR_VERSION
31 #define NO_RTOS
32
33 #include "em_cmu.h"
34 #include "ustimer.h"
35 #include "hal-config.h"
36 #include PLATFORM_HEADER
37 #include CONFIGURATION_HEADER
38 #include <stdio.h>
39 #define ntohs(x) __ntohs(x)
40 #define htons(x) __htons(x)
41 #endif /*USE_EFR_VERSION*/
```

Code Listing D.1: WuR-CTC library configuration header.

Additionally, the `lib_conf.h` abstracts away the presence of an Real Time Operating System (RTOS) by conditionally defining RTOS primitives, such as mutexes and queues. If no RTOS is configured, those are left as void implementations, otherwise, they are defined using the equivalent RTOS primitives.

```
1
2  #ifdef USE_FREERTOS
3
4  #define WuRBinarySemaphoreHandle_t SemaphoreHandle_t
5  #define WuRRecursiveMutexHandle_t SemaphoreHandle_t
6
7  #define WuRBinarySemaphoreCreate() xSemaphoreCreateBinary()
8  #define WuRBinarySemaphoreGive(x) xSemaphoreGive(x)
9  #define WuRBinarySemaphoreTake(x, y) xSemaphoreTake(x, y)
10
```

```
11 #define WuRRecursiveMutexCreate() xSemaphoreCreateRecursiveMutex()
12 #define WuRRecursiveMutexGive(x) xSemaphoreGiveRecursive(x)
13 #define WuRRecursiveMutexTake(x, y) xSemaphoreTakeRecursive(x, y)
14
15 #define WuRTaskCreate(task_function, task_name, stack_size, args_p, ...
       priority,
16 task_handle)
17 xTaskCreate(task_function, task_name, stack_size, args_p, priority, ...
       task_handle)
18
19 #define WuRTickPeriodMS portTICK_PERIOD_MS
20 #define WuRMaxDelayMS portMAX_DELAY
21
22 #else /* USE_FREERTOS */
23
24 #define WuRBinarySemaphoreHandle_t uint32_t
25 #define WuRRecursiveMutexHandle_t uint32_t
26
27 #define WuRBinarySemaphoreCreate() 0
28 #define WuRBinarySemaphoreGive(x)
29 #define WuRBinarySemaphoreTake(x, y)
30
31 #define WuRRecursiveMutexCreate() 0
32 #define WuRRecursiveMutexGive(x)
33 #define WuRRecursiveMutexTake(x, y)
34
35 #define WuRTaskCreate(task_function, task_name, stack_size, args_p, ...
       priority,
36 task_handle)
37
38 #define WuRTickPeriodMS 1
39
40 #define WuRMaxDelayMS 1
41
42 #endif /* USE_FREERTOS*/
```

Code Listing D.2: WuR-CTC RTOS configuration.

Other HAL functions include I2C transactions, WuTx interactions, GPIOs, and ISRs. Those are defined in the `i2c_com.h` and `ook_com.h` headers. Those definitions are implemented in a platform-specific manner in the respective "c" source files, one for each platform. The platform-specific sources for each supported platform are contained in the `esp_32` and `efr_32` folders, which are conditionally compiled.

The rest of the library is relatively generic, with few snippets that are conditionally compiled, depending on the target platform. The non-platform-specific functions that implement sending and receiving WuR-CTC messages over the WuTx are implemented in the `ook_wur.c` source file, while those comprising I2C interactions are implemented in the `i2c_wur.c` source file. Finally, `wur.c` implements the state machine handling the WuR-CTC stack implementation. This part of the implementation is RTOS aware and uses mutexes to allow concurrent access to its

functionality.

The WuR-CTC functionality is exposed to clients through the `wur.h` header that contains the functions and callbacks that are exposed to the client software. The `wur_init` function initializes the WuR-CTC library and its associated structures. This function must be called before starting any WuR-CTC interaction. With RTOS platforms, `wur_init` starts running the WuR-CTC state machine in its own thread.

The header exposes several functions to allow clients to send WuR-CTC messages:

- `wur_send_wake`: send **WAKE/SLEEP frames**, to a given address, using a return-to-sleep timeout in milliseconds passed as the argument `ms`.

```
1 wur_tx_res_t wur_send_wake(uint16_t addr, uint16_t ms);
```

Code Listing D.3: `wur_send_wake` signature.

- `wur_send_data`: send **DATA frames** and **DATA+ACK frames** to a given address.

```
1 wur_tx_res_t wur_send_data(uint16_t addr, uint8_t* data,
2                     uint8_t data_len, uint8_t is_ack,
3                     int8_t ack_seq_num);
```

Code Listing D.4: `wur_send_data` signature.

  Where the `addr` argument contains the target address, `data` and `data_len` the buffer to be sent and its size, `is_ack` can be set to configure the frame as **DATA+ACK**, and, finally, `ack_seq_num` passes the sequence flag. This last argument can be set as "-1" to use the expected `is_ack` as tracked by the library.

- `wur_send_ack`: send **ACK frames**, to a given address.

```
1 wur_tx_res_t wur_send_ack(uint16_t addr, int8_t ack_seq_num);
```

Code Listing D.5: `wur_send_ack` signature.

  In the same way as `wur_send_data`, the `ack_seq_num` can be set to force a certain sequence number in the outgoing frame.

Since the library can run with targets that do not use any RTOS, it offers an asynchronous callback-based interface to receive events. Clients can add their own callbacks to receive frame transmission and reception events using the following functions:

- `wur_set_tx_cb`: subscribe to a callback that will be called when the result of a frame transmission is known. The callback needs to present the following signature.

```
1 typedef void (*wur_tx_cb) (wur_tx_res_t tx_res);
```

Code Listing D.6: `wur_tx_cb` signature.

Where `wur_tx_res_t` is an enumeration containing the following cases:

```
1 typedef enum wur_tx_res{
2     WUR_ERROR_TX_OK = 0,
3     WUR_ERROR_TX_ACK_DATA_TIMEOUT = −1,
4     WUR_ERROR_TX_ACK_WAKE_TIMEOUT = −2,
5     WUR_ERROR_TX_NACK = −3,
6     WUR_ERROR_TX_FAILED = −4,
7     WUR_ERROR_TX_BUSY = −5
8 }wur_tx_res_t;
```

Code Listing D.7: `wur_tx_res_t` enumeration.

- `wur_set_rx_cb`: subscribe to a callback that will be called when a non-ACK frame is received. The callback will be passed needs to present the following signature.

```
1 typedef void (*wur_rx_cb) (wur_rx_res_t rx_res, uint8_t* ...
     rx_bytes, uint8_t rx_bytes_len);
```

Code Listing D.8: `wur_rx_cb` signature.

Where `wur_rx_res_t` is an enumeration containing:

```
1 typedef enum wur_rx_res{
2     WUR_ERROR_RX_OK = 0,
3     WUR_ERROR_RX_FAILED = −1
4 }wur_rx_res_t;
```

Code Listing D.9: `wur_rx_res_t` enumeration.

And `rx_bytes` is a stack-resident byte buffer with the received payload bytes. Finally, `rx_bytes_len` codes the length of the received buffer.

The main loop of the protocol, handling timeouts and the WuR-CTC stack logic is implemented in the `wur_tick`. In platforms without RTOS, the user must call this function repetitively, preferably at constant intervals. In RTOS platforms, this `wur_tick` is called at constant intervals in an associated thread started by `wur_init`.

The code of `wur_tick` handles timeouts, frame reception over I2C, and calls the associated client callbacks when a suitable message is required.

D.2 WuR-CTC demo scenario

To evaluate a file transmission using the WuR-CTC protocol, two demonstration applications using the WuR-CTC library were developed. In contrast to the WuR-CTC library, those make extensive use of their platform-specific libraries. In this Section, the implementation details for both platforms will be explained.

Nonetheless, both of them implement a scenario that includes a wake-up of another WuR-CTC station, sending 100 frames containing 64 bytes of data each, and finally, sending the receiving station to sleep. This functionality is coded into a state machine, which code is largely shared between the two platforms.

Additionally, the demonstration application provides the means to start the test scenario and to visualize the results in a simple way. For this purpose, both applications expose the controls for the test scenario using communication interfaces, a minimal web app in ESP-32, and a CoAP-based API in the EFR32MG12.

D.2.1 Event loop

The event loop of the demo application comprises several states, which enable a transmitter to wake up a receiver, send 100 **DATA frames**, and put the receiver to sleep. The receiver will just ACK received frames, as it is required and implemented in the WuR-CTC stack implementation. Moreover, the transmitter also tracks the run time of the test and keeps a score of frame dropouts, which are those not ACKed by the receiver.

The state machine handling the transmitter demo logic incorporates several states, that, on a correct demo execution, occur in an almost-sequential manner, as shown in Fig.D.1. These cover the wake-up of the other station and include an internal loop that sends the test frames and annotates the result of each transmission.

The code sections implementing the state machine described in Fig.D.1 are implemented in two functions, shared between both implementations. These are, `WuRAppTick`, handling timeouts and non-message-related state transitions, and `_wur_tx_cb`, handling state transitions that depend on a message transmission outcome.

```
1
2 void WuRAppTick(void){
3
4     uint32_t current_timestamp = halCommonGetInt32uMillisecondTick();
5     uint16_t wur_addr, wake_ms;
6     wur_tx_res_t tx_res;
7
8     wur_tick(current_timestamp);
```

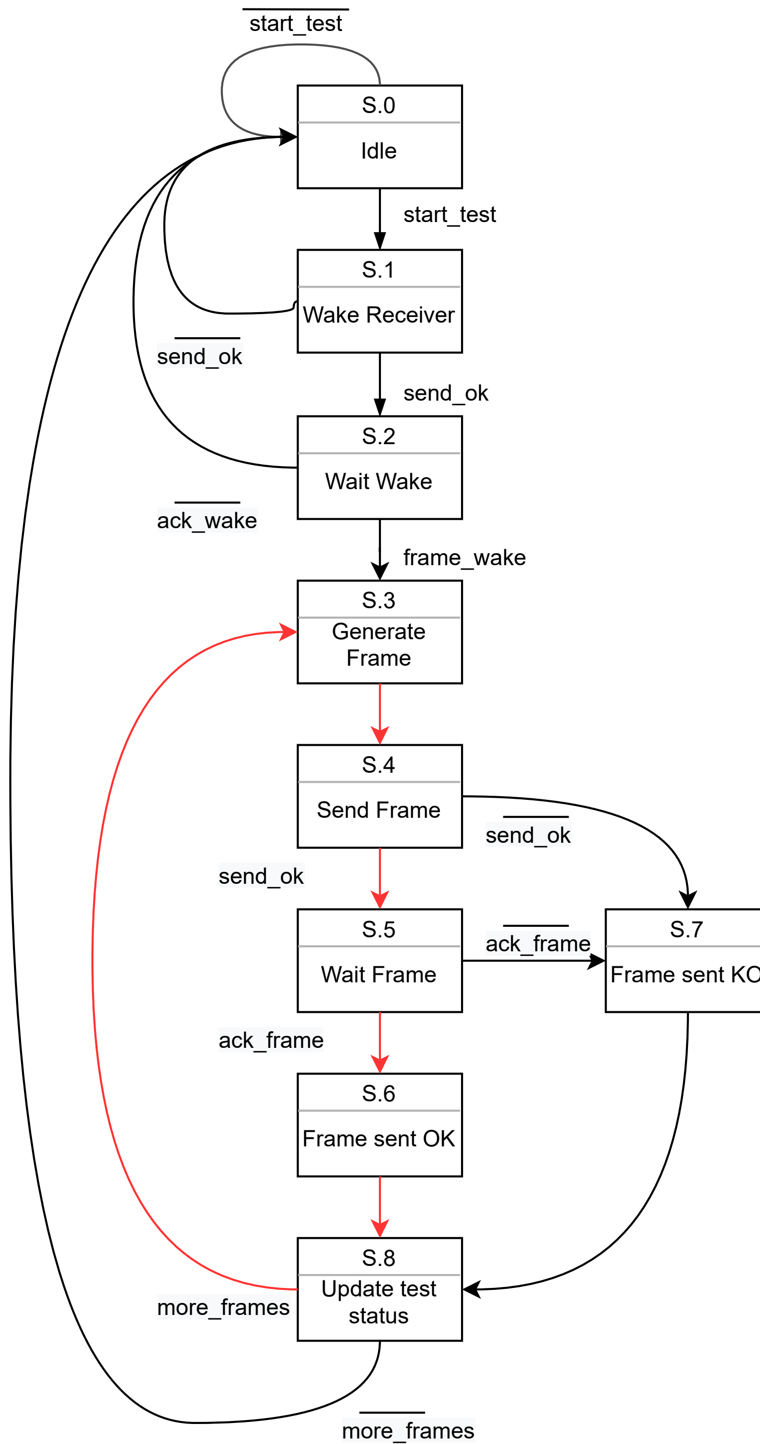


Figure D.1: State diagram of the transmitter demonstration. The internal loop that sends and validates the test frames is highlighted in red

```
9
10      switch(app_ctxt.app_status){
```

```
11          case APP_IDLE:
12              if(current_timestamp % 10000 == 0){
13                  emberAfCorePrintln("[%d]: Device IDLE", ...
                        current_timestamp);
14              }
15              break;
16
17          /* ... */
18
19
20
21          case TEST_SENDING_WAKE:
22              printf("[%d]: Sending Test Wake Device REQ!\n", ...
                    current_timestamp);
23              wur_addr = TEST_ADDR & (0x03FF);
24              wake_ms = TEST_WAKE_INTERVAL;
25
26              app_ctxt.app_status = TEST_WAITING_WAKE;
27              tx_res = wur_send_wake(wur_addr, wake_ms);
28              if(tx_res != WUR_ERROR_TX_OK){
29                  printf("[%d]: Failure to send Wake Device REQ!\n", ...
                        current_timestamp);
30                  _respondWithError(APP_TRANS_KO_TX);
31                  app_ctxt.app_status = TEST_COMPLETE_FAILURE;
32                  break;
33              }
34              printf("[%d]: Sent Test Wake Device REQ!\n", ...
                    current_timestamp);
35              break;
36
37          case TEST_WAITING_WAKE:
38              //printf("Waiting frame!");
39              break;
40
41
42          case TEST_GENERATE_FRAME:
43              printf("Generating frame!");
44              generate_test_frame(test_data_buf, TEST_FRAME_SIZE);
45              app_ctxt.app_status = TEST_SEND_FRAME;
46              break;
47
48          case TEST_SEND_FRAME:
49              printf("[%d]: Sending Data to Device test, frame %d/%d!\n",
50              current_timestamp, test_ctxt.current_frame, ...
                    test_ctxt.total_frames);
51              wur_addr = TEST_ADDR & (0x03FF);
52              app_ctxt.app_status = TEST_WAIT_FRAME;
53              tx_res = wur_send_data(wur_addr, ...
                    (uint8_t*)&test_data_buf, TEST_FRAME_SIZE,
54                  false, -1);
55              if(tx_res != WUR_ERROR_TX_OK){
56                  printf("[%d]: Failure to send Data to Device ...
                        REQ!\n", current_timestamp);
```

```
57                  update_text_context(&test_ctxt, false);
58                  app_ctxt.app_status = TEST_GENERATE_FRAME;
59              }
60              printf("[%d]: Sent Test Data Device REQ!\n", ...
                   current_timestamp);
61              break;
62
63          case TEST_WAIT_FRAME:
64              //printf("Wait Frame!");
65              break;
66
67          case TEST_COMPLETE_OK_FRAME:
68              printf("Frame sent OK!");
69              update_text_context(&test_ctxt, true);
70              if(test_ctxt.test_status == TEST_IN_PROGRESS){
71                  //printf("Prepare next frame!");
72                  app_ctxt.app_status = TEST_GENERATE_FRAME;
73              }
74              break;
75          case TEST_COMPLETE_KO_FRAME:
76              printf("Frame sent KO!");
77              update_text_context(&test_ctxt, false);
78              if(test_ctxt.test_status == TEST_IN_PROGRESS){
79                  printf("Prepare next frame!");
80                  app_ctxt.app_status = TEST_GENERATE_FRAME;
81              }
82              break;
83          case TEST_COMPLETE_FAILURE:
84              printf("Test failure!");
85              fail_test_context(&test_ctxt, "Error while taking the ...
                   test.\n",
86                  current_timestamp);
87              app_ctxt.app_status = APP_IDLE;
88              break;
89          default:
90              break;
91      }
92  }
```

Code Listing D.10: `WuRAppTick` implementation snippet handling WuR-CTC demo.

```
1
2   static void _wur_tx_cb(wur_tx_res_t tx_res){
3       uint32_t current_timestamp = halCommonGetInt32uMillisecondTick();
4
5       switch(app_ctxt.app_status){
6
7           /* ... */
8
9           case TEST_WAITING_WAKE:
10              if(tx_res == WUR_ERROR_TX_OK){
11                  app_ctxt.app_status = TEST_GENERATE_FRAME;
```

```
12              }
13              else{
14                  app_ctxt.app_status = TEST_COMPLETE_FAILURE;
15              }
16              break;
17          case TEST_WAIT_FRAME:
18              if(tx_res == WUR_ERROR_TX_OK){
19                  printf("[%d]: Received ACK!\n", current_timestamp);
20                  app_ctxt.app_status = TEST_COMPLETE_OK_FRAME;
21              }
22              else if(tx_res == WUR_ERROR_TX_ACK_DATA_TIMEOUT){
23                  printf("[%d]: Received NACK!\n", current_timestamp);
24                  app_ctxt.app_status = TEST_COMPLETE_KO_FRAME;
25              }
26              else{
27                  printf("[%d]: Received Error!\n", current_timestamp);
28                  app_ctxt.app_status = TEST_COMPLETE_FAILURE;
29              }
30              memset(app_ctxt.app_data_buf, 0, MAX_APP_DATA_BUF);
31              app_ctxt.app_data_buf_len = 0;
32              break;
33          default:
34              printf("[%d]: Received ACK while not waiting. Is this ...
                   an error?!",
35                  current_timestamp);
36              break;
37      }
38  }
```

Code Listing D.11: `_wur_tx_cb` implementation snippet handling WuR-CTC demo.

The test progress and results are contained in the `test_ctxt` structure, which is updated with the results by calling `update_text_context` every time that the outcome of a message is known. This same function ends the test when the target number of frame transmissions has been logged.

```
1
2  static void update_text_context(test_ctxt_t* ctxt, bool OK_result){
3      ctxt->current_frame++;
4      if(OK_result){
5          ctxt->OK_frames++;
6      }
7      else{
8          ctxt->KO_frames++;
9      }
10     if(ctxt->current_frame == ctxt->total_frames){
11         aprove_test_context(ctxt, halCommonGetInt32uMillisecondTick());
12     }
13 }
```

Code Listing D.12: `update_text_context` implementation.

The `test_ctxt_t` stores all the information related to the execution of the test, which is later accessed and sent to clients using platform specific methods.

```

typedef struct test_ctxt{
    test_status_t         test_status;
    uint16_t              total_frames;
    uint16_t              current_frame;
    uint16_t              OK_frames;
    uint16_t              KO_frames;
    uint32_t              start_timestamp;
    uint32_t              finish_timestamp;
    char                  failure_reason[TEST_REASON_LEN];
}test_ctxt_t;
```

Code Listing D.13: `test_ctxt_t` structure definition.

D.2.2 Control and display interfaces

The methods to start and display the status of the WuR-CTC test are specific to each platform. Both of them use different communication stacks and present diverging implementation constraints.

ESP-32

The ESP-32 demo implementation exposes an HTTP-based interface that can start a test scenario and poll the results of a test still ongoing, or finished. For this purpose, two HTTP endpoints are exposed.

- `POST /wur/test/start`: Requests the start of a WuR-CTC demonstration. The request produces the following responses, depending on the state:
  - If no WuR-CTC demonstration is currently running, a new demonstration is started. The response is a 200 (OK) code, with a JSON in the body with the vale: {"ok": true }
  - If a WuR-CTC demonstration is currently running, the operation fails with an error 500.
- `GET /wur/test/status`: Requests the run status of the WuR-CTC demonstration. The response contains a JSON with the following format.

```
{
```

```
2       "status": String ("idle", "in_progress", ...
            "finished", "failed"),
3       "currentFrame": Integer,
4       "totalFrames": Integer,
5       "framesOK": Integer,
6       "framesKO": Integer,
7       "runTime": Number
8   }
```

Code Listing D.14: /wur/test/status response format

For validation purposes, the ESP-32 demo includes endpoints to perform additional functionality. These include:

- `POST /wur/data`: Sends a WuR-CTC **DATA frame** to a station. The request must contain a JSON body with the following format:

```
1   {
2       "wur_address": String. Device address in ...
            hexadecimal format,
3       "data": String. Data buffer to be sent in ...
            hexadecimal format
4   }
```

Code Listing D.15: /wur/data request body format

The response values can be the following:

  - 200 (OK). The frame was successfully sent and acknowledged by the receiving station. The response contains a plaintext body with the hexadecimal representation of the data, if any, responded by the receiving station in the ACK frame.
  - 400 (Bad Request) The contents of the request could not be properly parsed. A plaintext string with more details is added to the body of the response.
  - 412 (Precondition Failed) The device is already busy with another task. A plaintext string with more details is added to the body of the response.

- `POST /wur/wake`: Sends a WuR-CTC **WuS frame** to a station. The request must contain a JSON body with the following format:

```
1   {
```

```
2     "wur_address": String. Device address in ...
        hexadecimal format,
3     "wake_time": Integer. Wake-up frame payload ...
        for the receiver, in milliseconds.
4 }
```

Code Listing D.16: /wur/wake request body format

A value of "wake_time" of "1" is used to trigger a **SLEEP frame**. Other values trigger the transmission of a WAKE frame with a configurable receiver sleep timeout set to "wake_time" milliseconds.

The response values can be the following:

- 200 (OK). The frame was successfully sent and acknowledged by the receiving station. The response contains a plaintext body with the hexadecimal representation of the data, if any, responded by the receiving station in the ACK frame.
- 400 (Bad Request) The contents of the request could not be properly parsed. A plaintext string with more details is added to the body of the response.
- 412 (Precondition Failed) The device is already busy with another task. A plaintext string with more details is added to the body of the response.

The ESP-32 demonstration firmware provides an embedded webapp that allows the HTTP API previously defined over any HTTP browser compliant with Javascript. A screenshot of the interface is shown in Fig.D.2.

With the webapp, a user can easily start a WuR-CTC demonstration test run with the "Start Test" button displayed at the bottom of the interface shown in Fig.D.2. Afterward, the test status will be periodically polled and updated by the webapp. The user can check the progress and the results under the "Test Status" textual label.

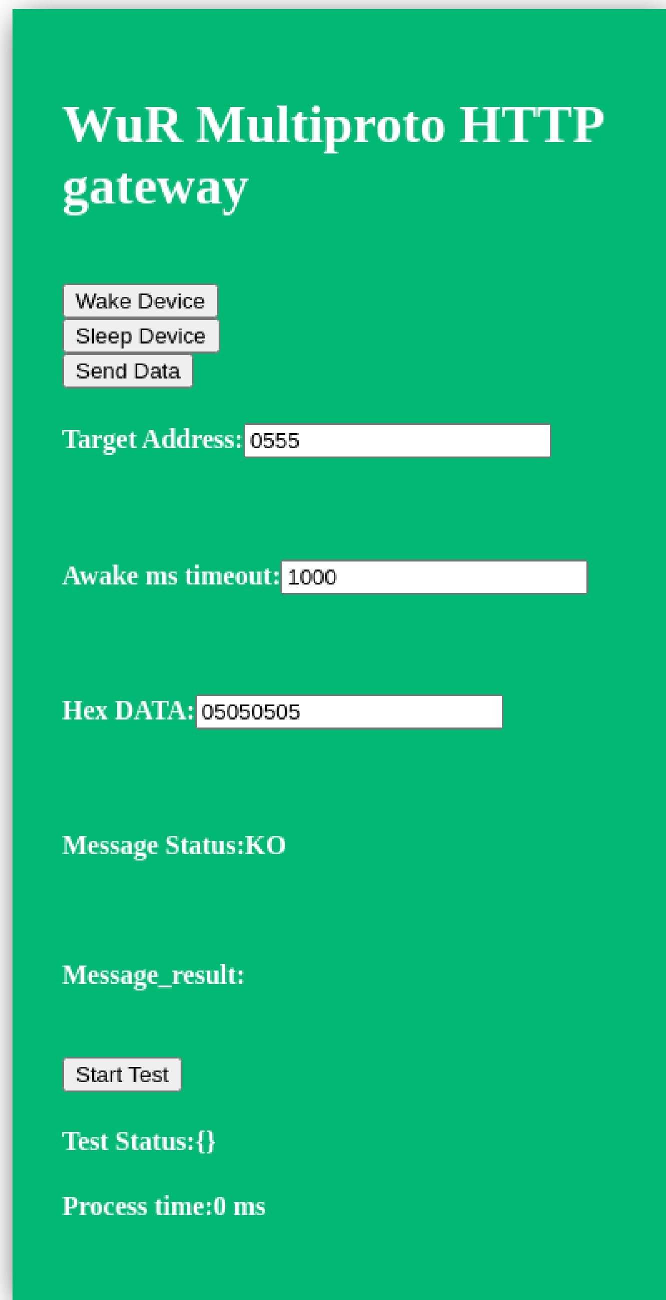


Figure D.2: Screenshot of the embedded ESP-32 control webapp

EFR32MG12

A CoAP interface is used to expose WuR-CTC demonstration controls and results. These implement a similar request/response-based interface as the one defined with the ESP-32, with the same endpoints and methods. Nonetheless, to support the

more constrained environment, the message format used in requests and responses is binary.

To control the WuR-CTC demonstration, two CoAP endpoints are used:

- `POST /wur/test/start`: Requests the start of a WuR-CTC demonstration. The request produces the following responses, depending on the state:
  - If no WuR-CTC demonstration is currently running, a new demonstration is started. The response is a 203 (valid) code, with an empty body.
  - If a WuR-CTC demonstration is currently running, the operation fails with an error 412, with the body field left empty.
- `GET /wur/test/status`: Requests the run status of the WuR-CTC demonstration. The response contains a binary encoded response with code 203 (VALID). The body contains the following successive fields, all coded in Big Endian.

```
{
    uint8_t test_status,        //Contains the test status ...
        coded as an integer
    uint16_t current_frame,     //Current frame offset
    uint16_t total_frames,      //Total number of frames to be sent
    uint16_t ok_frames,         //Total number of acknowledged
    uint16_t ko_frames,         //Total number of non-acknowledged
    uint32_t timestamp          //Test runtime, in milliseconds.

}
```

Code Listing D.17: /wur/test/status response format

For validation purposes, the EFR32MG12 demo includes endpoints to perform additional functionality. These include:

- `POST /wur/data`: Sends a WuR-CTC **DATA frame** to a station. The request must contain a body with the following format:

```
{
    uint16_t wur_address,       //Contains the target WuR-CTC ...
        address
    uint8_t wur_data[],         //Contains the buffer sent with ...
        the frame
}
```

Code Listing D.18: /wur/data request body format

The response values can be the following:

  - 203 (VALID). The frame was successfully sent and acknowledged by the receiving station. The response contains a plaintext body with the data, if any, responded by the receiving station in the ACK frame.
  - 400 (Bad Request) The contents of the request could not be properly parsed.
  - 412 (Precondition Failed) The device is already busy with another task.

- `POST /wur/wake`: Sends a WuR-CTC **WuS frame** to a station. The request must contain a JSON body with the following format:

```
1 {
2     uint16_t wur_address,      //Contains the target WuR-CTC ...
         address
3     uint16_t wake_time,        //Contains the intent of the ...
         WuS frame
4 }
```

Code Listing D.19: /wur/wake request body format

A value of “wake_time” of “1” is used to trigger a **SLEEP frame**. Other values trigger the transmission of a WAKE frame with a configurable receiver sleep timeout set to “wake_time” milliseconds.

In the same way as with the ESP-32, the EFR32MG12 CoAP API can be accessed with a simple application. Nonetheless, due to the protocol constraints of the device, the endpoint is another IEEE 802.15.4 device. For this purpose, a simple demonstration firmware was programmed in an EFR32MG12 development board, which includes user interactive elements, such as buttons and a monochrome screen, as can be seen in Fig.D.3

The firmware offers a simple button-based front-end to the CoAP API provided by the device running the WuR-CTC demonstration firmware, which runs on another EFR32MG12 device. The user can start a WuR-CTC Demonstration run by pressing PB0 on the main screen of the application (See Fig.D.4). Afterward, the progress and results will be displayed on the screen as soon as they become available (See Fig.D.5).

Figure D.3: The EFR32MG12 development board.

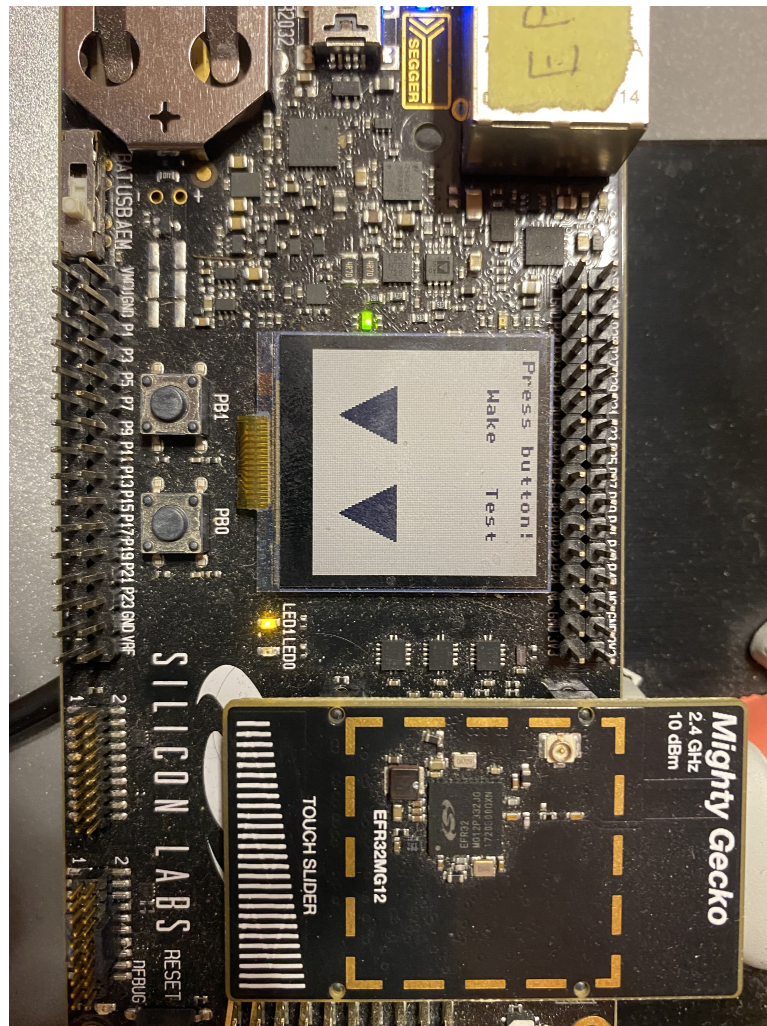


Figure D.4: The main screen of the application.

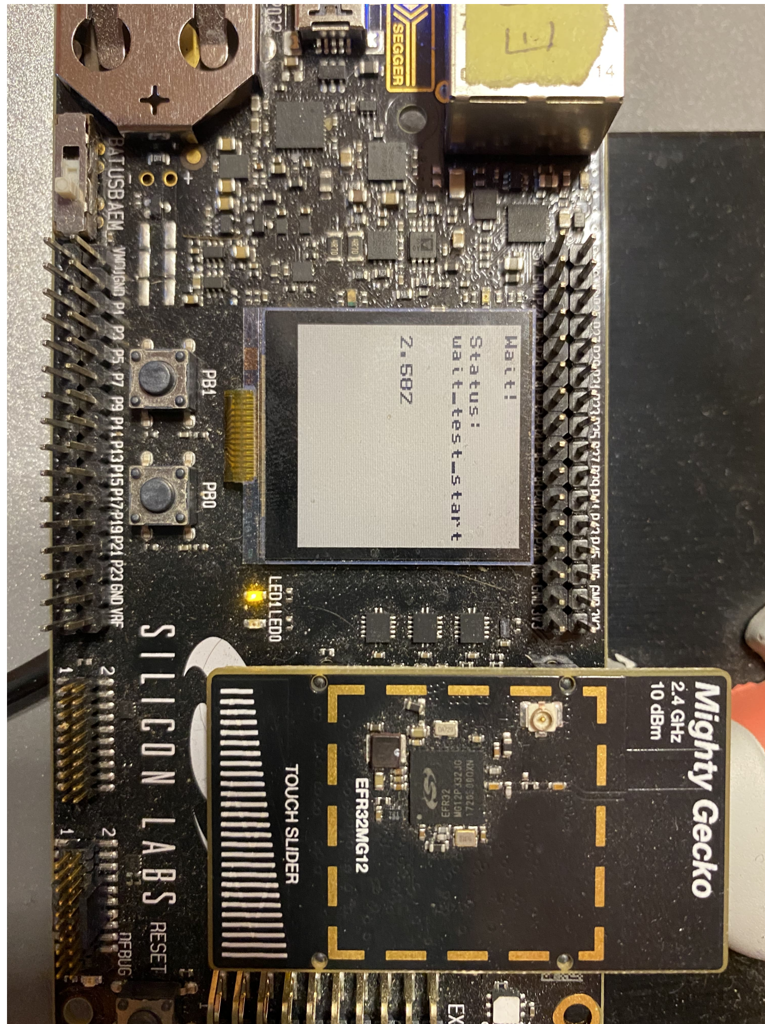


Figure D.5: Screen showing test status.

Bibliography

[1] M. Weiser. The computer for the 21 st century. *Scientific american*, 265(3):94–105, 1991.

[2] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[3] W. Dargie and C. Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley & Sons, 2010.

[4] E. Callaway, P. Gorday, L. Hester, J. A. Gutierrez, M. Naeve, B. Heile, and V. Bahl. Home networking with ieee 802.15.4: a developing standard for low-rate wireless personal area networks. *IEEE Communications Magazine*, 40(8):70–77, 2002.

[5] Bluetooth SIG. The Bluetooth Core Specification motherfucker. `https://www.bluetooth.com/specifications/bluetooth-core-specification/`, 2019. [Online; accessed: 2022-3-3].

[6] S. Schwarz, J. C. Ikuno, M. Šimko, M. Taranetz, Q. Wang, and M. Rupp. Pushing the limits of lte: A survey on research enhancing the standard. *IEEE Access*, 1:51–62, 2013. doi:10.1109/ACCESS.2013.2260371.

[7] B. H. Walke, P. Seidenberg, and M. P. Althoff. *UMTS: the fundamentals*. John Wiley & Sons, 2003.

[8] A. D. Zayas and P. Merino. The 3GPP NB-IoT system architecture for the Internet of Things. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 277–282. 2017. doi:10.1109/ICCW.2017.7962670.

[9] S. Li, L. D. Xu, and S. Zhao. 5g internet of things: A survey. *Journal of Industrial Information Integration*, 10:1–9, 2018. ISSN 2452-414X. doi: https://doi.org/10.1016/j.jii.2018.01.005.

[10] 20 billion wi-fi enabled devices to ship from 2019-2024. `https://www.everythingrf.com/News/details/8441-20-Billion-Wi-Fi-Enabled-Devices-to-Ship-from-2019-2024`, 2019.

[11] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The internet of things has a gateway problem. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, pages 27–32. 2015.

[12] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. U. Gamm, and L. Reindl. Has time come to switch from duty-cycled mac protocols to wake-up radio for wireless sensor networks? *IEEE/ACM Transactions on Networking*, 24(2):674–687, 2016.

[13] Bluetooth SIG. The Bluetooth Core Specification motherfucker. `https://www.bluetooth.com/specifications/bluetooth-core-specification/`, 2019. [Online; accessed: 2022-03-03].

[14] S.-L. Tsao and C.-H. Huang. A survey of energy efficient mac protocols for ieee 802.11 wlan. *Computer Communications*, 34(1):54–67, 2011.

[15] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver. IEEE 802.11 AH: the WiFi approach for M2M communications. *IEEE Wireless Communications*, 21(6):144–152, 2014.

[16] L. Alliance. Lorawan spec 1.0. 2. 2016.

[17] R. Piyare, A. L. Murphy, C. Kiraly, P. Tosato, and D. Brunelli. Ultra low power wake-up radios: A hardware and networking survey. *IEEE Communications Surveys Tutorials*, 19(4):2117–2157, 2017. doi:10.1109/COMST.2017.2728092.

[18] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. U. Gamm, and L. Reindl. Wake-up radio as an energy-efficient alternative to conventional wireless sensor networks MAC protocols. In *Proceedings of the 16th ACM International Conference on Modeling, Analysis & Simulation of Wireless and Mobile Systems*, MSWiM '13, pages 173–180. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-2353-6. doi:10.1145/2507924.2507955.

[19] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[20] S. Kumar and D. Shepherd. Sensit: Sensor information technology for the warfighter. In *Proc. 4th Int. Conf. on Information Fusion*, pages 1–7. 2001.

[21] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 271–278. ACM, New York, NY, USA, 1999.

[22] N. Stephenson. *The Diamond Age*. Bantam Books, New York, 1995.

[23] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800, 2020. doi:10.1109/IEEESTD.2020.9144691.

[24] T. I. of Electrical and E. Engineers. IEEE 802.15 WSN Task Group 4Cor (TG4Cor). `https://www.ieee802.org/15/pub/TG4Cor.html`, 2021. [Online; accessed: 2022-03-03].

[25] Zigbee specification. `https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf`, 2015. [Online; accessed: 2022-03-03].

[26] Thread white paper. `https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview_633_2.pdf`, 2015. [Online; accessed: 2022-03-03].

[27] C. Yibo, K. Hou, H. Zhou, H. Shi, X. Liu, X. Diao, H. Ding, J. Li, and C. de Vaulx. 6lowpan stacks: A survey. In *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4. 2011. doi:10.1109/wicom.2011.6040344.

[28] A. Dunkels. The uip embedded tcp/ip stack. *The uIP*, 1, 2006.

[29] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. 2004.

[30] Z-wave webpage. `http://www.z-wave.com/learn`, 2021. [Online; accessed: 2022-03-03].

[31] Insteon webpage. `https://www.insteon.com/`, 2021. [Online; accessed: 2022-03-03].

[32] S. Aggarwal and A. Nasipuri. Survey and performance study of emerging lpwan technologies for iot applications. In *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT and AI (HONET-ICT)*, pages 069–073. 2019. doi:10.1109/HONET.2019.8908117.

[33] C. Park, K. Lahiri, and A. Raghunathan. Battery discharge characteristics of wireless sensor nodes: An experimental analysis. In *2005 Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005.*, pages 430–440. Citeseer, 2005.

[34] K. S. Adu-Manu, N. Adam, C. Tapparello, H. Ayatollahi, and W. Heinzelman. Energy-harvesting wireless sensor networks (eh-wsns): A review. *ACM Trans. Sen. Netw.*, 14(2), 2018. ISSN 1550-4859.

[35] Silicon labs EFR32MG12 Datasheet. https://www.silabs.com/documents/public/data-sheets/EFR32MG1-SF-DataSheet.pdf, 2018. [Online; accessed 2022-03-03].

[36] Nrf52 datasheet. http://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf, 2018. Online; accessed: 2022-03-03.

[37] Texas instruments cc2650 datasheet. http://www.ti.com/lit/ds/symlink/cc2650.pdf, 2016. [Online; accessed: 2022-03-03].

[38] Silicon labs EFR32MG13 Datasheet. https://www.silabs.com/documents/public/data-sheets/efr32mg13-datasheet.pdf, 2021. [Online; accessed 2022-03-03].

[39] J. M. Gilbert and F. Balouchi. Comparison of energy harvesting systems for wireless sensor networks. *International Journal of automation and computing*, 5(4):334–347, 2008.

[40] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 88–97. ACM, New York, NY, USA, 2002.

[41] E. . A. Lin, J. M. Rabaey, and A. Wolisz. Power-efficient rendez-vous schemes for dense wireless sensor networks. In *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, volume 7, pages 3769–3776 Vol.7. 2004.

[42] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576. IEEE, 2002.

[43] M. Ali, U. Saif, A. Dunkels, T. Voigt, K. Römer, K. Langendoen, J. Polastre, and Z. A. Uzmi. Medium access control issues in sensor networks. *ACM SIGCOMM Computer Communication Review*, 36(2):33–36, 2006.

[44] A. El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 5, pages 3418–3423. IEEE, 2002.

[45] J. L. Hill and D. E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE micro*, 22(6):12–24, 2002.

[46] C. C. Enz, A. El-Hoiydi, J.-D. Decotignie, and V. Peiris. Wisenet: an ultralow-power wireless sensor network solution. *Computer*, 37(8):62–70, 2004.

[47] A. El-Hoiydi and J.-D. Decotignie. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 1, pages 244–251. IEEE, 2004.

[48] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM, 2004.

[49] A. Dunkels. The contikimac radio duty cycling protocol. 2011.

[50] J. M. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan. Picoradios for wireless sensor networks: the next challenge in ultra-low power design. In *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, volume 1, pages 200–201 vol.1. 2002.

[51] L. Gu and J. A. Stankovic. Radio-triggered wake-up for wireless sensor networks. *Real-Time Systems*, 29(2):157–182, 2005.

[52] R. Piyare, A. L. Murphy, C. Kiraly, P. Tosato, and D. Brunelli. Ultra low power wake-up radios: A hardware and networking survey. *IEEE Communications Surveys & Tutorials*, 19(4):2117–2157, 2017.

[53] J. M. Rabaey, M. J. Ammer, J. L. Da Silva, D. Patel, and S. Roundy. Picoradio supports ad hoc ultra-low power wireless networking. *Computer*, 33(7):42–48, 2000.

[54] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[55] T. Issariyakul and E. Hossain. Introduction to network simulator 2 (ns2). In *Introduction to Network Simulator NS2*, pages 21–40. Springer, 2012.

[56] F. Z. Djiroun and D. Djenouri. Mac protocols with wake-up radio for wireless sensor networks: A review. *IEEE Communications Surveys Tutorials*, 19(1):587–618, 2017.

[57] C. Guo, L. C. Zhong, and J. M. Rabaey. Low power distributed mac for ad hoc sensor radio networks. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 5, pages 2944–2948. IEEE, 2001.

[58] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Optimizing sensor networks in the energy-latency-density design space. *IEEE transactions on mobile computing*, (1):70–80, 2002.

[59] X. Yang and N. H. Vaidya. A wakeup scheme for sensor networks: achieving balance between energy saving and end-to-end delay. In *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, pages 19–26. 2004.

[60] M. J. Miller and N. H. Vaidya. A mac protocol to reduce sensor network energy consumption using a wakeup radio. *IEEE Transactions on Mobile Computing*, 4(3):228–242, 2005.

[61] S. Mahlknecht and M. S. Durante. Wur-mac: energy efficient wakeup receiver based mac protocol. *IFAC Proceedings Volumes*, 42(3):79–83, 2009.

[62] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, 2012.

[63] X. Fafoutis and N. Dragoni. Odmac: an on-demand mac protocol for energy harvesting-wireless sensor networks. In *Proceedings of the 8th ACM Symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 49–56. ACM, 2011.

[64] T. N. Le, M. Magno, A. Pegatoquet, O. Berder, O. Sentieys, and E. Popovici. Ultra low power asynchronous mac protocol using wake-up radio for energy neutral wsn. In *Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems*, page 10. ACM, 2013.

[65] S. Marinkovic and E. Popovici. Ultra low power signal oriented approach for wireless health monitoring. *Sensors*, 12(6):7917–7937, 2012.

[66] T. N. Le, A. Pegatoquet, and M. Magno. Asynchronous on demand mac protocol using wake-up radio in wireless body area network. In *Advances in Sensors and Interfaces (IWASI), 2015 6th IEEE International Workshop on*, pages 228–233. IEEE, 2015.

[67] F. Dressler, S. Ripperger, M. Hierold, T. Nowak, C. Eibel, B. Cassens, F. Mayer, K. Meyer-Wegener, and A. Kolpin. From radio telemetry to ultra-low-power sensor networks: tracking bats in the wild. *IEEE Communications Magazine*, 54(1):129–135, 2016.

[68] IEEE Draft Standard for Information Technology–Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Wake-Up Radio Operation. *IEEE P802.11ba/D7.0, September 2020k*, pages 1–189, 2020.

[69] I. Demirkol, C. Ersoy, and E. Onur. Wake-up receivers for wireless sensor networks: benefits and challenges. *IEEE Wireless Communications*, 16(4), 2009.

[70] L. Gu, J. A. Stankovic, et al. Radio-triggered wake-up capability for sensor networks. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 27–37. Citeseer, 2004.

[71] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. A. Paradiso. Cargonet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 145–159. ACM, 2007.

[72] J. Ansari, D. Pankin, and P. Mähönen. Radio-triggered wake-ups with addressing capabilities for extremely low power sensor network applications. *International Journal of Wireless Information Networks*, 16(3):118, 2009.

[73] E. Lopez-Aguilera, M. Hussein, M. Cervia, J. Paradells, and A. Calveras. Design and implementation of a wake-up radio receiver for fast 250 kb/s bit rate. *IEEE Wireless Communications Letters*, 8(6):1537–1540, 2019.

[74] N. Pletcher, S. Gambini, and J. Rabaey. A 65 μw, 1.9 ghz rf to digital baseband wakeup receiver for wireless sensor nodes. In *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE*, pages 539–542. Citeseer, 2007.

[75] R. G. Meyer. Low-power monolithic rf peak detector analysis. *IEEE Journal of Solid-State Circuits*, 30(1):65–67, 1995.

[76] N. M. Pletcher, S. Gambini, and J. Rabaey. A 52 μW wake-up receiver with -72 dbm sensitivity using an uncertain-if architecture. *IEEE Journal of solid-state circuits*, 44(1):269–280, 2009.

[77] P.-H. P. Wang, H. Jiang, L. Gao, P. Sen, Y.-H. Kim, G. M. Rebeiz, P. P. Mercier, and D. A. Hall. A near-zero-power wake-up receiver achieving- 69-dbm sensitivity. *IEEE Journal of Solid-State Circuits*, 53(6):1640–1652, 2018.

[78] C. Salazar, A. Kaiser, A. Cathelin, and J. Rabaey. 13.5 a- 97dbm-sensitivity interferer-resilient 2.4 ghz wake-up receiver using dual-if multi-n-path architecture in 65nm cmos. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[79] Eclipse IoT developer survey. `https://outreach.eclipse.foundation/download-the-eclipse-iot-developer-survey-results`, 2018. [Online; accessed: 2022-03-03].

[80] Texas instruments cc3220 datasheet. `http://www.ti.com/lit/ds/symlink/cc3220.pdf`, 2021. [Online; accessed: 2022-03-03].

[81] IEEE 802.11 TGba. Proposal for Wake- Up receiver (WUR) study group. https://mentor.ieee.org/802.11/dcn/16/11-16-0722-01-0000-proposal-for-wake-up-receiver-study-group.pptx, 2016. [Online; accessed: 2022-03-03].

[82] S. Yin, Q. Li, and O. Gnawali. Interconnecting WiFi Devices with IEEE 802.15.4 Devices without Using a Gateway. In *2015 International Conference on Distributed Computing in Sensor Systems*, pages 127–136. 2015. doi:10.1109/DCOSS.2015.42.

[83] H. Zhang, C. Li, S. Chen, X. Tan, N. Yan, and H. Min. A low-power OFDM-based wake-up mechanism for Io E applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(2):181–185, 2018.

[84] IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements - part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999) - Redline*, pages 1–1238, 2007.

[85] M. Lopez, D. Sundman, and L. Wilhelmsson. MC-OOK symbol design. https://mentor.ieee.org/802.11/dcn/18/11-18-0479-02-00ba-mc-ook-symbol-design.pptx, 2018. [Online; accessed: 2022-03-03].

[86] V. Kristem, S. Azizi, and T. Kenney. MC-OOK symbol design. https://mentor.ieee.org/802.11/dcn/18/11-18-0492-02-00ba-2us-ook-waveform-generation.pptx, 2018. [Online; accessed: 2022-03-03].

[87] The MathWorks, Inc. Reference web-page for MATLAB WLAN toolbox. https://es.mathworks.com/help/wlan/, 2021. [Online; accessed: 2022-03-03].

[88] S. A. Talwalkar and S. L. Marple. Time-frequency scaling property of discrete fourier transform (DFT). In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3658–3661. 2010. ISSN 1520-6149. doi:10.1109/ICASSP.2010.5495902.

[89] B. Bloessl, C. Sommer, F. Dressier, and D. Eckhoff. The scrambler attack: A robust physical layer attack on location privacy in vehicular networks. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 395–400. IEEE, 2015.

[90] IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC)and physical layer (PHY) specifications amendment 5: Enhancements for higher throughput. *IEEE Std 802.11n-2009 (Amendment to IEEE*

Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009), pages 1–565, 2009. doi:10.1109/IEEESTD.2009.5307322.

[91] IEEE standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications–amendment 4: Enhancements for very high throughput for operation in bands below 6 ghz. *IEEE Std 802.11ac(TM)-2013 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012, IEEE Std 802.11aa-2012, and IEEE Std 802.11ad-2012)*, pages 1–425, 2013. doi:10.1109/IEEESTD.2013.7797535.

[92] Q. Qu, B. Li, M. Yang, Z. Yan, A. Yang, D.-J. Deng, and K.-C. Chen. Survey and performance evaluation of the upcoming next generation WLANs standard-IEEE 802.11 ax. *Mobile Networks and Applications*, 24(5):1461–1474, 2019.

[93] A. Yadav, D. Mazumdar, B. Karthikeyan, and G. R. Kadambi. Linearization of saleh, ghorbani and rapp amplifiers with doherty technique. *SASTech Journal*, 9:79–86, 2010.

[94] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, and F. Piessens. Why mac address randomization is not enough: An analysis of wi-fi network discovery mechanisms. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*, pages 413–424. 2016.

[95] T. L. K. D. Community. Linux 802.11 driver developer's guide. `https://www.kernel.org/doc/html/latest/driver-api/80211/index.html`, 2021. [Online; accessed: 2022-03-03].

[96] R. Krishnan, R. G. Babu, S. Kaviya, N. P. Kumar, C. Rahul, and S. S. Raman. Software defined radio (sdr) foundations, technology tradeoffs: A survey. In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, pages 2677–2682. 2017. doi: 10.1109/ICPCSI.2017.8392204.

[97] B. Bloessl, M. Segata, C. Sommer, and F. Dressler. An ieee 802.11 a/g/p ofdm receiver for gnu radio. In *Proceedings of the second workshop on Software radio implementation forum*, pages 9–16. 2013.

[98] The gnu software radio. `https://gnuradio.org`, 2007. [Online; accessed: 2022-03-03].

[99] B. Bloessl. gr-ieee802.11, an IEEE 802.11 a/g/p transceiver for GNU Radio. `https://github.com/bastibl/gr-ieee802-11`, 2021.

[100] The MicroPython project. `https://micropython.org/download/`, 2021.

[101] Espressif. The Espressif Internet Development Framework (ESP-IDF). https://github.com/espressif/esp-idf/releases/tag/v3.2.4, 2021.

[102] Clarivate Analytics. 2019 Journal Impact Factor. 2020.

[103] M. C. Caballé, A. C. Augé, E. Lopez-Aguilera, E. Garcia-Villegas, I. Demirkol, and J. P. Aspas. An alternative to IEEE 802.11ba: Wake-up radio with legacy IEEE 802.11 transmitters. *IEEE Access*, 7:48068–48086, 2019. ISSN 2169-3536. doi:10.1109/ACCESS.2019.2909847.

[104] M. C. Caballé, A. C. Augé, E. Lopez-Aguilera, E. Garcia-Villegas, I. Demirkol, and J. P. Aspas. A method and a device to generate an amplitude-based modulation wireless signal using ofdm to be received by a low-power non-coherent receiver. https://patentscope.wipo.int/search/en/detail.jsf?docId=EP325120336&tab=NATIONALBIBLIO&_cid=P12-KUSH3Z-60908-1, 2018. [Online; accessed: 2022-03-03].

[105] E. Alpman, A. Khairi, R. Dorrance, M. Park, V. S. Somayazulu, J. R. Foerster, A. Ravi, J. Paramesh, and S. Pellerano. 802.11g/n compliant fully integrated wake-up receiver with -72-dbm sensitivity in 14-nm finfet cmos. *IEEE Journal of Solid-State Circuits*, 53(5):1411–1422, 2018.

[106] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. U. Gamm, and L. Reindl. Has time come to switch from duty-cycled mac protocols to wake-up radio for wireless sensor networks? *IEEE/ACM Transactions on Networking*, 24(2):674–687, 2015.

[107] M. Hussein. Low power iot based on wifi. 2019. Master's Thesis, Universitat Politècnica de Catalunya, ETSETB.

[108] D. Vila. Design and implementation of a radio wake up receiver. 2018. Bachelor's Thesis, Universitat Politècnica de Catalunya, ETSETB.

[109] Low-power rf detector. https://www.analog.com/media/en/technical-documentation/data-sheets/5508fa.pdf, 2002.

[110] Operational amplifier. https://www.renesas.com/eu/en/www/doc/datasheet/ca3140-a.pdf, 2005. Rev.10.00.

[111] Ultra-low power microcontroller. https://www.microchip.com/stellent/groups/picmicro_sg/documents/devicedoc/cn547043.pdf, 2016. Rev.G.

[112] IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016.

[113] K. Chebrolu and A. Dhekne. Esense: Communication through energy sensing. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 85–96. 2009.

[114] W. Jiang, Z. Yin, S. M. Kim, and T. He. Transparent cross-technology communication over data traffic. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[115] S. M. Kim and T. He. Freebee: Cross-technology communication via free side-channel. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 317–330. 2015.

[116] Z. Chi, Y. Li, H. Sun, Y. Yao, and T. Zhu. Concurrent cross-technology communication among heterogeneous iot devices. *IEEE/ACM Transactions on Networking*, 27(3):932–947, 2019.

[117] Z. Li and T. He. Webee: Physical-layer cross-technology communication via emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 2–14. 2017.

[118] Z. Li and T. He. Longbee: Enabling long-range cross-technology communication. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 162–170. 2018.

[119] Y. Chen, Z. Li, and T. He. Twinbee: Reliable physical-layer cross-technology communication with symbol-level coding. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 153–161. IEEE, 2018.

[120] W. Jiang, Z. Yin, R. Liu, Z. Li, S. M. Kim, and T. He. Bluebee: a 10,000 x faster cross-technology communication via phy emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–13. 2017.

[121] Z. Li and Y. Chen. Achieving universal low-power wide-area networks on existing wireless devices. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.

[122] S. Wang, Z. Yin, Y. Chen, Z. Li, S. M. Kim, and T. He. Networking support for bidirectional cross-technology communication. *IEEE Transactions on Mobile Computing*, 2019.

[123] C. Gomez, J. Paradells, C. Bormann, and J. Crowcroft. From 6lowpan to 6lo: Expanding the universe of ipv6-supported technologies for the internet of things. *IEEE Communications Magazine*, 55(12):148–155, 2017.

[124] R. Piyare, A. L. Murphy, C. Kiraly, P. Tosato, and D. Brunelli. Ultra low power wake-up radios: A hardware and networking survey. *IEEE Communications Surveys Tutorials*, 19(4):2117–2157, 2017. doi:10.1109/COMST.2017.2728092.

[125] M. C. Caballé, A. C. Augé, E. Lopez-Aguilera, E. Garcia-Villegas, I. Demirkol, and J. P. Aspas. An alternative to ieee 802.11ba: Wake-up radio with legacy ieee 802.11 transmitters. *IEEE Access*, 7:48068–48086, 2019.

[126] AN971: EFR32 Radio Configurator Guide 0.8. `https://www.silabs.com/documents/public/application-notes/an971-efr32-radio-configurator-guide.pdf`, 2018. [Online; accessed 2022-03-03].

[127] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Further Higher Data Rate Extension in the 2.4 GHz Band. *IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11, 1999 Edn. (Reaff 2003) as amended by IEEE Stds 802.11a-1999, 802.11b-1999, 802.11b-1999/Cor 1-2001, and 802.11d-2001)*, pages 1–104, 2003.

[128] T. S. Baicheva. Determination of the best crc codes with up to 10-bit redundancy. *IEEE transactions on communications*, 56(8):1214–1220, 2008.

[129] T. Group. Thread White Paper. `https://www.threadgroup.org/Portals/0/documents/support/ThreadOverview_633_2.pdf`, 2015. [Online; accessed 2022-03-03].

[130] J. Im, J. Breiholz, S. Li, B. Calhoun, and D. D. Wenzloff. A fully integrated 0.2v 802.11ba wake-up receiver with -91.5dbm sensitivity. In *2020 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*, pages 339–342. 2020.

[131] D. Deng, S. Lien, C. Lin, M. Gan, and H. Chen. IEEE 802.11ba Wake-Up Radio: Performance Evaluation and Practical Designs. *IEEE Access*, 8:141547–141557, 2020. ISSN 2169-3536. doi:10.1109/ACCESS.2020.3013023.

[132] M. Cervià-Caballé. WuR-CTC WuRx Firmware for the STM32L0R8. `https://github.com/marticervia/WuR-CTC-WuRx`, 2020.

[133] M. Cervià-Caballé. Multiplatform WuR-CTC Implementation. `https://github.com/marticervia/WuR-CTC-Library`, 2020.

[134] M. Cervià-Caballé. WuR-CTC Demo Application for the ESP-32. `https://github.com/marticervia/WuR-CTC-Demo-ESP32`, 2020.

[135] M. Cervià-Caballé. WuR-CTC Demo Application for the EFR32MG12. `https://github.com/marticervia/WuR-CTC-Demo-EFR32MG12`, 2020.

[136] C. Phillips and S. Singh. CRAWDAD dataset pdx/vwave (v. 2009-07-04). Downloaded from `https://crawdad.org/pdx/vwave/20090704`, 2009.

[137] M. C. Caballé, A. C. Augé, and J. P. Aspas. Wake-up radio: An enabler of wireless convergence. *IEEE Access*, 9:3784–3797, 2021. doi:10.1109/ACCESS.2020.3048673.

[138] P. Masek, M. Stusek, K. Zeman, R. Drapela, A. Ometov, and J. Hosek. Implementation of 3gpp lte cat-m1 technology in ns-3: System simulation and performance. In *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 1–7. 2019. doi:10.1109/ICUMT48472.2019.8970869.

[139] 3GPP. Release 16 Description; Summary of Rel-16 Work Items. Technical Report (TR) 21.916, 3rd Generation Partnership Project (3GPP), 2021. Version 16.0.1.