Universidade do Minho
Escola de Engenharia

Tiago Manuel Ribeiro Gomes

A Sensor Node SoC Architecture for Extremely Autonomous Wireless Sensor Networks

setembro de 2017

Tiago Manuel Ribeiro Gomes A Sensor Node SoC Architecture for Extremely Autonomous Wireless Sensor Networks

UMinho | 2017

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR
POPH QUALIFICAR É CRESCER.
QREN QUADRO DE REFERÊNCIA ESTRATÉGICO NACIONAL
GOVERNO DA REPÚBLICA PORTUGUESA
UNIÃO EUROPEIA Fundo Social Europeu

Universidade do Minho
Escola de Engenharia

Tiago Manuel Ribeiro Gomes

A Sensor Node SoC Architecture for Extremely Autonomous Wireless Sensor Networks

Tese de Doutoramento
Programa Doutoral em
Engenharia Eletrónica e de Computadores (PDEEC)
Área de Especialização:
Informática Industrial e Sistemas Embebidos

Trabalho efetuado sob a orientação do
Professor Doutor Jorge Miguel Nunes dos Santos Cabral
Professor Doutor Pedro Miguel Mestre Alves da Silva

setembro de 2017

DECLARAÇÃO

Nome: Tiago Manuel Ribeiro Gomes

Correio electrónico: gomes.tmr@gmail.com

Tel./Tlm.: 963522404

Número do Bilhete de Identidade:12487593

Título da dissertação: **A Sensor Node SoC Architecture for Extremely Autonomous Wireless Sensor Networks**

Ano de conclusão: 2017

Orientadores:

Doutor Jorge Miguel Nunes dos Santos Cabral

Doutor Pedro Miguel Mestre Alves da Silva

Designação do Doutoramento: Programa Doutoral em Engenharia Eletrónica e de Computadores (PDEEC)

Área de Especialização: Informática Industrial e Sistemas Embebidos

Escola: Escola de Engenharia

Departamento: Departamento de Electrónica Industrial

Assinatura: [illegible]

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, ______________________________

Full name: Tiago Manuel Ribeiro Gomes

Signature: [illegible]

Acknowledgments

I wish foremost to express my sincere gratitude to Dr. Adriano Tavares for the continuous support, for his patience and ceaseless motivation. His priceless guidance always turned big problems into non-problems. Thank you for being such a stimulating person.

My sincere thanks to the best advisor and friend, Dr. Jorge Cabral, for his insightful comments and valuable discussions, which incited me to widen my research from different perspectives and smoothed my PhD journey. Thank you for all the encouragement and continuous support.

I am also grateful to Dr. Mongkol Ekpanyapong at the Asian Institute of Technology, Dr. Sergio Montenegro at the University of Würzburg and Dr. Hao Xu at the Jilin University, for hosting me in their labs during my visits to their institutions.

A special thank to my fellow labmates for the stimulating discussions and for the sleepless nights we were working together in our endless "night shifts". You surely know about all that we have been through. Special mention to Dr. Sandro Pinto, Dr. Tiago Gomes and (soon to be Dr.) Filipe Salgado, for whom I express my deepest gratitude and eternal friendship. Thank you for all the great moments we have spent abroad in working and traveling together, and thank you for the grateful discussions and important enlightenments on those hard times of this journey. Seriously guys: Thanks for all the fun!

Last but not the least, I would like to thank some of the most important persons in my life. To my mom, dad and sisters, thank you for supporting me spiritually through such a long five years. Special thanks to Nuno Araújo for his friendship and tremendous support. To Cláudia, thank you for your eternal support and endless patience during these last two years. You have been wonderful.

This thesis was supported by a PhD scholarship from *Fundação para a Ciência e Tecnologia*, SFRH/BD/90162/2012.

Tiago Gomes

Guimarães, September 2017.

"It always seems impossible until it's done"

— Nelson Mandela

Abstract

The Internet of Things (IoT) is revolutionizing the Internet of the future and the way new smart objects and people are being connected into the world. Its pervasive computing and communication technologies connect myriads of smart devices, presented at our everyday things and surrounding objects. Big players in the industry forecast, by 2020, around 50 billion of smart devices connected in a multitude of scenarios and heterogeneous applications, sharing data over a true worldwide network. This will represent a trillion dollar market that everyone wants to take a share.

In a world where everything is being connected, device security and device interoperability are a paramount. From the sensor to the cloud, this triggers several technological issues towards connectivity, interoperability and security requirements on IoT devices. However, fulfilling such requirements is not straightforward. While the connectivity exposes the device to the Internet, which also raises several security issues, deploying a standardized communication stack on the endpoint device in the network edge, highly increases the data exchanged over the network. Moreover, handling such ever-growing amount of data on resource-constrained devices, truly affects the performance and the energy consumption. Addressing such issues requires new technological and architectural approaches to help find solutions to leverage an accelerated, secure and energy-aware IoT end-device communication.

Throughout this thesis, the developed artifacts triggered the achievement of important findings that demonstrate: (1) how heterogeneous architectures are nowadays a perfect solution to deploy endpoint devices in scenarios where not only (heavy processing) application-specific operations are required, but also network-related capabilities are major concerns; (2) how accelerating network-related tasks result in a more efficient device resources utilization, which combining better performance and increased availability, contributed to an improved overall energy utilization; (3) how device and data security can benefit from modern heterogeneous architectures that rely on secure hardware platforms, which are also able to provide security-related acceleration hardware; (4) how a domain-specific language eases the co-design and customization of a secure and accelerated IoT endpoint device at the network edge.

Resumo

Internet of Things (IoT) é o conceito que está a revolucionar a Internet do futuro e a forma como coisas, processos e pessoas se conectam e se relacionam numa infraestrutura de rede global que interligará, num futuro próximo, um vasto número de dispositivos inteligentes e de utilização diária. Com uma grande aposta no mercado IoT por parte dos grandes líderes na industria, algumas visões otimistas preveem para 2020 mais de 50 mil milhões de dispositivos ligados na periferia da rede, partilhando grandes volumes de dados importantes através da Internet, representando um mercado multimilionário com imensas oportunidades de negócio.

Num mundo interligado de dispositivos, a interoperabilidade e a segurança é uma preocupação crescente. Tal preocupação exige inúmeros esforços na exploração de novas soluções, quer a nível tecnológico quer a nível arquitetural, que visem impulsionar o desenvolvimento de dispositivos embebidos com maiores capacidades de desempenho, segurança e eficiência energética, não só apenas do dispositivo em si, mas também das camadas e protocolos de rede associados. Apesar da integração de pilhas de comunicação e de protocolos *standard* das camadas de rede solucionar problemas associados à conectividade e a interoperabilidade, adiciona a sobrecarga inerente dos protocolos de comunicação e do crescente volume de dados partilhados entre os dispositivos e a Internet, afetando severamente o desempenho e a disponibilidade do mesmo, refletindo-se num maior consumo energético global.

As soluções apresentadas nesta tese permitiram obter resultados que demonstram: (1) a viabilidade de soluções heterogéneas no desenvolvimento de dispositivos IoT, onde não só tarefas inerentes à aplicação podem ser aceleradas, mas também tarefas relacionadas com a comunicação do dispositivo; (2) os benefícios da aceleração de tarefas e protocolos da pilha de rede, que se traduz num melhor desempenho do dispositivo e aumento da disponibilidade do mesmo, contribuindo para uma melhor eficiência energética; (3) que plataformas de *hardware* modernas oferecem mecanismos de segurança que podem ser utilizados não apenas em prol da segurança do dispositivo, mas também nas capacidades de comunicação do mesmo; (4) que o desenvolvimento de uma linguagem de domínio específico permite de forma mais eficaz e eficiente o desenvolvimento e configuração de dispositivos IoT inteligentes.

Contents

List of Figures

List of Tables

Listings

Acronyms

6LA 6LoWPAN accelerator

6LoWPAN IPv6 over Low power Wireless Personal Area Networks

6P 6top Protocol

ACK acknowledgment

AES Advanced Encryption Standard

AMBA Advanced Microcontroller Bus Architecture

APB Advanced Peripheral Bus

API application programming interface

ARQ Automatic Repeat reQuest

ASH Auxiliary Security Header

ASIC application-specific integrated circuit

BLE Bluetooth Low Energy

BR border router

CDMA code division multiple access

CID Context IDentifier

CoAP Constrained Application Protocol

COTS commercial off-the-shelf

CPU central processing unit

CRC	cyclic redundancy check
CSMA	carrier-sense multiple access
CSMA/CA	carrier-sense multiple access with collision avoidance
CT	Context Table
CWT	continuous wavelet transform
DAC	Destination Address Compression
DAM	Destination Address Mode
DFD	Duplicate Frame Detector
DFF	D Flip-Flop
DoS	denial-of-service
DPA	Differential Power Analysis
DPM	dynamic power management
DSL	domain-specific language
DSP	Digital Signal Processsor
DTLS	Datagram Transport Layer Security
DVFS	dynamic voltage and frequency scaling
DVS	dynamic voltage scaling
DWT	discrete wavelet transform
ECC	elliptic curve cryptography
EL	Elaboration Language
EMT	Embedded Trace Macrocell
eNVM	embedded Non-Volatile Memory
ERCC	error correction codes
FCF	Frame Control Field
FCS	frame check sequence

FEC	forward error coding
FFT	fast Fourier transform
FIR	finite impulse response
FLPC	Physical Layer Power Conservation
FPGA	field-programmable gate array
GP	generative programming
GPIO	general-purpose input/output
HAL	hardware abstraction layer
HDL	hardware description language
HLIM	Hop LIMit
HTTP	Hypertext Transfer Protocol
I/O	input/output
ICMP	Internet Control Message Protocol
IDE	integrated development environment
IDS	intrusion detection system
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IoE	Internet of Everything
IoT	Internet of Things
IoT-ARM	Internet of Things - Architectural Reference Model
IPC	inter-process communication
IPHC	Internet Protocol Header Compression
IPsec	Internet Protocol Security
LE	logic elements
LLN	Low power and Lossy Networks

LPP	Low-Power Probing
LPDDR	low-power double data rate
LQE	linear-quadratic estimator
LQI	Link Quality Indication
LR-WPAN	low-rate wireless personal area networks
LSRAM	large static random-access memory
LUT	Look-Up Tables
M2M	machine-to-machine
MAC	Medium Access Control
MACA	Multiple Access with Collision Avoidance
MCU	microcontroller
MD5	MD5 Message-Digest Algorithm
MDD	model driven development
mDNS	multicast Domain Name System
MFR	MAC Footer
MHR	MAC Header
MLA	MAC layer accelerator
MPDU	Mac Protocol Data Unit
MQTT	Message Queuing Telemetry Transport
MSS	microcontroller subsystem
MTU	Maximum Transmission Unit
NH	Next Header
OS	operating system
OSes	operating systems
PAN	personal area network

PDR	packet discard rate
PHY	physical layer
PSR	packet sending rate
PUF	Physically Unclonable Function
RA	reference architecture
RAM	random-access memory
RCU	reconfigurable computing unit
RDC	radio duty cycling
RMS	root mean square
ROM	read-only memory
RPL	Routing Protocol for Low power and Lossy Networks
RSSI	Received Signal Strength Indicator
RT	real-time
RTL	register-transfer level
RTOS	real-time operating system
SAC	Source Address Compression
SAM	Source Address Mode
SCA	Service-Component Architecture
SDP	sensor data processing
SDR	software-defined radio
SDRAM	synchronous dynamic random-access memory
SerDes	Serializer/Deserializer
SFD	start frame delimiter
SHA-1	Secure Hash Algorithm 1
SN	Sequence Number

SNMP Simple Network Management Protocol

SOA service-oriented architecture

SOAP Simple Object Access Protocol

SoC system-on-chip

SPI Serial Peripheral Interface

SRAM static random-access memory

TCP Transmission Control Protocol

TDMA time division multiple access

TF Traffic Class & Flow Level

TSCH Time Synchronized Channel Hopping

TSMP Time Synchronized Mesh Protocol

TLS Transport Layer Security

TTA transport triggered architecture

UDP User Datagram Protocol

VHDL VHSIC Hardware Description Language

WPAN wireless personal area network

WSN wireless sensor networks

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

Chapter 1

Introduction

The ubiquitous connectivity of endpoint devices in the Internet of Things (IoT) brings new challenges to the traditional wireless sensor networks, whose network architectures are mainly centralized and most of the time disconnected from the Internet. Such challenges not only comprise the connectivity and its subsequent interoperability of heterogeneous wireless nodes, as specified by several architecture reference models, but also encompass security and privacy-related features, along with robust solutions to handle the ever-growing amounts of data transferred over the network. However, tackling such challenges on resource-constrained devices is not straightforward. The need for energy-efficient wireless devices, while preserving their performance and security capabilities, requires for new architectural solutions at the hardware mote.

This chapter presents the overall content of this thesis. It is prefaced by an introduction and then succeeded by the following sections: Section 1.2 describes the problem statement and the main scope of this work, while Section 1.3 raises some research questions and the proposed methodology to address them. Section 1.4 presents the state-of-the-art for endpoint devices on the network edge, along with the associated challenges and the envisioned solution to tackle the problems identified in Section 1.2. Section 1.5 summarizes the contributions of this thesis and Section 1.6 presents the adopted thesis structure. Finally, Section 1.7 concludes this chapter.

1.1 Introduction

In the past few years, wireless sensor networks (WSN) have become extremely popular on a wide range of domains, such as critical monitoring systems, security, habitat monitoring or industrial applications [1–3]. Traditionally, the low-power nodes form a network where a large number of devices communicate with each other, collecting data, processing and sending them to a centralized controlling application. The overriding need to send data over the Internet to online servers has considerably increased, triggering the paradigm shift from the centralized and isolated WSN to the new Internet era, the Internet of Things (IoT) [4–6]. The IoT pervasive technology materializes the concept of a worldwide infrastructure which enables people, processes and devices to interact and interconnect at any time, from anywhere.

The IoT is growing at a breathtaking pace. According to the U.S. National Intelligence Council (NIC), "By 2025 Internet nodes may reside in everyday things, food packages, furniture, paper documents, and more" [7], and will start to play a key role in our surrounding infrastructures and daily routines. Figure 1.1 depicts the heap cycle for the emerging technologies in 2015, according to Gartner [8]. The IoT is at the peak of the inflated expectations and it is expected to reach the plateau of productivity in less than ten years. In the near future, a countless number of

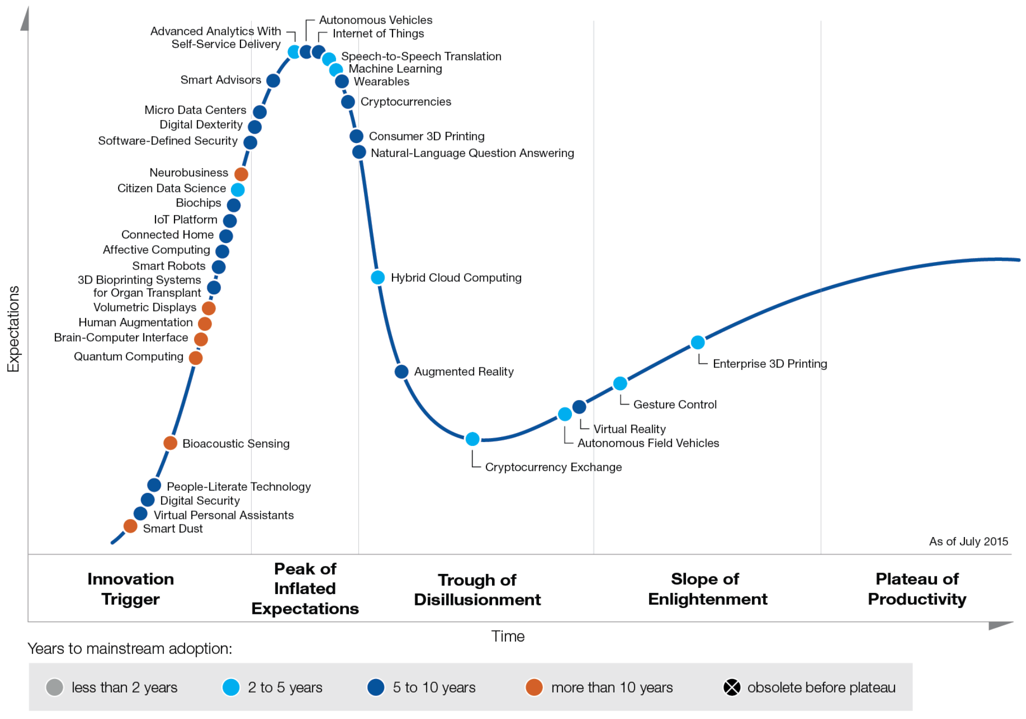


Figure 1.1: Gartner 2015 hype cycle for IoT emerging technologies [8].

heterogeneous smart objects, e.g., sensors, personal and wearable devices, within a wide range of applications will have to coexist, sharing large amounts of valuable data over the Internet [9]. The rapid growth of the IoT concept, where every device can seamlessly connect to the Internet, generated significant interest from big players in the semiconductor and network industries, such as ARM, Intel, Cisco, IBM, Microsoft, among others, pushing them into creating solutions, from the silicon to the cloud, that aim to satisfy a broad range of requirements on a multitude of applications and scenarios.

ARM predicted in 2015 that, by 2020, there will be around 26 billion (26,000 million) of smart devices installed and connected to the cloud, sharing valuable data securely through the Internet. By its turn, Cisco expects around 50 billion devices, in a wider network designated by the Internet of Everything (IoE). The IoE is described as a huge network where people, processes, data and things co-exist in this globally connected world [10]. In a more optimist prediction, Intel expected the IoT to be a multi-trillion dollar market with an install base of 15 billion connected things in 2015, being this number increased to 200 billion by the end of 2020, which will represent an annual growth higher than 20% [11]. Estimated to have been born between 2008 and 2009, the IoT is well under way. As detailed in Figure 1.2, Cisco forecasts by 2020 around 50 billion connected devices. With an expected world population of 7.5 billion, this gives around six smart objects for every human being on Earth [12]. Those distinct predictions have one common thing: myriads of smart connected devices are expected to surround us in the very near future, which will represent a huge revenue opportunity for several market investments.

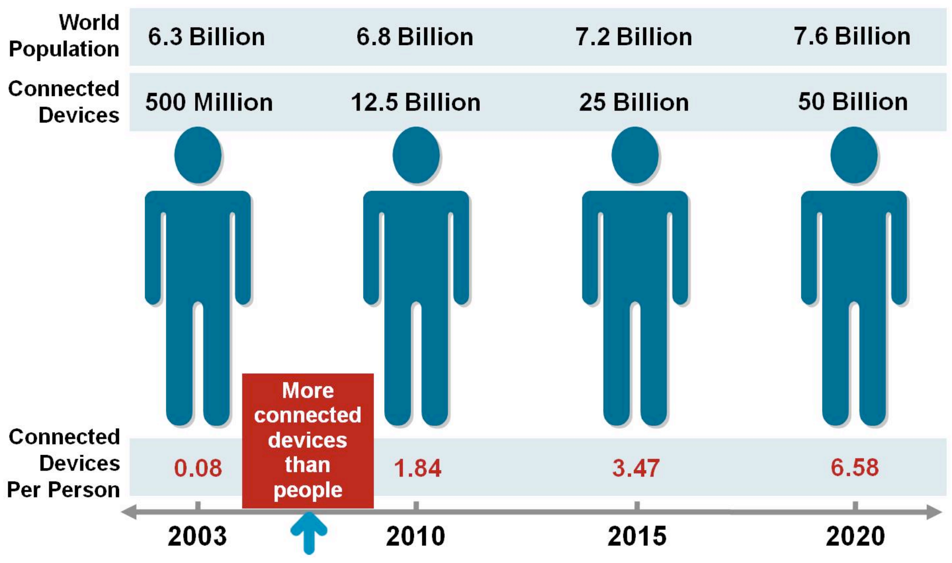


Figure 1.2: The IoT vs. world population (Cisco forecasts) [12].

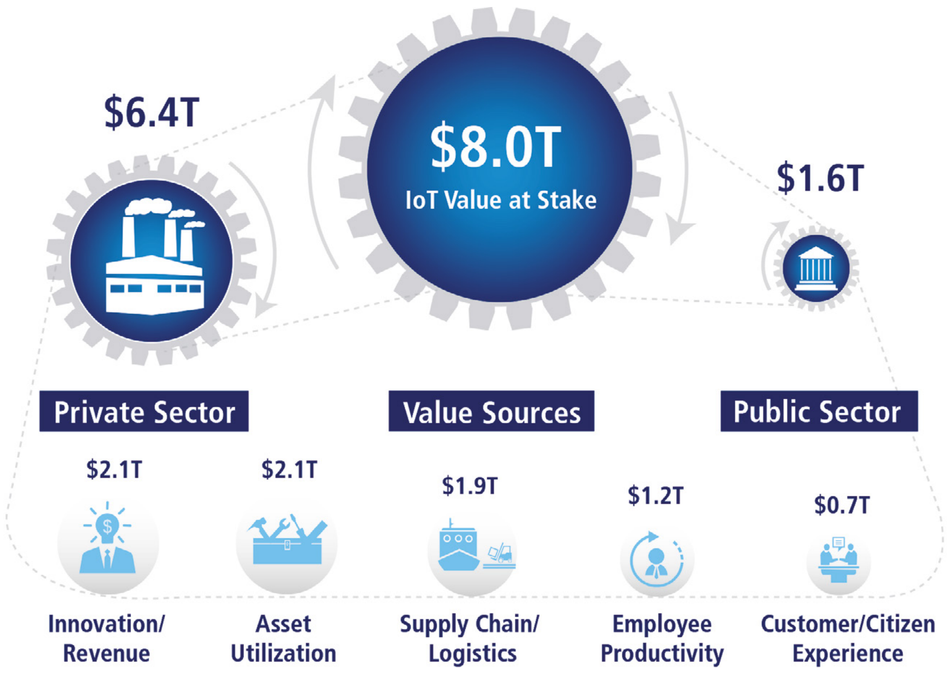


Figure 1.3: IoT Value at Stake [13].

A collaborative report by DHL and Cisco on implications and use cases for the logistics industry [13], as illustrated in Figure 1.3, predicts that IoT will generate $8 trillion worldwide in Value at Stake over the next decade. This value will come from five primary drivers: innovation and revenue; asset utilization; supply chain and logistics; employee productivity improvements; and enhanced customer and citizen experience. Supply chain and logistics alone are estimated to provide $1.9 trillion in value, which is a promising indication of the untapped potential and profits to gain from utilizing IoT in the logistics industry. The Value at Stake calculations stem from a bottom-up economic analysis conducted by Cisco on several IoT use cases, both public and private sectors. Each use case represents a business capability and resulting economic value brought about by connecting the unconnected. Forward-looking forecasts by analysts at McKinsey institute, expect a potential impact of $3.9 trillion up to $11.1 trillion per year in 2025 [14], in a cross-sector view between nine big sector markets: Vehicles, Home, Offices, Factories, Retail environments, Worksites, Human, Outside and Cities. It becomes clear that this flourish of interconnected devices promise to drive a plethora of applications with technological, economic, and social prospects.

Based on the target context and application scenario, every IoT deployment may adopt different processing and network communication architectures, standard technologies and design approaches. Present-day IoT scenarios already include smart cities [15, 16], intelligent buildings [17], robust home monitoring systems [18], industrial smart-controlled environments [19], health care systems [20, 21], intelligent electrical grid systems [22, 23], smart vehicles [24] and many more.

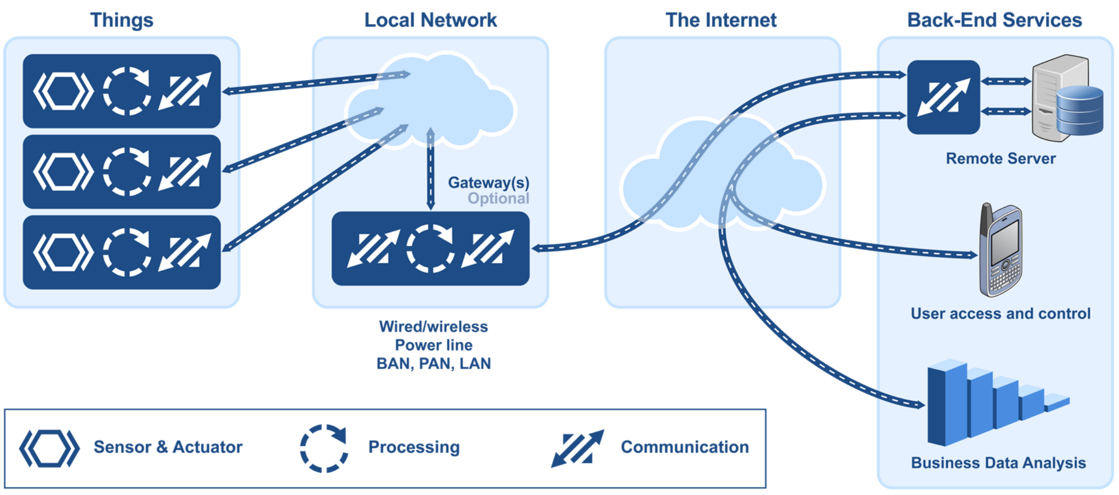


Figure 1.4: The IoT ecosystem from an embedded system point of view [25].

Figure 1.4 depicts a simplified IoT ecosystem, from the sensor to the cloud, from an embedded system point of view [25]. It can be divided in four main components: (1) the Thing, which represents the physical sensor, i.e., the device that is able to collect and generate data; (2) the Local Network (which may also include specific gateways) that is responsible to collect data-in-transit, perform some additional processing such as data aggregation and further data retransmission over the Internet; (3) The Internet, which consists of the big infrastructure that connects all the participating entities, servers, gateways, routers, web services, databases, and so on; (4) the Back-End Services, which are mainly responsible to receive data-in-transit, keep data-in-rest and make them available to authorized users and business data analytics.

In actual fact, IoT systems are inherently complex. For every target context, it is crucial to define how the participating devices can efficiently communicate, interoperate and securely exchange information among them and/with remote cloud web servers. Such implementation involves a wide set of technology layers e.g., cloud services, communication protocols, connectivity interfaces, embedded device software, embedded device security, and so on. On each technology layer, the complexity increases, as there are several choices to perform specific requirements and tasks.

IoT Architecture Reference Models

To drive the realization of the IoT, tremendous efforts in creating IoT alliances and consortium, reference architecture (RA) models, and design guidelines have been issued by most of the important players in industry. As wireless communication technologies proliferate, industry players are creating partnerships to unite the IoT by developing consistent standards and broad interoperability in all the important sectors. For instance, Intel IoT Solutions Alliance, Industrial Internet Alliance, OneM2M, AllSeen Alliance, Internet of Things Consortium, IPSO Alliance, Alliance for Internet of Things Innovation, Industrial Internet Consortium, and LORATM have been formed to tackle the fragmented IoT industry in a multilayer approach. Other alliances and working groups, such as, IEEE, DASH7 Alliance, Wi-Fi Alliance, LoRa Alliance, ZigBee Alliance, focus on the communications layer, while the IETF, Open Mobile Alliance, UPnP, W3C focus on the messaging and transport layers. Moreover, and vertically focused on their application domain, other alliances also try to make their way: HealthKit and Wireless Life Sciences Alliance in wireless body networks solutions; HomePLUG, HomeKit, ZWave, Thread Group for home automation applications; Enocean Alliance and The Connected Lighting Alliance for smart cities and connected buildings; GENIVI and Open Automotive Alliance in connected smart transports; Modbus and HART also for the Industrial IoT.

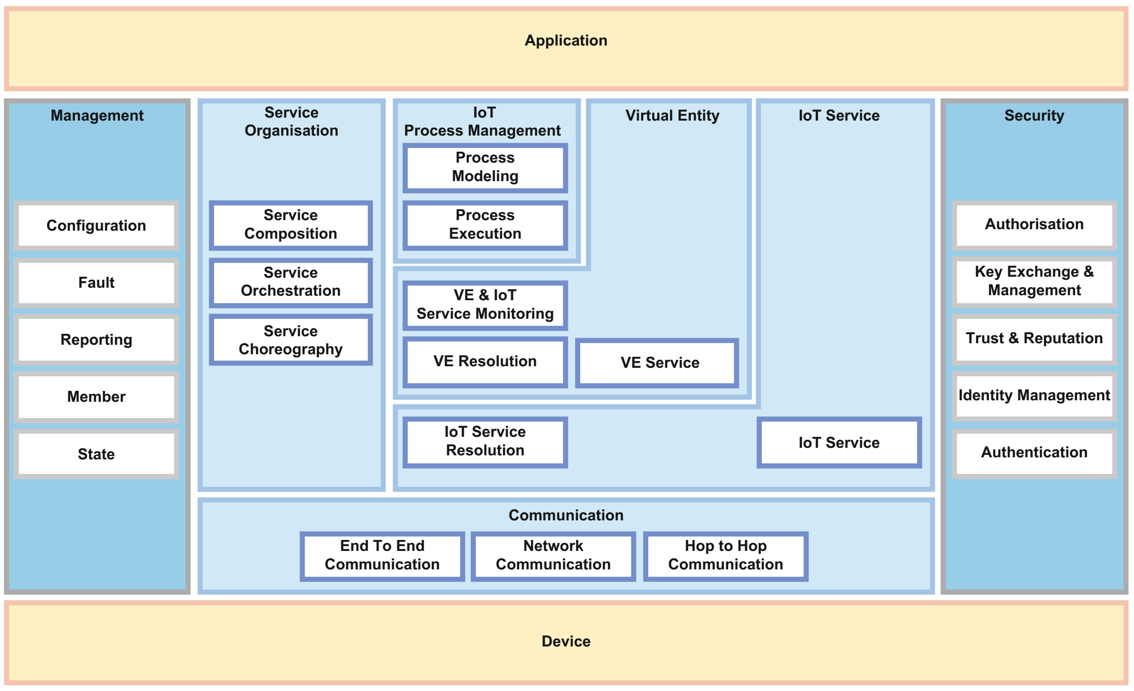


Figure 1.5: Functional-decomposition viewpoint of the IoT-ARM [26].

Aiming in defining solutions to tackle the interoperability at the communication level, as well at the service level across various platforms, several reference architectures have also been proposed by industry and academia [26–32]. Moreover, several guidelines to leverage secure designs and truly connected IoT endpoint devices have also been proposed [25,33–43]. Their main goal is to help and guide designers and developers in creating IoT solutions always bearing in mind several important common issues and concerns. For instance, Figure 1.5 depicts the functional view diagram of the Internet of Things - Architectural Reference Model (IoT-ARM), composed by nine functionality groups: Application, Management, Service Organization, IoT Process Management, Virtual Entity, IoT Service, Security, Communication and Device [26].

Connectivity and Security are a Paramount

The IoT requires solutions that allow for the creation of products adapted to the specific constraints of a certain market: security, power efficiency, standard-based communications and scalability. This leads to an increasing need for standardized and reliable networks. The IoT-ARM Communication Model aims at defining the main communication paradigms for connecting elements, as defined in the IoT Domain Model. It provides a reference set of communication rules to build interoperable stacks, together with insights about the main interactions among such elements of the IoT Domain Model. It proposes a Communication Model that leverages the ISO-OSI 7-layer model for low-rate networks and aims at highlighting those peculiar aspects inherent to the interoperation among different stacks, which are called interoperability features.

From the device communication point of view, three main requirements can be identified:

- **Connectivity**: Traditional connectivity barriers to the connectedness continue to drop. For instance, IPv6 overcomes the IPv4 limit by allowing for 340,282,366,920,938,463,463,374,607,431,768,211,456 (2^{128}) more people, processes, data, and things to be connected to the Internet. "Amazingly, IPv6 creates enough address capacity for every star in the known universe to have 4.8 trillion addresses" [10].
- **Interoperability**: Concerns the interoperability between heterogeneous devices and environments. Interoperability at the local network level, can be

achieved by employing a standard network stack or, when needed, the utilization of protocol gateways/translators in order to convert from one technology or standard to another, e.g., ZigBee to standard IP network.

- **Reachability**: Refers to the availability of the endpoint device on the network, over the time. It results from the utilization of standard protocols, such as the IPv6 on the normally resource-constrained endpoint devices, turning it directly reachable from everywhere, at any time, without the need for gateways or network translators.

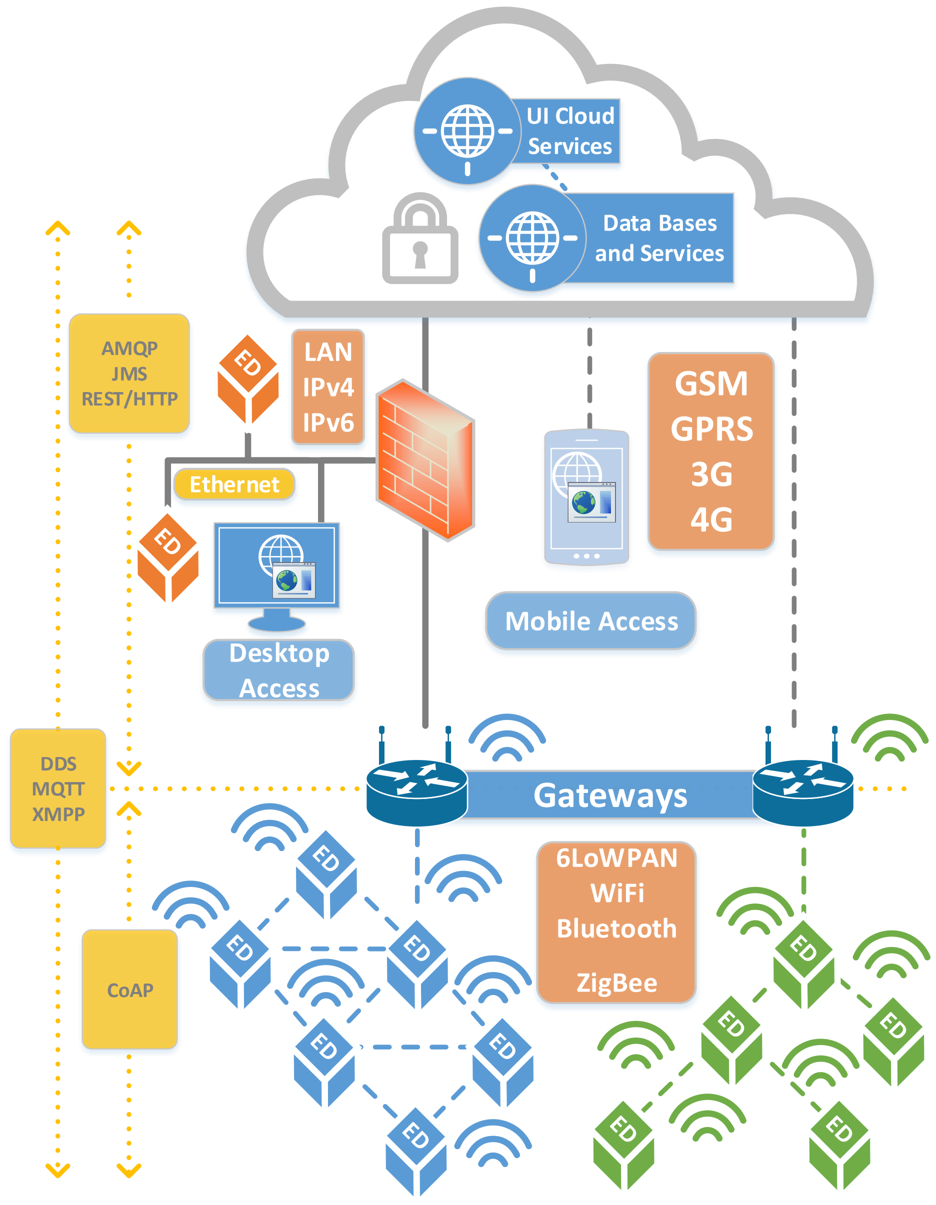


Figure 1.6: IoT Connectivity Problem Space. ED - endpoint device.

As mentioned before, the exponential growth of the IoT infrastructure leads to several challenges, among these: security, scalability and interoperability [32]. Scalable and standard communication protocols will better fulfill these requirements [36, 39], as standard protocols target the interoperability, contributing for a rapid development by easily enabling heterogeneous devices to communicate. However, this requirement is highly affected due to the existence of a multitude of proprietary solutions connected to the Internet, which resulted in low interoperability between devices and low scalable networks. This is pointed as the IoT Connectivity Problem Space, as illustrated in Figure 1.6 and identified by [36], which defines four variations of such problem: (1) inter-device communication; (2) device-to-cloud communication; (3) inter-data-center communication; (4) intra-device communication. These problems are proposed to be solved using standard protocols and messaging technologies, further discussed in the next section.

Challenges on the embedded devices in the IoT domain are due not only to the ever-growing amount of data to be handled, but are also due to the increasing importance of security and privacy [4, 5, 44, 45]. As long as security and privacy are concerned, data confidentiality, integrity and anonymity need to be guaranteed, as well as authentication and authorization mechanisms, in order to prevent unauthorized entities, such as devices and/or humans, from accessing the system components and data. It is also crucial to ensure both data protection and confidentiality, since devices may manage personal and/or sensitive information. Finally, trust is a fundamental issue since the IoT environment is characterized by different devices that have to process and handle data in compliance with user needs and rights. Trusted software and trusted devices must fulfill this requirement. From the endpoint perspective, security features must be provided at three distinct levels:

- **Data Security**: which mainly concerns the data security at the network level, i.e., data in transit. This is commonly provided by employing robust communication protocols and it is frequently described as the **CIA triad**:
  - **Confidentiality**: roughly equivalent to privacy, confidentiality ensures that data can be exposed to attackers or unintended end users. However, its content is always kept in secret. Data encryption is the traditional method applied to ensure data confidentiality.
  - **Integrity**: Involves maintaining data consistency (usually associated with data-in-transit), i.e., unauthorized people cannot change it during its life-cycle, without being noticed by the end destination.

  - **Availability**: Typically ensured by the device, which may keep data available and accessible when required, while keeping always the associated security measures.

- **Hardware Security**: It is, on the latest technology, provided by secure silicon and any other secure hardware building block or feature, such as:

  - **Root of Trust**: Provided by highly secure hardware, typically includes, from the silicon manufacturing phase, unique security keys that are only known to the device and used for security keys exchange and by strong cryptography algorithms. Differential Power Analysis (DPA) counter measures are also included in the hardware security.

  - **Tamper detection and counter measures**: On-chip tamper detectors with available counter measures, which may include: device zeroization that is used to permanently erase sensitive data, such as cryptographic keys, device lock, among others.

  - **Crypto-engines**: Certified built-in crypto-engines needed by the most used cryptographic algorithms in data protection or in securing communication channels.

Few examples of mechanisms that provide a hardware-based security are: Intel Boot Guard [46], tRoot [47] and ARM TrustZone [48]. Recently extended to Cortex-M processors, ARM TrustZone can now be applied to endpoint devices [49]. TrustZone technology provides a foundation for system-wide security and the creation of a trusted platform. Any part of the system can be designed to be part of the secure world, including debug, peripherals, interrupts and memory. By creating a security subsystem, assets can be protected from software and common hardware attacks [48].

1.2 Scope

This thesis focus on the communication model and the security requirements for IoT endpoint devices operating at the network edge. It is motivated by the ever-growing number of resource-constrained devices and the challenges the designer faces to satisfy such important requirements to the device. Concerning the connectivity, this research work is motivated by the connectivity space problem identified in [36] and tackled by the IoT-ARM Communication model and other IoT guidelines [32,39,50].

Standardized Communication Stack

The Internet as we know it cannot address the expected big number of connected endpoint devices. Although the Internet Protocol version 4 (IPv4) leverages a good infrastructure and robust protocol to reach devices from anywhere, it cannot provide unique global reachability since it is limited to 32-bit of singular addressable interfaces and it was not initially designed to handle such kind of devices. As seen above, IPv6 is the key for connecting myriads of smart devices on the new Internet era [37, 50]. Initially conceived to support scalability, with 128-bit for unique addressing along with other enhanced and new features, it allows all devices to be singly identified and reachable at any time and from anywhere.

Traditional WSN, whose forming devices are mainly resource-constrained in terms of memory, processing capabilities, power source, etc., are predominantly IEEE 802.15.4-based networks. On such networks, the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) protocol is the key to allow the usage of the IPv6 protocol, as it enables IPv6 packets to be sent and received over IEEE 802.15.4 Data frames [50,51]. Due to its lightweight implementation and its ability to enable the ubiquitous connectivity and interoperability between heterogeneous devices, the 6LoWPAN has become a de facto standard for low-rate wireless personal area networks (LR-WPAN), adopted by many machine-to-machine (M2M) communication systems and protocols [39,52]. Thus, in order to use the most of the IPv6 standard with the current technology, the 6LoWPAN was also designed to support a wide set of features, such as packet fragmentation and packet reassembly, packet header compression mechanisms, network protocols, such as Internet Control Message Protocol (ICMP), among others.

Figure 1.7 depicts the proposed stack to be used in the network edge, which aims to increase the connectivity and the interoperability among devices [26]. For instance, interoperability can be achieved at any layer, depending on the selected technology, i.e., for the Wi-Fi and the IEEE 802.15.4 this is achieved in Layer 3 where the IPv6 is the common protocol. The 5-layer stack can be perceived as follows:

- **Layer 1**: The physical layer (PHY) is provided by the IEEE 802.15.4, as this standard is the foundation of LR-WPAN and proven to suit the PHY.
- **Layer 2**: The Data-Link layer is composed by the Medium Access Control (MAC) layer (which is also defined by the IEEE 802.15.4 standard) and the adaptation layer provided by the 6LoWPAN standard protocol. Thanks to the

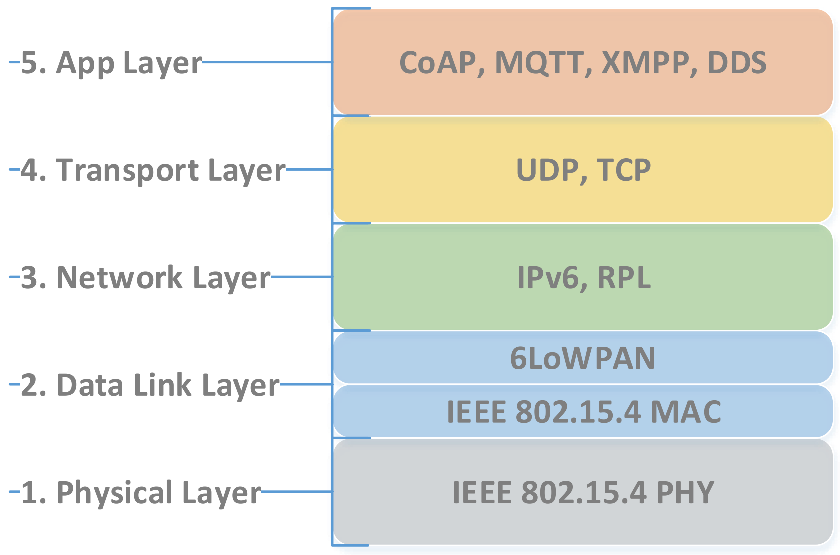


Figure 1.7: 5-layer IoT-ARM Communication Model [26].

6LoWPAN, the IPv6 protocol can now reach even the tiny resource-constrained devices.

- **Layer 3**: The Network layer, typically represented by the IPv4, is now upgraded to support IPv6. The Routing Protocol for Low power and Lossy Networks (RPL) resides also at this layer of the stack.

- **Layer 4**: The Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) protocols are still the most suitable for handling the Transport layer requirements of the stack. However, the message control mechanism used by TCP can be hard to handle in extremely resource-constrained devices. In such cases, the UDP is the most suitable protocol to be used.

- **Layer 5**: Traditional web technologies, such as Hypertext Transfer Protocol (HTTP), cannot be efficiently used by endpoint devices, thus new protocols are proposed to replace them. The Constrained Application Protocol (CoAP) is defined to be used over UDP, while the Extensible Messaging and Presence Protocol (XMPP) and Message Queuing Telemetry Transport (MQTT) may run over TCP.

Security Concerns

In what concerns the security requirements, software-based security approaches do not provide the strongest protection, as many are led to believe [53]. Hardware-based security delivers a much more rock-solid methodology. Today's embedded security devices can provide an easier and lower cost way to integrate endpoint devices designs early on with layers of advanced security, support for cryptographic algorithms, tampering detection, and many other protections. As the communications definitely require strong end-to-end security to ensure protection against a variety of attacks [45, 54–56], such important component must also benefit from hardware approaches that aim to provide hardware security as well as acceleration to mitigate the complexity of the communication and data security protocols. In what concerns the protocols, some approaches for deploying hardware-assisted network stack capabilities have already been tried, e.g., in [57] an "RTOS in hardware for energy efficient Software-based TCP/IP Processing" was proposed, while in [58] specific packet processors have been used to perform specific application-related tasks. However, concerning the network layer specified by the IoT-ARM, only few hardware-assisted approaches have been partially attempted. Regarding cryptographic algorithms, several devices already provide on-chip crypto-engines to accelerate the processing of the most important and most used encryption standards and algorithms. These are later discussed in this thesis.

Modeling and Automation Enabling Tools

The configuration and deployment of an endpoint device can be complex, mainly due to the devices' hardware heterogeneity and the high variability of the embedded operating system (OS) and network stacks. The task of configuring and customizing network parameters, such as the device's MAC and IP addresses, as well as OS services and protocols, e.g., 6LoWPAN for the network and CoAP for the application layer, can be mitigated by enabling design automation through the development of a tool that allows full system configuration and code generation, according to the user needs and application requirements. The applicability of such tool can be explored in a way that generating firmware for several nodes in an IoT network, while providing mechanisms for code verification and validation, can be performed by automated systems and/or embedded systems designers without deep knowledge of the OS and/or the network stack. Usually these tools are based in a domain-specific language (DSL), and target a specific domain with specific problems [59–61].

Several approaches targeting design automation by providing a DSL to model a desired system, have already been undertaken in the recent years. For instance, dynamic code-generation in binary translation systems [62], general system design both at the software and register-transfer level (RTL) layers [63], business-oriented DSL for sensor networks [64] and IoT-based applications, where the complexity and the heterogeneity of the WSN nodes is facilitated by the DSL-4-IoT [65]. In this thesis, a configuration tool based on a DSL that targets an embedded communication stack, which runs on IoT endpoint devices, is also explored. Such language will help reduce and simplify the development time by promoting a design automation tool that can configure, and automatically generate code, ready to be compiled, for devices running an IoT-compliant communication stack.

Out of Scope

As already stated, this thesis focus on IoT endpoint devices at the network edge. Since they are mainly resource-constrained, the connectivity, interoperability and security features require optimizations not only at the network/communications level, but also at the device's hardware itself. This primarily aims to tackle the large amounts of data exchanged between endpoint devices, which is mainly caused by the standardized stack and the overhead introduced by the new protocols supported. Furthermore, such optimization problem is compounded by security-related issues raised by connecting things to the Internet. As already explored in [53], and because commercial off-the-shelf (COTS) solutions are based on homogeneous architectures that do not provide full flexibility to explore acceleration capabilities, this thesis will focus on exploring heterogeneous solutions to mitigate the emerging challenges brought by the IoT-ARM Communication and Security model requirements.

Although related, this thesis does not directly concerns with:

1. Energy consumption analysis and energy-saving techniques.
2. Middle- and high-end devices, even if they operate at the network edge.
3. High-end communication that use other standard communication protocols, rather than the simplified 5-layer stack presented in Figure 1.7.
4. High-level security, such as complex end-to-end data security protocols like Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS).
5. Security at the OS level.

1.3 Research Questions and Methodology

By employing a standardized communication stack at the network edge, not only promotes the connectivity and interoperability among all devices, but also substantially increases the amount of data exchanged. While the handling of such ever-growing amounts of data on resource-constrained devices will mainly compromise its performance and the overall energy consumption, the connectivity will totally expose the device to the Internet, which will raise several security issues. Given this, this thesis poses this main question:

How to leverage an accelerated, secure and energy-aware IoT end-device communication?

For a better understanding, such a broad question can be further divided as follows:

1. How to identify functionalities and common operations for hardware-acceleration according to the specificities of the endpoint device under design?
2. How to implement and deploy the identified accelerators' candidates according to the demanded design metrics?
3. How to further improve the endpoint communication under the demanded design metrics?
4. How to contribute for a secure, reliable and trustable endpoint device?
5. How to leverage reduction in terms of engineering effort, while tackling security at the onset?

To evaluate these questions, the following methodology is proposed:

1. *Exploration of heterogeneous solutions for the low-end devices at the edge network*, in order to enhance the performance by deploying acceleration modules into the solution. Such accelerators not only comprise application-specific tasks but also typical operations in any IoT endpoint device, such as network-related tasks.

2. *Adopt a hardware-software co-design methodology*, which contributes to find strategies to help identify the best offloading approaches to integrate customized accelerators.

3. *Improve the endpoint communication by accelerating the network stack, in a bottom-up strategy*, since IPv6 is the key to interconnect the myriads of endpoint devices that participate in the IoT. Accelerating the network layer (IPv6) requires acceleration at the MAC layer (IEEE 802.15.4) as well, because this standard is the foundation for the IPv6 support on the low-end devices at the network edge. This not only improves the endpoint performance but also enables the device to explore energy-saving operation modes for larger periods of time, which contributes for a better energy usage.

4. *Deploy secure endpoints at the network edge*, which can be achieved by employing cutting-edge secure hardware architectures, as well as secure and reliable communication protocols to secure and protect data-in-transit over the network. In an age of an unprecedented technological development, where devices and data security are a paramount, this strategy will play a key role in the future of the endpoint devices.

5. *Development of a DSL that aims to promote and facilitate design automation tools.* Such tools contribute for a fast and reliable development of different solutions based on the same low-end device capabilities.

1.4 State-of-the-Art

COTS Endpoint Devices at the Network Edge

Traditional WSN systems are homogeneous architectures based on software tasks running on processors ranging from a small microcontroller (MCU) to a more complex Digital Signal Processsor (DSP), which mainly integrates or connects to a radio transceiver used for the communication requirements. From available COTS solutions, several available devices developed by academia or the industry can already provide the connectivity and interoperability requirements [66–72]. However, they fail into provide the flexibility of exploring heterogeneous architectures with acceleration capabilities and the security-related requirements. Thus, they are considered out of the scope of this thesis.

Heterogeneous Architectures for Low-end Devices

Due to the increasing complexity of heavy sensing-related tasks on wireless motes, field-programmable gate array (FPGA)-based heterogeneous architectures, which combine a reconfigurable computing unit (RCU) and a low-power MCU with an attached radio transceiver, are gaining special focus. Such sensing wireless solutions that rely on FPGA, either based upon standalone platforms or as a combination of an MCU and FPGA, are proven to play a key role in the future of sensor networks [73, 74] in fields where heavy processing capabilities such as strong cryptography algorithms [75–80], low-level hardware security [81, 82], self-testing and data compression schemes [83–86], image and multimedia processing [87–91], local and remote reconfigurability [92–99], among others, are a major requirement in a broad range of applications [100–102].

With the arising interest in creating heterogeneous IoT-enabled wireless motes, sensor systems based on FPGA have included an IEEE 802.15.4-compliant radio transceiver [103] in the solution, in accordance with the IoT-ARM specification for the layer 2. The low-priority and the low-level radio operations are usually kept on the MCU, while complex and performance-demanding tasks are executed in the RCU. Such tasks typically consist in collecting data from available sensors, followed by its subsequent processing and aggregation for further transmission over the network. From the current state-of-the-art and the available literature, the most relevant research, closely related with the hard-core of this thesis, are the HaLoMote [104], the CookiesWSN [105], the PowWow Mote [106], among others [107–109], which are discussed below.

The HaLoMote [104] is described as a hardware-accelerated low-power mote targeting the IoT. It combines FPGA technology, used for energy-efficient data aggregation, with a system-on-chip (SoC) radio transceiver (the ATmega256RFR2 from Microchip) used for network management and data exchange. Integrating the RCU into a sensor mote and endowing it with wireless capabilities, this solution aims to provide a reconfigurable and energy efficient wireless architecture for IoT low-end endpoint devices. The RCU performs a dynamic power management (DPM) system and lossless data compression operations for application-specific implementations. In [110], the HaLoMote deploys a forward-adaptive differential pulse code modulation with a Rice symbol coder to compress vibration data from microelectromechanical systems sensors, while in [111], its previous version, it is used in a structural health monitoring system that performs data aggregation mechanisms to reduce the collected data before they are sent through the wireless interface. The first version of the HaLoMote included a CC2531 RF-SoC with an integrated IEEE 802.15.4 radio transceiver, while the latest version is composed by an 8-bit AVR ATmega256FRTR2 RF-SoC, also with integrated radio transceiver, with an Microsemi Igloo M1AGL1000 FPGA (which performs the RCU) for the hardware-accelerated computations.

The CookiesWSN mote [105] and the PowWow Mote [106] have combined a TI MSP430 MCU with FPGA technology beside the CC2420 IEEE 802.15.4 radio transceiver. They mainly differ in the type of application supported and the type of accelerators used, which is directly related with the application domain. The CookiesWSN mote provides several types of accelerators, such as elliptic curve cryptography (ECC) [112] for helping in securing the WSN communications, a specific sensor data processing (SDP) accelerator, applied in a coffee factory to monitor the manufacturing process as well as some environmental parameters [113], and a reconfigurable Kalman Filter - also known as a linear-quadratic estimator (LQE) - to remove noisy samples during data acquisition [114]. Additionally to the LQE accelerator, and aiming to provide Data Security mechanisms (data integrity and data authentication), the CookiesWSN mote includes acceleration blocks for the Secure Hash Algorithm 1 (SHA-1) and MD5 Message-Digest Algorithm (MD5) algorithms. However, such algorithms are nowadays discouraged from being used due to several well-known attack surfaces and vulnerabilities [115].

The PowWow mote explores the FPGA in order to deploy low-level network-related accelerators. Such accelerators aim to assist the Contiki-OS, which is hosted by the MCU, in some link-layer tasks, such as error correction codes (ERCC) mechanisms. The goal of the link-layer is to manage the Automatic Repeat reQuest (ARQ) and the forward error coding (FEC) to ensure a reliable link established between two nodes. The FEC at the link-layer can be done using several techniques among block and convolutional code. The ARQ mechanism used is known as selective acknowledgment (SACK) where only erroneous packets are requested to be retransmitted. This way the MCU is kept in a low-power mode when the received IEEE 802.15.4 frames are not intended to be forward to Contiki-OS. Aiming to improve the energy efficiency, PowWow also includes a dynamic voltage and frequency scaling (DVFS) feature to handle the digital processing part. This feature consists in a power management technique where the voltage of the MCU can be decreased to lower levels (when not in use) in order to minimize the energy consumption.

Other relevant contributions on this field [107,108], discuss important advances in reconfigurable systems oriented for FPGA-based WSN applications, where once again, application-specific tasks are deployed on FPGA in order to increase the systems performance. Vera-Salas et al. [107] used a micropositioning measurement system to test and deploy their platform. The deployed accelerators consist of a specific SDP for the given application, a root mean square (RMS) statistical process for data analysis and interpretation, a finite impulse response (FIR) filter for signal processing, a fast Fourier transform (FFT) algorithm used for differential digital signal processing, and algorithms for signal and image compression such as discrete wavelet transform (DWT) and continuous wavelet transform (CWT) [116,117]. Nyländen et al., have also deployed in their solution RMS, FFT and SDP accelerators. The main difference between both is the absence of an MCU on the first, and the presence of a transport triggered architecture (TTA) soft-core processor on the second. Aiming to tackle security-related issues, highly secure WSN nodes with an efficient cryptosystem are proposed in [109]. The main goal is to provide cryptographic security to the low-power WSN nodes, while maintaining the energy efficiency and performance requirements. Stelte [109] proposes a heterogeneous architecture for WSN motes that comprises a soft-core MCU that can deploy SDP and ECC accelerators to assist security-related tasks. Also, he proposes the addition of device hardware security features by introducing ARM TrustZone technology to his mote. However, such mote was only proposed and never implemented. Other important components such as the radio interface and the network-related accelerators are also present in the architecture but not addressed in the publication.

In short, by exploring low-power operation modes with very low static energy drain (provided by modern FPGA flash-based technology) and using a DPM system allied with a DVFS or dynamic voltage scaling (DVS) technique [118,119], heterogeneous solutions emerge as a great option for low-power heterogeneous WSN motes. Applying the DPM principle, the mote can select single components to be completely shut down in idle or low-power modes, contributing for further energy saving schemes. However, by only performing application-specific tasks on the RCU, such solutions are confined to their applications and domains. With the ever-growing amount of data over the network that needs to be handled, and considering the standardized communication stack, network-related tasks must be also considered to be performed by the RCU.

GAP Analysis

Table 1.1 summarizes the gap analysis on heterogeneous architectures that target WSN applications, displaying their differences on the most important features when developing low-end devices for the edge network.

Table 1.1: Gap analysis between existing heterogeneous solutions. N/A - feature not available, N/P - information not provided.

	HaloMote [104]	***Cookies WSN*** [105]	***PowWow*** [106]	***Vera-Salas et al.*** [107]	***Nyländen et al.*** [108]	***Stelte*** [109]
Application Support	bare metal	bare metal	Contiki-OS	bare metal	bare metal	N/P
Real-Time Support	No	No	No	N/P	N/P	N/P
IoT-ARM Stack	No	No	Yes	No	N/A	N/P
Radio IEEE 802.15.4	AT256FRTR2 (RF-SoC)	CC2420	CC2420	MRF24J40	N/A	N/P
MCU Arch. (Type)	8-bit AVR (RF-SoC)	TI MSP430 (MCU)	TI MSP430 (MCU)	N/A	TTA soft-core (FPGA)	MCU soft-core (FPGA)
Acceleration Used	RDT [111] Rice [110]	ECC [112] SDP [113] LQE, SH-1, MD5 [114]	ERCC, DFVS	RMS, FIR FFT, SDP	RMS FFT, SDP	ECC, SDP
Net. Acceleration	MAC filter (RF-SoC)	MAC addr. filter (RF)	MAC addr. filter (RF), ARQ & FEC (FPGA)	MAC addr. filter (RF)	N/A	N/P
Advanced Net. Acceleration	No	No	No	No	No	N/P
Device Security	No	No	No	No	No	N/A
Data Security	N/P	ECDSA, SH-1, MD5	N/P	N/P	N/A	N/A
Maturity	Proto./Final	Proto./Final	Proto./Final	Proto./Final	Concept/Proto.	Concept

Such requirements/characteristics are described as follows:

- ***Application Support:*** It characterizes the solution in terms of the application support, which can use a bare metal approach or add an embedded OS to the system. This latter option often includes interoperability support and abstracts the communication stack, which is very useful for rapidly connecting smart things to the Internet.

- ***Real-Time Support:*** It evaluates the real-time (RT) support of the global solution, which means that the RT capabilities shall be provided not only by the hardware, but also from the Application Support (bare metal or OS).
- ***IoT-ARM Stack:*** The communication stack, as favored by the IoT-ARM, eases and promotes the connectivity and the interoperability among all devices. In this comparison, the minimum requirement is the compliance with the Network layer which enables the communication at the IP level (IPv6).
- ***IEEE 802.15.4-compliant radio transceiver:*** The IEEE 802.15.4 is the recommended standard for the MAC layer, also defined by the IoT-ARM stack model. Since the main goal is to work with endpoint devices in the edge network, the IEEE 802.15.4 is set as a primary requirement.
- ***MCU Architecture (Type):*** It concerns with the MCU architecture used by the presented solution. It is also considered its deployment type, which can be either a hard-core or a soft-core MCU performed by available FPGA.
- ***Acceleration Used:*** It refers to the available accelerators deployed on the proposed solution. They can range from application-specific accelerated tasks, or generic algorithms that can be used in different application scenarios.
- ***Network Acceleration:*** It refers to available accelerators used for network-related tasks. Such accelerators are set available in the RCU or traditionally provided by the radio transceiver.
- ***Advanced Network Acceleration:*** Since the studied solutions are based on heterogeneous architectures, this enables the deployment of customized accelerators for the higher layers of the network stack, e.g., Network, Transport and Application layers. Such accelerators are classified in this analysis as *Advanced Network Accelerators*. If supported, they must be specified.
- ***Device Security:*** It concerns with the device's security features from the hardware point of view. Such features may comprise security mechanisms to prevent unauthorized reproductions of the deployed system, reverse engineering, on-chip security keys, cryptographic hardware accelerators for secure algorithms and protocols that can also assist available Data Security features.

- ***Data Security:*** It approaches available mechanisms that can provide data security, comprising the three major requirements when developing secure communications and secure data exchange: confidentiality, integrity and authenticity. Such requirements can be achieve by security mechanisms that can resort software- and/or hardware-assisted cryptographic blocks.
- ***Maturity:*** This metric classifies the maturity state of the solution: (1) Concept, which refers to an idea or proposal but not yet implemented; (2) Prototype, which consists of a real implementation of the solution but with on-going developments; (3) Final product, when the solution reaches a high level of maturity and may be already in use in real case scenarios.

When the information is not provided or cannot be retrieved, the keyword N/P is used, and in case the solution is only at its concept stage or the feature is not available, the keyword N/A is displayed. Trading-off all the requirements and the characteristics of the identified works, it is clear that is hard to achieve a device that fulfills all the desired characteristics.

1.5 Thesis Contributions

This thesis aims to improve the state-of-the-art with the following contributions:

- A reconfigurable heterogeneous architecture for low-end devices in the IoT network edge;
- A secure endpoint device deployment at the network edge, which can benefit both from secure hardware architectures and from secure and reliable communication protocols;
- An improved endpoint communication with network stack acceleration for network services and communication protocols performance, with resilience enhancement against common denial-of-service (DoS) attacks.
- A DSL to facilitate design automation tools, contributing for a fast and reliable development of IoT low-end devices.

1.6 Thesis Structure

The remaining of this thesis is structured as follows:

- **Chapter 2** presents the research tools and materials used throughout the developed work, describing the requirements and motivations that led to their choice. Due to the lack of turnkey solutions, customized tools specially developed for assisting this work are also discussed in this chapter.
- **Chapter 3** shows the challenges of designing and developing a heterogeneous architecture for IoT low-end devices and the associated hard task of following a hardware/software co-design methodology. Such methodology is used to help select and evaluate software functionalities suitable to be offloaded and performed by dedicated hardware accelerators. The selected functionalities are mainly related to generic and network-related tasks. This chapter also presents the concept and prototype of the CUTE mote, described as a customizable and trustable end-device, which performs the proposed architecture and is able to deploy customized hardware accelerators.
- **Chapter 4** presents the design and deployment on an IEEE 802.15.4 accelerator that is able to filter and process the received Data frames as specified by the standard for the First, Second and Third-level of filtering, suitable for any IEEE 802.15.4-based network. As this standard is the foundation for the IoT network layer (IPv6 over IEEE 802.15.4 provided by the 6LoWPAN adaptation layer), further research relies on these developed functionalities.
- **Chapter 5** describes the development and deployment of an IPv6 Network accelerator in the CUTE mote, that is able to retrieve and generate IPv6 packets from the IEEE 802.15.4 Data frames (through the 6LoWPAN adaptation layer), in order to filter and process specific IPv6 packet header fields. Such accelerator performs some important tasks related to the 6LoWPAN standard, such as the implementation of the IPv6 header compression/decompression mechanism.
- **Chapter 6** presents automation tools developed through the design of a DSL. Due to the increasing complexity of adding new features at several layers of the network stack, this tool facilitates the development of customized solutions in highly configurable systems.

- **Chapter 7** concludes this research work and discusses the limitations faced during the development of the presented solutions. In conclusion, further work and research directions towards fulfilling the aforementioned limitations are suggested. Moreover, a list of publications that have directly or indirectly contributed for the development of this work are presented in this chapter.

1.7 Conclusions

This chapter contributed with a brief introduction about the core-study of this thesis. After the problems identified in the scope of this thesis, some research questions were raised in search of possible solutions. Such questions are then answered throughout the development of this thesis, following the proposed research methodology. Afterwards, it was identified the state-of-the art on the most related contributions and a gap analysis among them was performed, showing where such contributions fail to provide a complete solution that can directly or indirectly tackle the identified issues and how this thesis plans to address them. Finally, it was described a brief summary of the contributions and the remaining of this thesis. The next chapter presents the research platform and tools used during this thesis for the testing deployment and evaluation of the proposed solution.

Chapter 2

Research Platform and Tools

This chapter presents the research platform and tools used throughout this thesis, for the testing, evaluation and deployment purposes. Section 2.1 describes the platform requirements for researching on the proposed solution, while Section 2.2 discusses available embedded OS that deploy a network stack, suitable for constrained IoT endpoint devices. This section also contains a study on the hardware platforms supported by the embedded OS, which enables the possibility of further comparisons of the proposed implementation with other similar software-based COTS solutions, like the Wireless Connectivity family devices from Texas Instruments, described in Section 2.3. In Section 2.4 it is presented the selected hardware platform, which consists of a secure hardware solution that integrates on the same SoC flash-based FPGA and an MCU, enabling the deployment of heterogeneous architectures with customized accelerators for endpoint devices in the IoT domain.

Further results can be obtained in order to perform a total comparison between software-based COTS solutions and a hardware mote that deploys the heterogeneous architecture on the selected hardware platform. The platform also eases power consumption analysis, by including on-board measuring hardware circuitry. The development and simulation of the designed hardware, before its final deployment on FPGA, is done using the Libero SoC and ModelSim tools, provided by Microsemi and described in Section 2.5. For the software validation purposes, the Thread-Metric Benchmark Suite was selected over other available benchmarking suites.

2.1 Platform Requirements

When developing heterogeneous solutions, which leverage customized accelerators for IoT endpoint devices, several aspects must be considered. Due to the common low-budget concerns, e.g., price value, processing capabilities, low-power consumption, among others, the following requirements must be considered, both for the platform and for the final solution that is desired to achieve:

1. Support an **IoT-enabled communication stack** that promotes the three most important aspects when it concerns to the device's communication support: (1) Connectivity; (2) Interoperability; (3) Reachability.

2. Promote design **Scalability**, to easily accommodate more hardware acceleration engines.

3. Support component **Modularity** and device **Customization**, enabling only the selection of required components, in order to meet tight requirements and application constrains.

4. Enable solution's **Portability**, to reduce the time-to-market cost when developing such kind of embedded systems. Similarly to a COTS microcontroller architecture, which may be supported by most of the IoT-enabled embedded operating systems (OSes), the hardware platform must allow component's portability and smooth the integration of the architecture in other similar platforms.

5. Provide a **Secure Hardware Architecture**, in a commercially available low-power SoC. Such solution, combining both FPGA technology and a well-known MCU architecture suitable for low-end embedded systems, empowers a fast development of the research topics proposed in Chapter 1, while keeping the low-budget requirements. The SoC platform must facilitate the fast development of accelerators and simplify their integration in the form of soft-hardware peripherals, loosely coupled to the MCU architecture by using standard architectural communication buses.

6. Boost the incorporation of **Data Security** mechanisms, in accordance with the CIA triad requirements identified in Chapter 1.

Due the lack of available turnkey solutions for all the aforementioned requirements, and due to the fact that similar solutions are only provided in non-disclosed systems, it was necessary to select a hybrid SoC that enables the deployment of custom accelerators with an MCU architecture supported by an available embedded OS suitable for low-end IoT devices. Moreover, the selected embedded OS must also support all the wireless standards proposed in the IoT-ARM network stack, while keeping the interoperability requirements among all the available heterogeneous devices and supported hardware platforms.

2.2 Embedded Operating Systems for Low-end IoT Devices

From the sensor to the cloud, the participating devices in the IoT may range from simple and resource-constrained endpoint devices to extremely complex cloud servers and Internet hosts. While middle- and high-end devices, e.g., servers, routers and gateways, can support traditional OSes such as Linux, this is not possible with low-end endpoints. Moreover, traditional OSes currently running on high-end devices, or typical OSes suitable for traditional WSN, such as TinyOS [120] or Nano-RK [121], are not able to fulfill the diverse requirements of an IoT environment. To leverage the IoT, redundant development should be avoided where deployment and maintenance costs must be reduced.

Until now, only powerful computer hosts, robust embedded devices and complex networks have been able to participate natively with the Internet [122]. Direct communication with traditional IP networks requires many Internet protocols, often requiring an OS to deal with the inherent complexity and maintainability. Traditional Internet protocols are required for embedded devices for the following reasons:

- **Security**: IPv6 includes optional support for IP Security, traditionally provided by the Internet Protocol Security (IPsec) [123], authentication mechanisms and encryption schemes. Moreover, web services also make use of secure sockets or transport layer security mechanisms. However, these techniques are too complex, specially to be deployed on simple and resource-constrained embedded devices.

- **Web services**: Internet services today rely on web-services, mainly using the TCP, HTTP, Simple Object Access Protocol (SOAP) and Extensible Markup Language (XML) with complex transaction patterns.

- **Management:** Management with the Simple Network Management Protocol (SNMP) and web-services is often inefficient and complex.
- **Frame size**: Current Internet protocols require links with sufficient frame length (minimum of 1280 bytes for IPv6), and heavy application protocols require substantial bandwidth.

These requirements have in practice limited the IoT to devices with powerful processing capabilities, an OS with a full TCP/IP stack and with an IP-capable communication link. Still, it is expected from the low-end devices, extremely constrained in terms of computing power, available memory, communication, and energy source, to support an adaptive communication stack to integrate the Internet seamlessly.

Linux certainly is a mature, robust and developer-friendly OS that has been getting attention as a platform for IoT, supporting an ever-increasing number of embedded devices, particularly the ones that provide graphically rich user interfaces. However, there are thousands of applications for which Linux is ill suited [25], due to its large memory footprint and required random-access memory (RAM) for kernel support, turning this OS not suitable for endpoint devices at the network edge.

An embedded OS for the IoT must fulfill a set of important requirements [124]:

- **Small Memory Footprint**: since low-end IoT devices are much more resource constrained, particularly in terms of memory, this requirement is set to few KB, both for RAM and persistent read-only memory (ROM) requirements.
- **Support for Heterogeneous Hardware**: IoT faces nowadays a large variety of hardware and communication technologies. This requirement sets the OS to run on a wide spectrum of hardware, ranging from nodes based on a low-power MCU, to nodes powered by new generations of energy-efficient 32-bit processors, where the available memory sizes may also vary.
- **Network Connectivity**: The main key of the IoT devices is the ability to interconnect and communicate with one another or with the Internet. Since it is expected from IoT devices to connect the Internet seamlessly, the support for a standard communication stack [32, 39] is mandatory.
- **Energy Efficiency**: Since most IoT devices will be running on batteries, they must support energy-aware protocols in order to provide a better and more efficient energy utilization. This can be explored by managing the most energy-consuming elements of a WSN node such as the MCU, the radio transceiver and

Table 2.1: Embedded OSes from three different categories and their key features.

Name	*Category*	*MCU w/o MMU*	*<32 kB RAM*	*6LoWPAN*	*RT scheduler*	*HAL*	*Energy-efficient MAC layers*
Contiki-OS	Event-driven	✓	✓	✓	✗	✓	✓
RIOT	Multithreading	✓	✓	✓	✓	✓	✗
FreeRTOS	RTOS	✓	✓	✗	✓	✗	✗

available sensors. The MCU achieves low-power capabilities when operating in sleep or idle modes, while the radio transceiver contributes for the low-power consumption when energy-efficient MAC protocols are explored.

- **Real-Time Capabilities**: RT characteristics may be crucial in some IoT applications where precise timing and timely execution are mandatory, such as required by smart health-care applications [20].
- **Security**: Security plays an important role in the IoT and it must be also provided by the OS. A requirement (and challenge) for an OS for the IoT is to provide the necessary mechanisms (cryptographic libraries and security protocols) while retaining flexibility and usability that help in keeping the CIA triad requirements.

There are several embedded OSes provided open-source which are good candidates for the IoT such as the Contiki-OS [125], RIOT [126], FreeRTOS [127], TinyOS [120], OpenWSN [128], NuttX [129], eCos [130], uClinux [131], ChibiOS/RT [132], CoOS [133], Nano-RK [121], Nut/OS [134], among others. However, not all fulfill most of the desired aspects specified above. Table 2.1 summarizes the three best candidates, grouped according to their kernel type, that best represent their category: Contiki-OS, RIOT and FreeRTOS. Contiki-OS and RIOT stand out from the list due to their intrinsic support to the 6LoWPAN protocol stack, reduced RAM size that is required to run the kernel, and their support to energy-efficient MAC protocols. Such MAC protocols are fully supported by Contiki-OS but only partially present in RIOT. However, their support is scheduled to be included in the near future.

2.2.1 Contiki-OS

Contiki-OS [135, 136] is an embedded and open-source OS for IoT that enables connecting tiny low-cost and low-power microcontrollers to the Internet. The architecture and the network stack is depicted in Figure 2.1. It is implemented in C

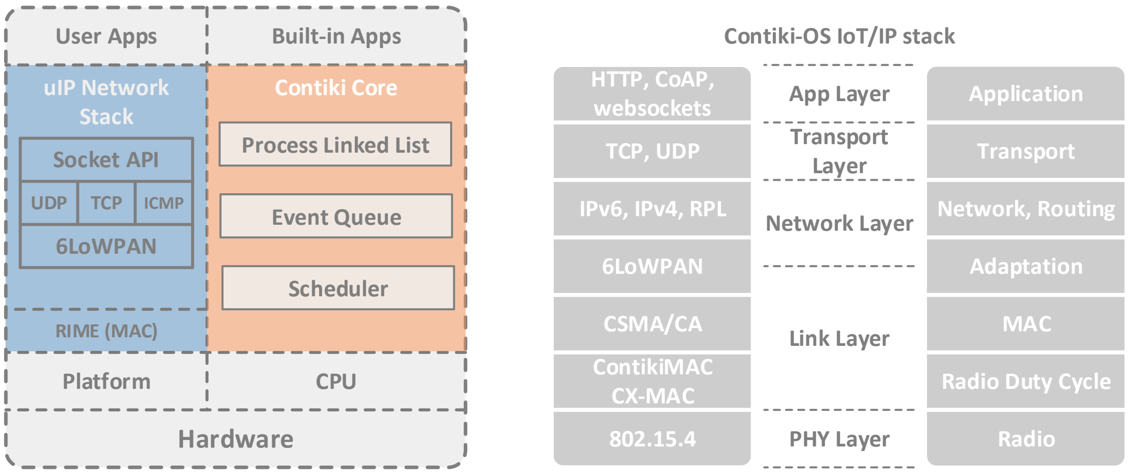


Figure 2.1: Contiki-OS stack and supported IoT/IP stack.

language and uses a make build environment for cross-compilation on most popular hardware platforms. The low-level hardware abstraction is split into `"Platform"` and `"CPU"` for portability purposes, which include hardware drivers, e.g., for sensors and other peripherals, for each different platform. In Contiki-OS, code in executes in either of two execution contexts: cooperative or preemptive. Cooperative code executes sequentially with respect to other cooperative code, while preemptive code temporarily stops the cooperative code. Processes are the primary way applications are run in Contiki-OS. They run, triggered by an event, in the cooperative context, whereas interrupts and real-time timers run in the preemptive context. There are two types of events: asynchronous events and synchronous events. When an asynchronous event is posted, the event is pushed into the kernel's event queue and delivered to the receiving process at a later time. When a synchronous event is posted, the event is immediately delivered to the receiving process. The purpose of the process scheduler is to invoke processes, by calling the function that implements their associated threads, when it is their time to run. All process invocation in Contiki-OS is done in response to an event being posted to a process, or a poll being requested for the process, and then the process scheduler passes the event identifier to the process that is being invoked.

For the network stack, it offers the Rime system, a flexible MAC layer and network protocol library which includes many low-level communication paradigms. The uIPv6 stack makes use of Rime, and provides a socket-like application programming interface (API) for use by applications, called protosockets. Both built-in and user applications are executed over Contiki-OS using a lightweight thread model called protothreads [136]. The uIPv6 stack provides a full network stack with all the layered protocol standards recommended by the IoT-ARM, supporting IPv6 and IPv4 Internet protocols along other recent wireless standards such as 6LoWPAN, RPL, CoAP, among others. For the MAC control, Contiki-OS implements by default the carrier-sense multiple access with collision avoidance (CSMA/CA) mechanism. On top of that, it offers different drivers for accessing the MAC layer, e.g., ContikiMAC [137] or XMAC [138] protocols. They offer an energy-efficient radio duty cycling (RDC) mechanism that allow the radio to be switched off when unneeded, fostering all the network devices to be battery-powered, even routers. Offering a variety of software applications and examples, the Cooja simulation tool, and a very active developing community, ease the development of software for a broad range of applications and scenarios.

Table 2.2: Popular Contiki-OS supported hardware platforms.

Platform	*MCU/SoC*	*CPU*	*Radio*	*Simulator*
CC2538DK	TI CC2538	ARM Cortex-M3	Integrated	✗
SensorTag	TI CC2650	ARM Cortex-M3	Integrated	✗
CC2530DK	TI CC2530	8051	Integrated	✗
wismote	TI MSP430x	MSP430	TI CC2520	✓
RE-Mote	TI CC2538	ARM Cortex-M3	Integrated, CC1200	✗
micaz	Atmel AVR	ATMega128L	TI CC2420	✓
Z1	TI MSP430x	MSP430	TI CC2420	✓
Sky	TI MSP430	MSP430	TI CC2420	✓
ESB	TI MSP430	TI MSP430	RFM TR1001	✓
MSB430	TI MSP430	TI MSP430	TI CC1020	✗
nRF52 DK	nRF52832	ARM Cortex-M4	Integrated	✗
exp5438	TI MSP430x	MSP430	TI CC2420	✓
redbee-xxx	MC1322x	ARM7	Integrated	✗
AVR-Raven	Atmel AVR	ATmega2561	AT RF230	✗

From the hardware side, Contiki-OS can run on small MCU architectures such as the AVR, the MCS-51 and the MSP430 and supports a wide range of low-power wireless devices and radio transceivers used by several research and commercial platforms. Table 2.2 summarizes the main Contiki-OS supporting platforms with the respective MCU and supported radio transceiver, which can be internally available or externally attached [139].

Cooja Simulator

Cooja is the Contiki-OS network simulator. Cooja allows large and small networks of Contiki-OS motes to be simulated. Motes can be emulated at the hardware level, which is slower but allows precise inspection of the system behavior, or at a less detailed level, which is faster and allows simulation of larger networks. Cooja is a highly useful tool for Contiki-OS development, as it allows developers to test their code and systems before running it on the target hardware. Developers regularly set up new simulations both to debug their software and to verify the behavior of their systems [125].

2.2.2 RIOT

RIOT [140–142], Figure 2.2, is also a small size embedded OS designed for the particular requirements of the IoT. Typical requirements comprise a low memory footprint, high energy efficiency, real-time capabilities, a modular and configurable communication stack, and support for a wide range of low-power wireless devices. RIOT was implemented as a microkernel and it provides utilities like cryptographic libraries, data structures (bloom filters, hash tables, priority queues), a shell and different network stacks. RIOT provides support to several MCU architectures, radio drivers, sensors, and configurations for entire platforms, e.g., CC2538DK or STM32 Discovery Boards.

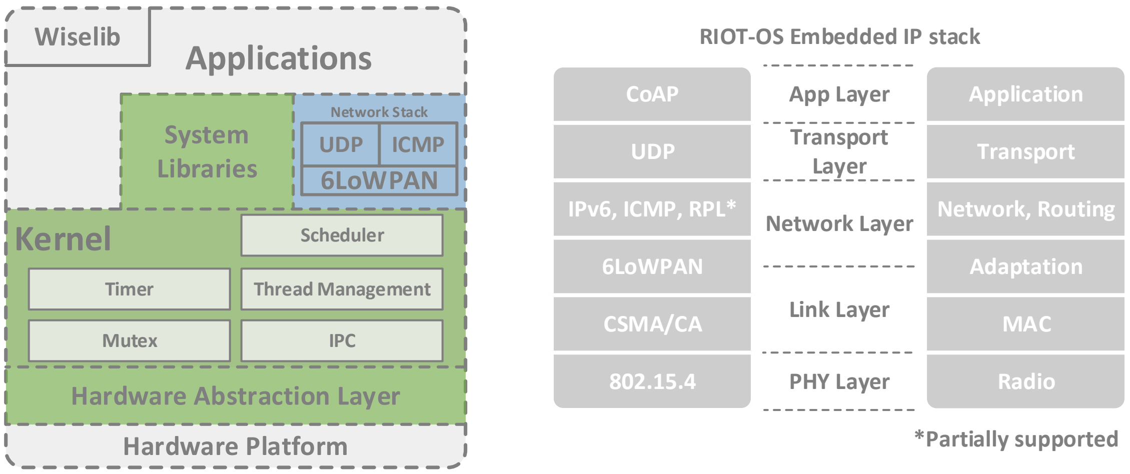


Figure 2.2: RIOT stack and embedded IP stack.

The microkernel architecture is inherited from FireKernel [142], comprising multi-threading, thread management, a priority-based scheduler, a powerful API for inter-process communication (IPC), a system timer and mutexes. To fulfill strong real-time requirements, RIOT enforces constant periods for kernel tasks (e.g., scheduler run, IPC, timer operations). An important prerequisite for guaranteed constant runtime duration is the exclusive use of static memory allocation in the kernel. Yet, dynamic memory management is provided for applications. Constant runtime of the scheduler is achieved by using a fixed-sized circular linked list of threads. The devices lifetime is related to the time the MCU spends in deep sleep modes. RIOT introduces a scheduler that works without any periodic events. Whenever there are no pending tasks, RIOT will switch to the idle thread, which determines the deepest possible sleep mode depending on peripheral devices in use. Only interrupts (external or kernel-generated) wake up the system from idle state.

Low complexity of kernel functions is a main factor for the energy efficiency of an OS. Therefore, the duration and occurrence of context switching must be minimized. In RIOT, context switching is performed in two cases: (1) a corresponding kernel

Table 2.3: Popular RIOT supported hardware platforms.

Platform	*MCU/SoC*	*CPU*	*Radio*	*Simulator*
CC2538DK	TI CC2538	ARM Cortex-M3	Integrated	✗
OpenMote	TI CC2538	ARM Cortex-M3	Integrated	✗
Z1	TI MSP430x	MSP430	TI CC2420	✗
RE-Mote	TI CC2538	ARM Cortex-M3	Integrated, CC1200	✗
UDOO	SAM3X8E	ARM Cortex-M3	N/A	✗
HiKoB Fox	STM32F103	ARM Cortex-M3	AT86RF231	✗
chronos	CC430	MSP430	CC1100	✗
telosb	MSP430x	MSP430	CC2420	✗
IoT-LAB M3	STM32F103	ARM Cortex-M3	AT86RF231	✗
Samr21-xpro	ATSAMR21	ARM-Cortex-M0	AT86RF233	✗
arduino-due	SAM3X8E	ARM Cortex-M3	N/A	✗
Arduino	ATmega2560	ATmega2560	N/A	✗
stm32iscovery boards	STM32Fxxx	ARM Cortex-M0/M4	N/A	✗
stm32nucleo boards	STM32F0xx	ARM Cortex-M0/M4	N/A	✗
pca10000	NRF51822	ARM-Cortex-M0	BLE	✗
msba2	LPC2387	ARM7	CC1100	✗

operation is called, e.g., a mutex locking or creation of a new thread, or (2) an interrupt causes a thread switch. The first case will occur rarely. For example, every thread is usually created once. Hence, it is important to reduce the processing time in case of a thread switch. Therefore, RIOT's kernel provides a minimized context switch that enables a task switch to be performed in few clock cycles.

The network stack is IoT-compliant and offers the most important standards and protocols such as IPv6, UDP and CoAP, yet, does not give full support to some other protocols, e.g., the RPL. Simulation is not directly supported, but it is still possible through modifications to the generated firmware which can run in Cooja under the supported Contiki-OS hardware platforms. However, this can be hard to be performed. RIOT also provides a great variety of hardware [143], as summarized in Table 2.3. However, the support to some platforms is still in progress as, for those, the radio connectivity is not yet implemented.

2.3 TI's Wireless Connectivity Portfolio

Table 2.4: Texas Instruments Wireless Connectivity portfolio.

	Smart RF transceivers	*Wireless network processors*	*Wireless microcontrollers (MCUs)*	*Wi-Fi combo*
Application	✗	✗	✓	✗
Wireless stack	✗	✓	✓	✗
RF Radio	✓	✓	✓	✓

Despite Contiki-OS being supported and widely tested over a wide range of hardware [139], the CC2538 and the CC2650 devices, along with their respective development hardware, are selected (by the development team) as the primary Contiki-OS hardware platforms. These are part of a wide range of devices composing the Texas Instruments Wireless Connectivity portfolio, presented in Table 2.4 [144]. Such devices accomplish the Connectivity, Interoperability and Reachability requirements when developing connectivity for IoT devices. Each category contains a broad range of devices suitable for almost any kind of application with a vast variety of requirements:

- **SmartRF transceivers**, such as the CC2420 and the CC2520, are used for connecting any MCU that does not directly provide wireless capabilities. The CC2520 is the latest IEEE 802.15.4 radio transceiver and provides extensive hardware support for frame handling, data buffering, burst transmissions, hardware data encryption, data authentication, clear channel assessment, link quality indication and frame timing information. These features reduce the load on the host controller.

- **Wireless network processors**, suitable for connecting IoT applications to the cloud. Available standalone (without MCU) or on a SoC, they run network related operations such as web servers or TCP/IP protocol handling. For the wireless connectivity, they provide standard IEEE 802.11 support. These devices mostly play the role of a middle-end device since they provide more powerful capabilities, yet, they are not suitable for battery-powered nodes.

- **Wireless MCU SoC** solutions (Wireless MCU, SimpleLink and Wireless Connectivity families), which are composed by a wireless MCU with integrated IEEE 802.15.4 radio transceiver. Available with different MCU architectures, they also provide several options in terms of the supported radio transceivers and communication standards. Table 2.5 summarizes some of the available devices with 6LoWPAN and Bluetooth Low Energy (BLE) support.

- **Development Platforms**, such as the SmartRF05EB, SmartRF06EB and SensorTag, by giving support to their respective devices (Table 2.5), they can provide low-cost solutions for testing and prototyping wireless nodes in the IoT domain.

Table 2.5: Wireless MCU SoC devices with 6LoWPAN and BLE support.

Device	***CPU***	***Radio Standard***	***Family***
CC1310	ARM Cortex-M3	IEEE 802.15.4 (Sub-1 GHz)	SimpleLink
CC2530	8051	IEEE 802.15.4 (2.4 GHz)	Wireless Connectivity
CC2540	8051	BLE 4.0	SimpleLink
CC2538	ARM Cortex-M3	IEEE 802.15.4 (2.4 GHz)	Wireless Connectivity
CC2564	ARM Cortex-M3	Bluetooth BR/EDR & BLE 4.1	Wireless Connectivity
CC2630	ARM Cortex-M3	IEEE 802.15.4-based (2.4 GHz)	Wireless Connectivity
CC2640	ARM Cortex-M3	IEEE 802.15.4-based (2.4 GHz)	Wireless Connectivity
CC2650	ARM Cortex-M3	IEEE 802.15.4 & BLE 4.2	SimpleLink

2.4 Microsemi's SmartFusion2 SoC

Regarding the hardware platform, several modern FPGA solutions could have been considered such as PolarFire and IGLOO2 FPGA Family [145, 146], but they primarily fail into provide a hard-core MCU, which was set as a major requirement to comply with this thesis research work. In what concerns hybrid platforms with both MCU and FPGA support, from the available solutions, Zynq-7000 [147], Cyclone V [148], LatticeXP2 SoC [149] or PSoC 5 [150], none of them perfectly suits low-end endpoint devices with the previously demanded requirements. For instance, PolarFire, IGLOO2 and LatticeXP2 provide the MCU only in a soft-core version, while Zynq-7000 and Cyclone V, widely used for supporting hardware security, are only available with powerful hard-core ARM Cortex-A8/A9 processors, thus not suitable for the desired solution. Although PSoC families providing FPGA technology with low-power hard-core ARM Cortex-M CPUs, they are very limited in terms of available hardware reconfigurability and customized peripherals support.

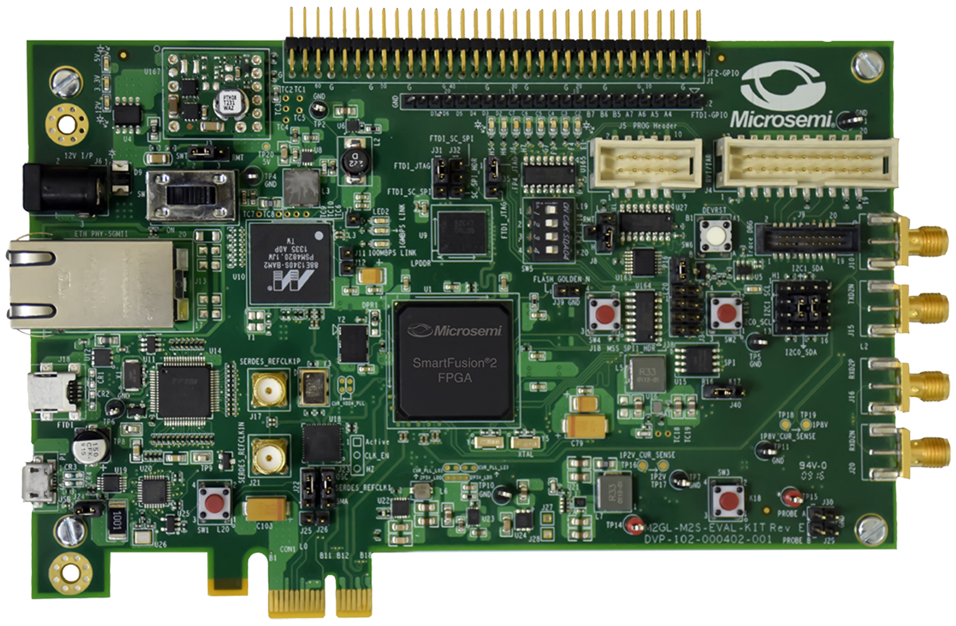


Figure 2.3: SmartFusion2 Security Evaluation Kit.

The SmartFusion2 SoC, included in the SmartFusion2 Security Evaluation Kit (Figure 2.3) from Microsemi [151], perfectly fits the hardware platform requirements. It consists of a cost effective SoC that integrates reliable flash-based FPGA fabric along a 166 MHz ARM Cortex-M3 processor and high-performance communication standard interfaces. Such interfaces include a 1Gbit Ethernet, a full-duplex Serializer/Deserializer (SerDes) lane, and a 64-bit general-purpose input/output (GPIO) header interfacing the SoC hard-peripherals. In addition, the SmartFusion2 memory management system supports 512 Mb on-board low-power double data rate (LPDDR) synchronous dynamic random-access memory (SDRAM) and a 64 Mb Serial Peripheral Interface (SPI) flash memory.

The Security Evaluation Kit offers the SmartFusion2 SoC M2S090TS with 90K logic elements (LE) FPGA along the hard-core MCU, including Embedded Trace Macrocell (EMT) and Instruction Cache with on-chip embedded static random-access memory (SRAM), embedded Non-Volatile Memory (eNVM) and a microcontroller subsystem (MSS) with extensive peripherals including CAN, TSE and USB. Architecture highlights include advanced security processing accelerators (DPA Hardened, AES256, SHA256), DSP blocks, SRAM and eNVM, PCIe Gen 2, hardware-based 667 Mbps DDR2/3 controllers, and a 7 mW (typical) standby power consumption. Because of the available eNVM, the system can be completely powered down with-

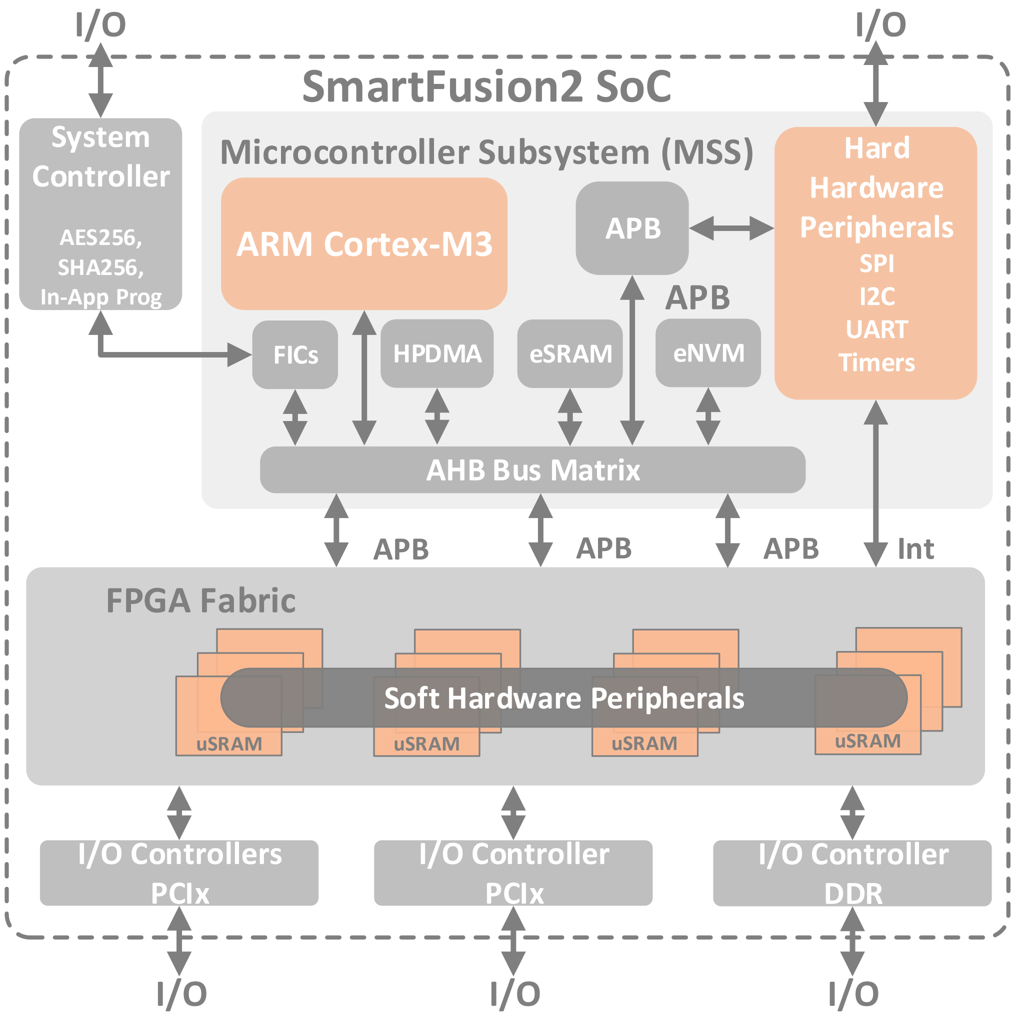


Figure 2.4: SmartFusion2 SoC FPGA architecture.

out losing its configuration or data. This SoC FPGA-based solution is a great choice for implementing hardware accelerators on the available FPGA fabric, integrating them with the MSS through standard Advanced Microcontroller Bus Architecture (AMBA) communication buses. The available standard communication hard-peripherals, such as SPI, Inter-Integrated Circuit (I^2C) and UART, make possible to connect any IEEE 802.15.4 compliant radio transceiver. The detailed SoC block diagram is depicted in Figure 2.4.

2.5 Test and Evaluation Tools

For the testing and evaluating purposes, two main tool suites are used: (1) Thread-Metric Benchmark suite [152], used for evaluating the OS performance achievements and the (2) Libero SoC Design Suite, for designing with the SmartFusion2. This last includes the ModelSim verification tool, used to simulate hardware modules before they are deployed inFPGA.

Regarding the Thread-Metric, it consists of a freely available benchmark suite, used for measuring any real-time operating system (RTOS) performance. Targeting RTOS services, e.g., cooperative and preemptive context switching and interrupt processing with and without preemption, it can be used, in the context of this thesis, for evaluating the embedded OS performance by measuring how much the system load is alleviated by offloading software-based tasks to dedicated hardware accelerators. For the Contiki-OS, and despite supporting preemptive and cooperative modes, preemptive scheduling is not yet implemented for the ARM Cortex-M3 architecture, thus, only the cooperative context switching benchmark can be supported. Regarding RIOT, since it is an RT OS, more benchmarks can be performed.

Libero SoC Design Suite (design flow depicted by Figure 2.5) offers high productivity with its comprehensive, easy-to-learn, easy-to-adopt development tools for designing with Microsemi's IGLOO2, SmartFusion2, RTG4, SmartFusion, IGLOO, ProASIC3 and Fusion families. The suite integrates industry standard Synopsys Synplify Pro synthesis and Mentor Graphics ModelSim simulation with best-in-class constraints management, Programming & Debug Tools capabilities, and secure production programming support. The ModelSim ME HDL Simulator [153], included in the Microsemi's development suite, is a tool for verifying and simulate hardware description language (HDL) code, line by line. It can perform simulations at all levels, i.e., behavioral (pre-synthesis), structural (post-synthesis), and back-annotated

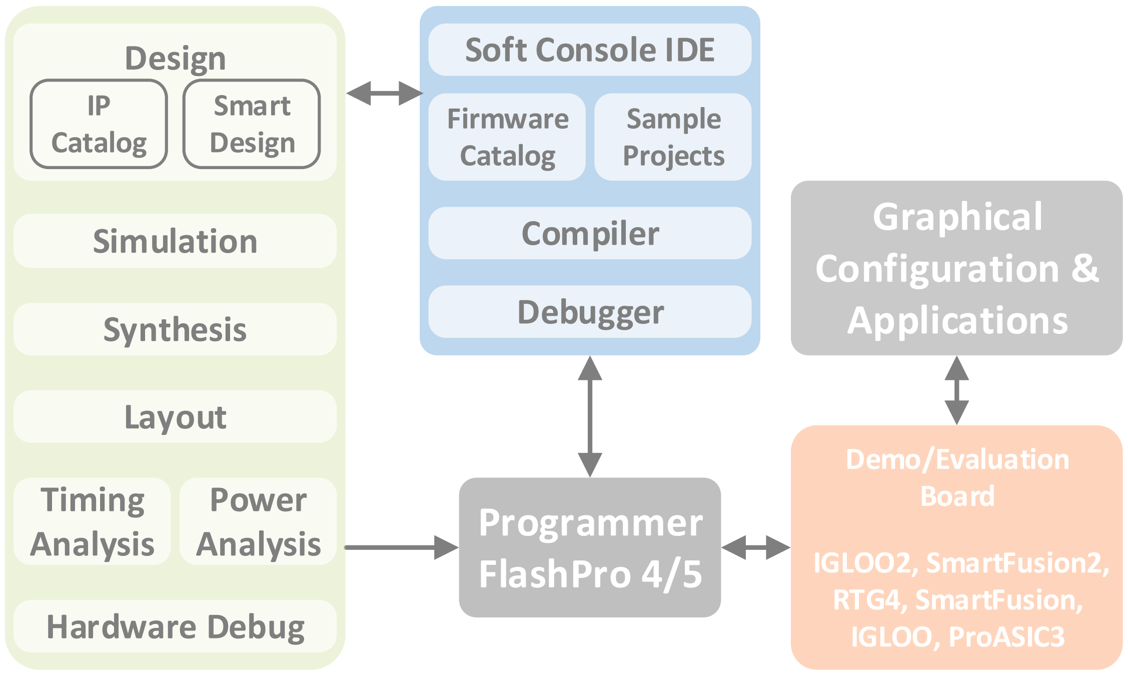


Figure 2.5: Libero SoC Design Suite design flow.

dynamic simulation. ModelSim ME fully supports current VHSIC Hardware Description Language (VHDL) and Verilog language standards, allowing to simulate behavioral, RTL, and gate level code separately or simultaneously. ModelSim supports all Microsemi FPGA libraries and it can simulate AMBA protocols through available simulation scripts. This feature is quite useful when developing peripherals connected to the MCU via such standard bus protocols, before the FPGA deployment.

2.6 Conclusions

This chapter presented and discussed the development platform and tools required to fulfill all the requirements when deploying and testing the proposed research performed throughout this thesis. The Microsemi's SmartFusion2 SoC hardware platform, which combines fabric FPGA with an ARM Cortex-M3 CPU attached with an IEEE 802.15.4 compliant radio transceiver, enables the development of a heterogeneous architecture for IoT low-end devices. As it will be presented in the next chapter, such platform provides the flexibility for exploring customized and generic-purpose hardware accelerators to be deployed under resource-constrained endpoint devices, while keeping low-budget requirements. The next chapter will also present the selection of the embedded OS that best suits the hardware platform.

Chapter 3

Heterogeneous Architectures for Low-end IoT Devices

The ubiquitous connectivity of IoT low-end devices brings new challenges over traditional WSN. Such challenges require not only security- and privacy-related features, but also solutions to handle the ever-growing amount of data that is transferred over the network. Ideally, low-end devices would be able perform complex tasks, with low energy consumption. However, performing such heavy processing tasks on these resource-constrained devices is not straightforward. The need for energy-efficient devices, while preserving their performance and security capabilities, requires new solutions at the architectural level of the wireless device.

This chapter describes the challenges of designing a heterogeneous architecture that targets low-end and resource-constrained devices, which combines a hard-core MCU and a RCU beside an IEEE 802.15.4 radio transceiver. The MCU hosts an embedded OS with an IoT-enabled network stack, as specified by the IoT-ARM [32], exploring the available FPGA technology to perform the RCU and to deploy customized sensing- and network-related accelerators, offloading heavy and/or complex software tasks to dedicated hardware blocks. After a brief introduction given by Section 3.1, the remaining of this chapter is organized as follows: Section 3.2 proposes a heterogeneous architecture for IoT low-end devices and details the design choices that led to its adoption. Section 3.3 discusses implementation of the CUTE mote, described as a *CUstomizable and Trustable End-device for the Internet of Things*, which is based on the proposed architecture. It also presents an embedded OS to be included in the heterogeneous architecture and the associated porting and integrity efforts. Moreover, an evaluation on the best software candidates that could be deployed on the RCU is shown, where software components, processes and network-related tasks of the OS are analyzed. Section 3.4 evaluates and characterizes the proposed offloading candidates in terms of software-based computational resources, while Section 3.5 evaluates the proposed mote and the architecture in terms of hardware resources and energy consumption for different power modes. Finally, Section 3.6 concludes this chapter.

Related Publications

Ideas and findings presented in this chapter resulted in the following publications:

- **T. Gomes**, S. Pinto, T. Gomes, A. Tavares and J. Cabral, "*Towards an FPGA-based edge device for the Internet of Things*," 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 2015, pp. 1-4.
- **T. Gomes**, F. Salgado, A. Tavares and J. Cabral, "*CUTE Mote, A CUstomizable and Trustable End-device for the Internet of Things,*" in **IEEE Sensors Journal**, vol.PP, no.99, pp.1-1.

3.1 Introduction

The ubiquitous connectivity of IoT networks brings many challenges when developing and designing wirelessly connected devices [6]. This technological trend to shift from traditional WSN to IoT-enabled environments requires not only security- and privacy-related mechanisms, but also solutions to handle the ever-growing amount of data that is transferred over the network [4, 5, 45]. When compared to traditional WSN, whose network architectures are mainly centralized and most of the times isolated from the Internet, bringing such connectivity and subsequent interoperability to the wireless nodes, as specified by the IoT-ARM network stack, is not straightforward. Although the IEEE 802.15.4 standard performing well in terms of energy, due to its low-power transmissions and low data-rate links, it was not initially conceived bearing in mind the IoT connectivity paradigm, which requires more data to be transferred. Ideally, the low-end devices would be able to perform complex tasks with the lowest energy consumption. Typical strategies consist in reducing the amount of data transmitted over the network, both in terms of the packet size (which can also benefit from data compression mechanisms) and the overall number of required transmissions. In addition, reducing the node's duty cycle and disabling the radio transceiver when not in use, contributes for a better energy usage and an extended node's lifetime. Trading-off the overall energy consumption with the performance and security capabilities of the low-budget wireless devices is quite challenging, as these requirements usually point into opposite directions.

As seen in Section 1.4 of Chapter 1, recent solutions aim to tackle the performance and power issues at the architectural level of the wireless mote. Such solutions not only integrate a traditional MCU beside an IEEE 802.15.4 radio transceiver but also rely on FPGA technology to assist the complex processing tasks. As also seen, an architecture based on the combination of an MCU, an FPGA and a radio transceiver, can play a key role in the future of sensor networks, in fields where processing capabilities such as strong cryptographic algorithms, data aggregation, data compression, among others, are major requirements. One of the biggest problems previously pointed to FPGA technology was the high power consumption, which was always one of the main constrains in traditional WSN solutions. However, in the recent years, FPGA technology has been thoroughly addressing the energy consumption problem, turning this technology into a great alternative for developing customized systems applied to wireless sensing applications [103]. Low-power optimized FPGA-based solutions are now able to enhance the computation of several types of algorithms in terms of speed and energy consumption, when compared with COTS microcontroller-based solutions for sensor nodes. Power consumption issues related to the use of FPGA technology applied to WSN can be found in [154]. Based on the principle that a radio transceiver can be completely turned off when it is not in use, a full implementation of the PHY layer in hardware takes great advantages of the Physical Layer Power Conservation (FLPC) principle. Other techniques such as DVS applied to FPGA-based sensor nodes can help in reducing the overall power consumption.

Following the approach of studied solutions and considering the previously identified requirements, this chapter contributes to the state-of-the-art with: (1) a reconfigurable heterogeneous architecture for IoT low-end devices; (2) the integration of the heterogeneous architecture in a customizable and trustable end-device (CUTE) mote that targets IoT applications; (3) the integration of an embedded OS, with a standardized communication stack, hosted by the MCU, that can benefit from hardware acceleration; (4) a deep analysis on software-based network-related tasks to be offloaded to hardware accelerators, aimed to integrate the RCU; (5) a performance, energy and hardware resource evaluation of the proposed heterogeneous architecture over the CUTE mote.

3.2 Heterogeneous Architectures

Designing heterogeneous motes for low-end wireless devices requires, at least, three main components: (1) a radio transceiver, for handling and supporting data communication; (2) an RCU unit, used to perform dedicated hardware-assisted tasks and deploy the acceleration parts; (3) an MCU, responsible to handle the software-based tasks. When developing such systems, several aspects must be considered, such as the interface between components (as well as the interaction between them) and the respective functionalities performed by each one. These aspects must be taken in consideration during the design phase following a hardware/software co-design approach, focusing mainly on energy and performance trade-offs between software- and hardware-based components.

Figure 3.1 depicts several design alternatives for deploying a heterogeneous architecture on the hardware platform selected in Chapter 2. In all architectural options, there are connected sensors (S) and available memory (M) that can be used by the MCU and/or the RCU, when needed. Typically, FPGA solutions do not include

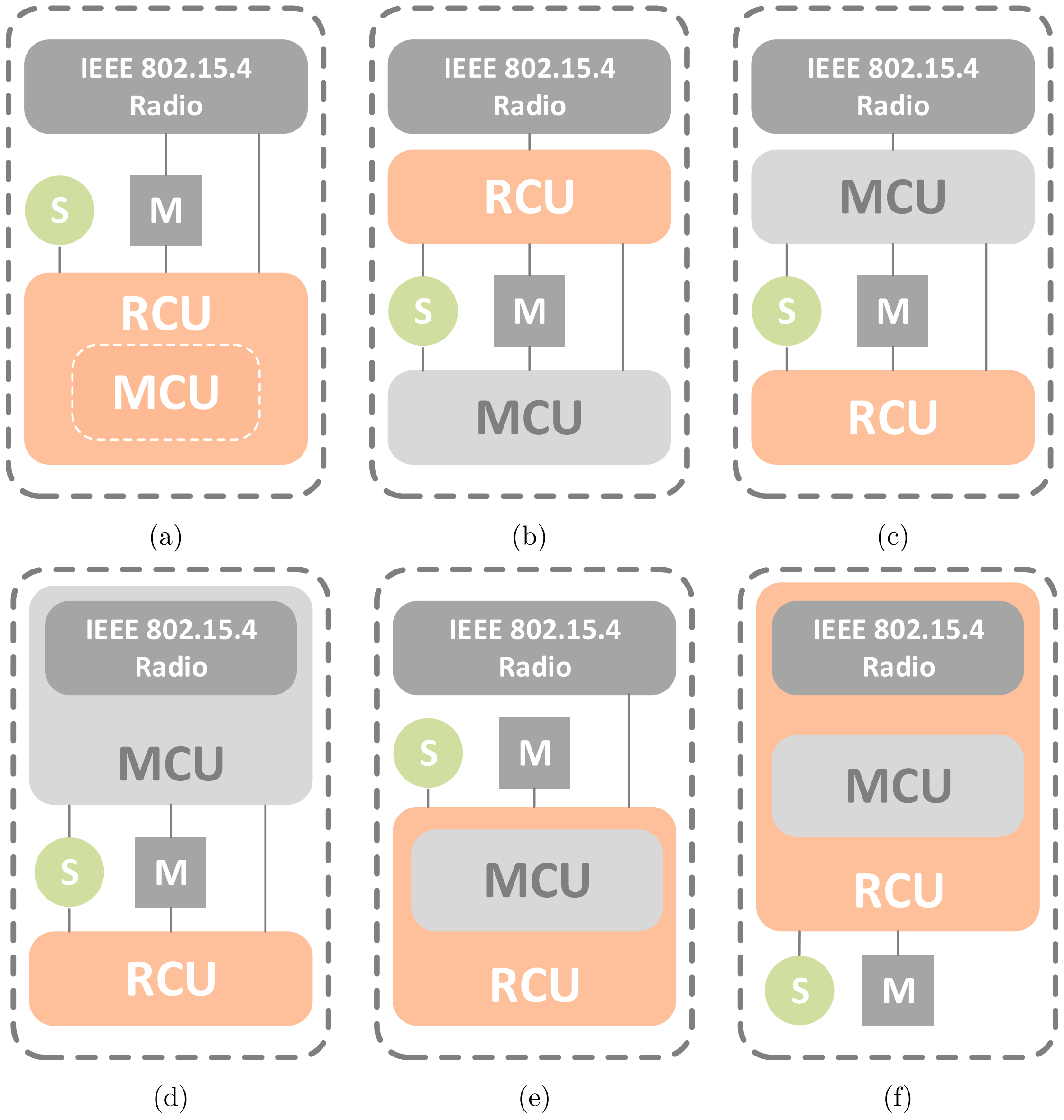


Figure 3.1: Design alternatives for the heterogeneous architecture.

a radio transceiver to provide wireless connectivity and such component must be attached (standalone or combined with other systems) to the architecture. Figure 3.1a corresponds to the design alternative that only includes the RCU attached to an IEEE 802.15.4 radio transceiver, adopted by [107]. Since the radio mainly handles the PHY and the MAC layers (which correspond to the wireless transmission, RF modulations schemes and MAC Data frames), higher layers and corresponding tasks must be deployed and performed by a soft-core MCU deployed on the RCU. Since the IoT network stack comprises several layers with quite complex algorithms and protocols, this would require a big amount of available programmable logic and a long development time. Deploying a soft-core MCU on the RCU, as performed and suggested by [108,109], should be also avoided since the MCU typically requires also a considerable amount of available programmable logic to be deployed. This architecture also requires the RCU to be active all the time in order to make the MCU available to run the software-based application that performs the IoT network stack operations, which would contribute to a higher energy consumption.

When a hard-core MCU is added to the architecture, the workload of the wireless node can be efficiently distributed over the available components and the heterogeneity of the architecture truly explored. In Figure 3.1b, the RCU is placed between the MCU and the IEEE 802.15.4 radio transceiver. This design decision is adopted by the PowWow [106] mote architecture. The MCU is based on the MSP430 microcontroller and runs the low-level radio tasks (performed and provided by Contiki-OS) along with other application-specific processes, while the RCU (based on a Igloo FPGA) accelerates a 32-bit cyclic redundancy check (CRC) algorithm, used for error detection in the data-link layer of the PowWow software. In order to save energy, the power saving schemes, such as the DVFS, are performed by the RCU. The design option adopted by the Cookie WSN [105] (Figure 3.1c), places the MCU between the RCU and the radio transceiver. This architecture relies on a wake-up radio (operating at 915 MHz frequency) connected to the RCU to wake-up the sensor node before activating other power consuming parts, e.g., the main IEEE 802.15.4 radio transceiver, when data exchange is required between two nodes. Low-power management control is performed by the RCU, which uses the wake-up signal from the attached wake-up radio. The MCU is used to serve the low-level radio and application tasks. The bottleneck of these two last architectural designs is the communication protocol needed between the RCU and the MCU that relies on GPIO pins available from the MCU, which not only reduces the available input/output (I/O) ports but it is also limited by the selected protocol bandwidth (e.g., SPI, I^{2}C).

HaLoMote [104] implements the architecture depicted by Figure 3.1d. This approach allows the node to temporarily shut down system components that are not in use, e.g., the radio transceiver that is tightly coupled to the MCU in a SoC, which can be easily suspended when not in use or during idle time. In the HaLoMote the inter-processor communication, i.e., data sharing between the RCU and the MCU, is done through a communication protocol that also uses GPIO pins. Because heavy computational tasks are performed by the RCU, only processed and aggregated data (to be transmitted over the IEEE 802.15.4 radio transceiver) are transferred between the RCU and the MCU using the memory attached to these blocks. Although providing an energy efficient heterogeneous WSN mote, all the sensing tasks are performed by the RCU, e.g., the processing algorithms and heavy computational loops, which are application-specific and dictated by the application demands, turning the overall solution not suitable for a wide range of solutions.

Figure 3.1e illustrates a different solution from the previous ones for implementing a heterogeneous architecture for the low-end devices. The RCU is coupled to the hard-core MCU on a single SoC solution and only the IEEE 802.15.4 radio transceiver is externally attached. This SoC design allows the RCU to deploy customizable hardware accelerators (available to the MCU as typical hardware peripherals) that can be accessed through standard communication buses, widely used by well-known MCU architectures. The DPM blocks are also added to the RCU to accomplish the low-power requirements of the heterogeneous architecture. When strategically placing the RCU between the MCU and the radio transceiver, all the wireless data can be intercepted and processed by network accelerators before reaching or leaving the MCU. This architectural design eases the handling of the low-level radio tasks, typically performed by the MCU on the aforementioned architectures, by the RCU, which connects the MCU by standard AMBA protocols. Since one of the main research questions of this thesis is to explore the utilization of accelerators for low-end IoT devices, this design choice is the one that allows special focus on the communication stack, while all the previous design options better fit application-specific tasks. The ideal solution would implement the alternative architecture illustrated by Figure 3.1f, which integrates every component on a single SoC. However, this solution is not yet provided by any known commercially available device.

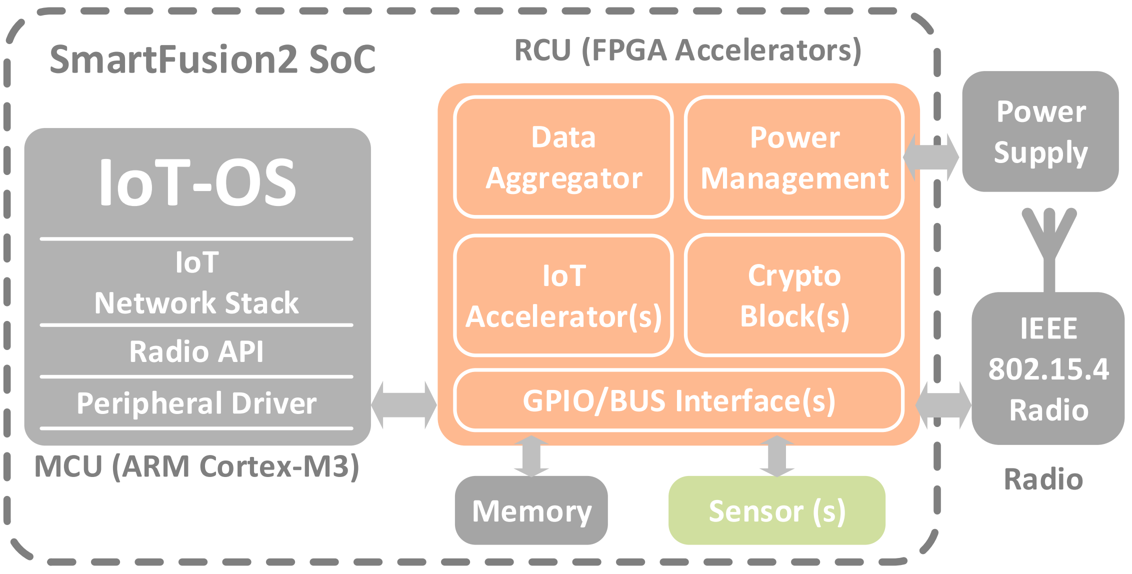


Figure 3.2: Heterogeneous Architecture for IoT low-end devices.

Figure 3.2 illustrates the proposed heterogeneous architecture based on design choices of Figure 3.1e and deployed on the SmartFusion2 SoC. The ARM Cortex-M3 MCU hosts the embedded OS with an embedded IoT-compliant network stack, which is responsible for handling low-level radio tasks and low-priority application processes. By its turn, the RCU integrates:

1. A DPM system, which is responsible to perform power saving schemes in order to save energy, i.e., disabling the radio device or acceleration blocks when not in use;
2. AMBA bus protocol standard interfaces for accessing different available memories and peripherals deployed on the FPGA;
3. Access to other external devices, such as sensors and the IEEE 802.15.4 radio transceiver though standard peripheral communication protocols;
4. A Data aggregation accelerator, used for data gathering and data compression before making it available to the OS;
5. Network-related accelerators for handling the message exchange between nodes over the network;
6. Security-related accelerators for handling the encryption/decryption mechanisms applied in data confidentiality, integrity and authenticity processes.

Since all the accelerators are available to the OS as standard peripherals (accessed by internal buses), the deployment of such components on the RCU and its integration with the OS becomes simple, through the utilization of low-level software device drivers for memory-mapped peripheral access.

3.2.1 Securing the Architecture

Cyber security is one of the major concern when developing connected devices on the network edge. Not only the functional requirements of the embedded applications must be fulfilled, but they also must be achieved in a secured way. The SmartFusion2 SoC provides security mechanisms, which start during silicon manufacturing process and continue through system development and deployment. Security is provided (in a layered approach) at tree levels: (1) Secure Hardware, (2) Design Security and (3) Data Security.

From the hardware point of view, the hardware and design security features comprise:

1. **Overbuilding & Cloning Prevention** system, that avoids any unauthorized reproduction of the developed system;
2. **Design Security** system, that prevents deployed IP cores to be copied or retrieved from the chip for reverse engineering purposes;
3. **DPA Protection Counter Side Channel Attacks** system, which helps in protecting on-chip security keys and configuration bit streams, and tamper detectors with respective countermeasures.

To help securing data and applications, Data Security mechanisms provide an extensive range of cryptographic hardware accelerators for secure algorithms (such as AES, SHA, HMAC, ECDH, ECDSA) and security protocols (such as IPsec, SSL, TLS, SSH). Since the *Root of Trust* starts from the silicon, all the provided security mechanisms and accelerators are as well trusted. Security keys are stored using a Physically Unclonable Function (PUF) mechanisms which makes them impossible to retrieve from the hardware. Moreover, the provided *Secure Boot* mechanism protects the start-up code for processors and MCUs from intentional attacks. In summary, this platform provides a robust solution for both Design Security, when protecting the hardware and design IP is critical, and Data Security, when protecting application data is necessary. Resorting such mechanisms, along with the proposed heterogeneous architecture, a customizable and trustable mote for low-end devices can be developed over the hardware platform.

3.3 CUTE Mote

For testing the proposed architecture on the hardware platform, the Customizable and Trustable End-device (CUTE) mote was developed. The CUTE mote, which deploys the heterogeneous architecture on the SmartFusion2 SoC, not only benefits from the flexibility to add customized accelerators (network- and application-specific) to the final system, but also from all the security features and services the hardware platform provides. Such features and services on the network edge device are mainly intended to be explored for Data Security and secure communication protocols, e.g., the IPsec. The root of trust provided by the SmartFusion2 SoC hardware platform eases the usage of this protocol as it provides built-in encryption

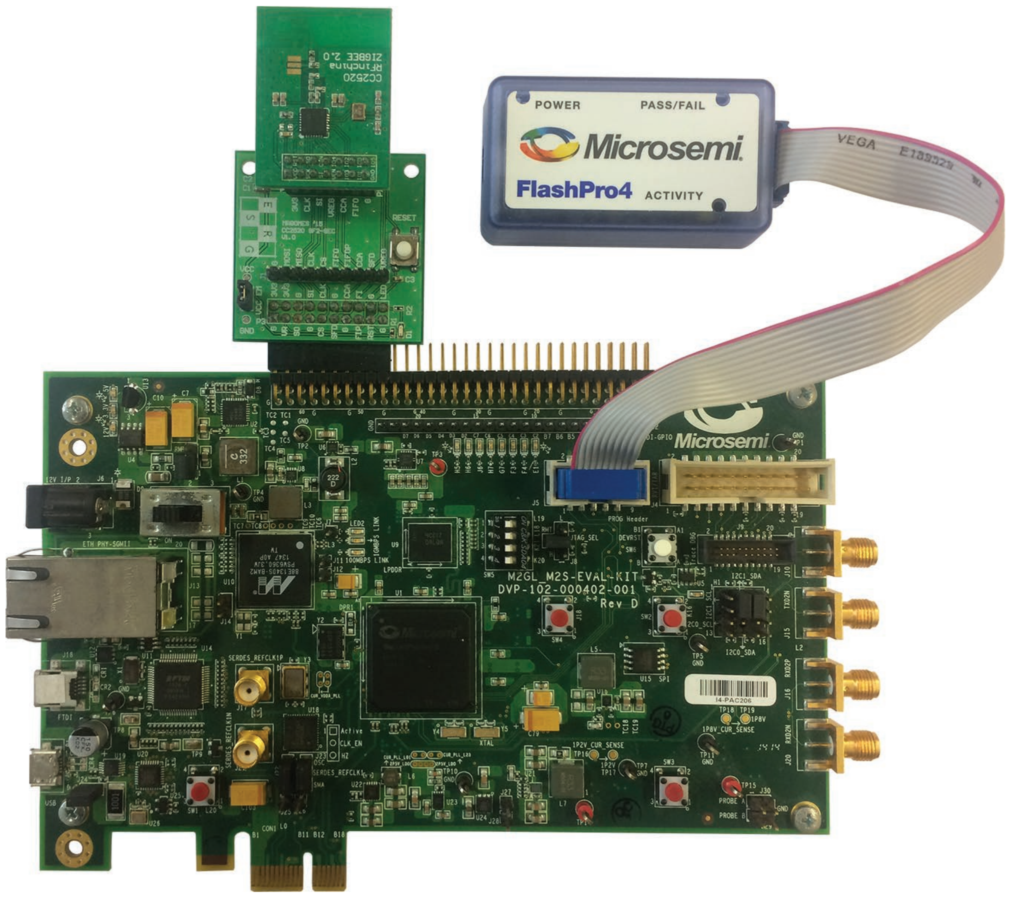


Figure 3.3: CUTE mote prototype.

keys that can be used by network nodes in a pre-shared mechanism or used in an Internet Key Exchange (IKE) protocol.

Figure 3.3 depicts the first prototype of the CUTE mote, with the CC2520 IEEE 802.15.4 radio transceiver attached. This device, integrated in the CUTE mote, endorses the hardware platform with the connectivity, interoperability and reachability capabilities. Other devices (e.g., CC2538 and CC2650), which are supported by the embedded OSes, will be used as comparison platforms to verify the OS behavior and validate the developed solution (architecture and accelerators). At this stage of development, the prototype only utilizes the energy provided by the platform, thus energy supply systems are not yet developed.

3.3.1 Adding an Embedded OS

As seen in Chapter 2, from the available embedded OSes [124] for low-end resource constrained devices, Contiki-OS and RIOT are the most suitable to run on the

CUTE mote, as they both provide support for the ARM Cortex-M3 architecture with a very small footprint IoT-enabled network stack. The Contiki-OS implements the μIP [155] while RIOT inherited the OpenWSN [156] stack. Since they mainly differ on their kernel implementation (monolithic with a cooperative scheduler for Contiki-OS, while RIOT implements a preemptive tickless microkernel), they cover a wide range of applications where both can be used, according to application requirements. Other embedded OSes would fit the selected platform, based both on monolithic and on microkernel implementations, as they also support the microcontroller architecture (ARM Cortex-M3) and provide the low size network stack. However, Contiki-OS and RIOT prime for their popularity in good on-line support and development communities. Contiki-OS has one main advantage over RIOT, it provides the Cooja simulator for testing any parameters or changes to the network topology and/or stack protocols. For this reason, the first choice goes towards Contiki-OS to run on the heterogeneous architecture of the CUTE mote.

Porting Contiki-OS

Despite of Contiki-OS supporting several hardware platforms and architectures, there is no direct support for the selected hardware platform, therefore a software porting is necessary. Although providing an easy development, Contiki-OS is set available as open-source and independent from any development tools, therefore, does not give support or integration to any specific integrated development environment (IDE) or framework. This hampers the setup of new development environments with code debugging support and the generation of the final binary file. Furthermore, using 3rd-parties software with hybrid platforms such as the SmartFusion2 is not straightforward, since it is necessary, when creating soft-hardware peripherals on the FPGA, to generate the correspondent SoC configuration files and respective firmware artifacts, both for the FPGA and the MSS. These configuration files include software libraries, drivers and different memory setups.

Since Microsemi provides its own Eclipse-based IDE with full debugging support (Microsemi SoftConsole IDE v3.4), the porting was carried out over the firmware provided and generated from the SoC designing tool suite (Libero SoC v11.7). The firmware artifacts include startup code for the ARM Cortex-M3 microcontroller, hardware abstraction layer (HAL) code, linker-script files, as well as drivers for the hardware peripherals added to the SoC configuration. On the folders added to the Contiki-OS directory structure, it was needed to provide an API to remap

OS system calls to the platform and CPU dependent libraries generated by the SoC designing tool. Support for external devices, such as the IEEE 802.15.4 radio transceiver, requires a different implementation and it will be later detailed. From the Libero tool, two directories are created: (1) `CONTIKI_MSS_CM3_app` folder, where the Contiki-OS source-code is placed and (2) `CONTIKI_MSS_CM3_hw_platform` folder, where the generated SoC firmware can be found.

Inside the Contiki-OS directory structure, there are two important folders (and their corresponding files), that must be created:

1. **`contiki/cpu/newcpu`**, where the microcontroller-dependent code and libraries must be placed:

    - **`Hardware libraries`**: The peripheral drivers, USB support, etc., remapped to the MSS libraries on the `CONTIKI_MSS_CM3_hw_platform` folder;

    - **`Clock`**: The Contiki-OS system clock driver. Remapped to the corresponding MSS libraries, which uses available timers to perform time-dependent operations;

    - **`rtimer`**: Contiki-OS requires a high-precision timer for timing-critical features. This implementation allocates one dedicated hardware timer;

    - **`Startup code, linker scripts and makefiles`**: In opposite to the original development style, these are now generated by the IDE and Libero tools, and later changed by the user according to the application requirements, e.g., stack configuration (IPv4 or IPv6) and main application behavior;

    - **`Watchdog`**: Watchdog timer library remapped to the MSS watchdog support, which is necessary to compile and properly run the Contiki-OS;

    - **`Interrupts`**: Corresponding handlers for the respective interrupt source, e.g., GPIO and UART;

    - **`Node MAC address & node ID`**: Since the platform has no built-in IEEE 802.15.4 MAC interface, it is necessary to generate a node MAC address and a node ID (both required by Contiki-OS) to set in the IEEE 802.15.4 radio. This number was retrieved from the unique board serial number, which is provided by the on-chip services. Since it is built-in on the hardware, it is unique for every SoC and can never be changed.

2. **`contiki/platform/newplatform`**, which contains platform specific code:

    - **`main.c`**: Platform dependent main file which initializes the entire system;
    - **`hardware platform peripherals`**: External peripherals, such as LEDs, are remapped to the corresponding MSS GPIO library;
    - **`Sensors`**: Software libraries used for push buttons and other connected sensors. Buttons are connected to the GPIO peripheral and are considered as sensors to Contiki-OS;
    - **`contiki-conf.h`** file: Consists of the platform dependent configuration file that contains all the system setup and all customized network stack parameters, such as the RF channel in use (`RF_CHANNEL`), the IEEE personal area network (PAN) identifier (`IEEE802154_CONF_PANID`), device routing support (`UIP_CONF_ROUTER`), etc.;
    - **`Radio`**: The low-level driver support to access and configure an IEEE 802.15.4 radio transceiver.

Although the multi-thread limitation for the ARM Cortex-M3 is no longer present in Contiki-OS, at this stage of development it was still not available, thus, it was not implemented.

Contiki's Radio Driver

The CC2520 provides an SPI interface for connecting the host MCU. Since the RCU interfaces the MCU and the radio transceiver, an SPI IP core provided by Microsemi was deployed on the RCU, instead of using the hard SPI peripheral present on the SoC. This design option will allow to deploy accelerators with network-related functions to execute on the RCU, which directly interfaces the radio via the provided SPI IP core. By its turn, the MCU interfaces the SPI IP core via the Advanced Peripheral Bus (APB) 3 protocol. Regarding the SoC configuration, which is exported from Libero SoC, it includes on the `CONTIKI_MSS_CM3_hw_platform` folder the `core_spi` folder, which contains the driver for accessing the deployed SPI core. The provided API is then remapped, as done with previous peripheral drivers such as the LEDs and push buttons, with the corresponding Contiki-OS system calls for accessing the IEEE 802.15.4 radio transceiver.

From now on, this setup is set as the native solution, which is all software-based, for the Microsemi SmartFusion2 SoC and the selected hardware platform. Future software analysis and comparisons with deployed accelerators will always be related to this setup. Further accelerators, which are meant to be deployed as customized hardware peripherals on the architecture's RCU, will also need their own device drivers to enable access to their data buffers and configuration registers. Such drivers will be later added to the Contiki-OS software stack. Ideally, if the accelerators only comprise network-related tasks, only modifications ot the radio driver are needed. This OS agnostic implementation allows an easy integration with other IoT-enabled embedded OSes, requiring only minimal modifications to the device drivers that interface the IEEE 802.15.4 radio. Due to the efforts required to port an OS to the selected hardware platform tool-chain, and since the results will take the same effect on any OS, RIOT was, at this time, left out of the research work of this thesis.

3.3.2 Adding Hardware Accelerators

Traditional sensor nodes that rely on FPGA (either based on a combination of microcontroller and FPGA or as a standalone platform) mainly take benefit from such technology on sensing-related tasks, like sensor data collecting, data aggregation and/or compression, and data protection. These solutions, by leveraging the available FPGA to enhance the computation of heavy tasks and algorithms, can offer even-more optimized sensors nodes in terms of speed and overall power consumption. However, their tasks are highly customized and usually application-specific, thus, not suitable to every connected node. Connecting such devices to the Internet, adds new requirements to the node, such as the communication stack (or part of it), needed to provide the Connectivity, Interoperability and the Reachability requirements. In resource-constrained devices, such a network stack introduces significant overhead in terms of energy consumption and MCU processing time. This is due to the nature of radio communications, where a device with an enabled radio receiver will be able to receive and decode transmissions from all devices in range that use the same standard and operate on the same channel, along with interference from other sources.

For every incoming frame, it is necessary to allocate resources for retrieving the frame from the radio transceiver and processing the data on each layer until it reaches its final destination. Reducing these overheads, by accelerating network related tasks can result in more efficient wireless nodes, and since the stack is standardized, such accelerators can be used by any connected node that relies on heterogeneous architectures. Because not everything is suitable to be hardware-accelerated, it is necessary to evaluate, following a hardware/software co-design approach, each layer of the communication stack, as there are features/tasks suitable to be offloaded to dedicated accelerators, while others might not. Such approach requires a deep analysis on the OS network stack (which can also benefit from a cross-layer design optimization), regarding standard operations and protocols. It also provides agnosticism to the solution, as the offloaded tasks are turned available in the form of soft-hardware peripherals, accessed through standard communication buses. This way other IoT-enabled OSes can take advantage of such features by only performing slight modifications to the IEEE 802.15.4 radio driver.

Following the network stack bottom-up, several offloading candidates can be identified. From the MAC to the application, going through the network and transport layers (6LoWPAN and TCP/IP), which may require security mechanisms such as IPsec for authentication, data encryption, and data integrity, the processing of an IEEE 802.15.4 Data frame comprises several operations and verification steps that can be simplified and better performed in hardware.

MAC Sub-layer

The IEEE 802.15.4 standard already specifies some tasks that are usually accelerated by compliant devices and which can be found in most of the available COTS radio transceivers, such as the CC2420, CC2530 and the CC2538 integrating radio. Such tasks comprise mainly the automatic acknowledgment (ACK) and filtering of the received frames, which rejects frames that are not of interest to the receiving device. The filtering task is performed at three levels. The first level is related to the frame integrity, which rejects malformed frames or with errors, mainly caused by low-quality links at the PHY layer. The second level corresponds to the promiscuous mode operation, which specifies that a device operating in this mode shall forward all the received frames (after passing the first filter) to the upper layers without applying further filtering rules. The Third-Level of filtering, takes action when the frame passes the first level and the devices operates in non-promiscuous mode. Such filtering comprises the following verifications:

- The *Frame Type* field shall not contain a reserved frame type;
- The *Frame Version* field shall not contain a reserved value;

- The destination PAN identifier shall match the *macPANId* or shall be the broadcast PAN identifier, `0xFFFF`;
- If a short destination address is being used, it shall match either the node's *macShortAddress* field or the broadcast address. Otherwise, if an extended destination address is used instead, it shall only match the *macExtendedAddress* value;
- If a Beacon frame is received, the source PAN identifier must match the *macPANId*, unless the *macPANId* is the broadcast PAN identifier (`0xFFFF`) and the frame must be accepted;
- If a data or command frame is received and only the source addressing fields are present, the frame must be only accepted if the device is the PAN coordinator and the PAN identifier matches the *macPANId* value;

If any of the filtering rules, at any filtering level, are not satisfied, the MAC sublayer shall discard the incoming frame without any additional processing. Otherwise, the frame must be accepted for further processing and considered valid. If the received frame is an ACK request, the MAC sublayer shall send an ACK frame. At last, if the valid frame corresponds to a data frame, the MAC sublayer must be delivered to the next higher layer.

Aiming to efficiently reduce some overhead from the MCU, such filtering features are full or partially implemented by some IEEE 802.15.4 radio transceivers, e.g., the CC2520, used in the architecture. However, since not all the transceivers provide filtering and processing hardware blocks for the IEEE 802.15.4 frames, by default the Contiki-OS still performs these verifications. Moreover, some other features are provided by the Contiki-OS and not yet presented in any radio transceiver, such as the verification and detection of multiple receptions of the same frame by a device.

Network Layer

Contrarily to the MAC layer, all the processing of a received IPv6 packet on the Network layer is purely performed in software. After the packet is retrieved and restored by the 6LoWPAN adaptation layer, such tasks mainly encompass the following procedure:

- The first packet inspection corresponds to IP address checking and packet header checksum. If the destination IP address of the incoming packet does not match any of the local IP addresses, or if the header checksum mismatches, the packet must be dropped;

- The second packet inspection relates to the IP fragment reassembly feature. If packet reassembly is in use, the incoming fragment is copied into the right place in the buffer and a bit map is used to keep track of which fragments have been received. When the packet is reassembled, the resulting IP packet is ready to be passed to the Transport layer. If all the fragments are not received within a specified time frame, the packet must be discarded. This mechanism avoids the system to run out of memory and prevents low-system availability to store and process other incoming IP packets;
- If the packet is a broadcast or multicast packet, it shall be handled accordingly. Currently, the Contiki-OS μIP stack supports receiving broadcast packets as well as sending multicast messages for handling vital network protocols, such as the multicast Domain Name System (mDNS). Joining multicast groups and receiving non-local multicast packets is, however, not currently supported;
- If the packet corresponds to an ICMP version 6 message, which main use is the echo mechanism implemented by the `PING` application, a reply message must be sent to the requesting node. Such feature is implemented by Contiki-OS in a very simply manner, by mainly swapping the source and destination IPv6 addresses and rewriting the ICMP header with the Echo-Reply message type, forwarding it next back to the IEEE 802.15.4 radio interface.

Further protocol processing, such as the RPL, is still performed at this layer of the stack, however the received packets must pass through these previous verifications. Due to complexity reasons, the RPL protocol handling is left to be performed by the Contiki-OS, thus it is considered to be out of the scope of this thesis.

As previously mentioned, the IPv6 support on low-end devices is provided by the 6LoWPAN protocol, which defines an adaptation layer to enable IPv6 packets to be carried by IEEE 802.15.4 data frames. However, retrieving IPv6 packets from IEEE 802.15.4 data frames is not straightforward and requires additional processing as most of the fields must be retrieved and calculated following a set of rules provided by the robust IPv6 header compression mechanism. Such compression is needed mainly due to the Maximum Transmission Unit (MTU) size limitation for IPv6 packets over IEEE 802.15.4 data frames, which corresponds to 1280 octets. Hence, far from the maximum PHY layer packet size of 127 octets specified by the IEEE 802.15.4 standard. With the added headers needed by the IEEE 802.15.4 Data frames and also additional features such as Link-layer security, the available payload to carry an IPv6 packet is now reduced, varying from 102 to 81 octets. This leads to packet fragmentation and reassembly requirements, aided by the Internet Protocol Header Compression (IPHC) mechanism. For this purpose, an IPHC compression/decompression engine is needed and also implemented by the Contiki-OS. Such mechanism is supported by the HC1, HC0 and IPv6 standard specifications [51]. When a packet is required to be sent, it is compressed at its maximum level, according to the network topology and settings, as well as the protocol in use. When a packet is received, it must be decompressed and all the IPv6 fields retrieved. The above mentioned features can be good candidates to be offloaded from the Contiki-OS to dedicated hardware accelerators. Thus, it is required to evaluate them in terms of performance and load that can be alleviated from the MCU.

Transport Layer

In the Contiki-OS μIP stack, the Transport layer, represented by the TCP and UDP, is tightly coupled to the Network layer for optimization purposes. Moreover, because most of the services require a pair of remote/local ports to bind connections between the local and remote hosts, such verifications perform better at this level of the stack. For instance, in a simple UDP connection, after the packet being transferred from the Network layer, the paired local/remote ports must be verified and if the received packet (which specifies an UDP payload) does not match the local port, it must be dropped. That is to say, if a valid IPv6 packet is received with all matching IP addresses and protocols in use, it can still be dropped if there are no active connections on the listening ports between the host and remote nodes. Such situation occurs when unintentionally malformed packets are received by a node or in a DoS [54] attack scenario. Such verifications can also be evaluated as they represent a good candidate to be deployed in hardware. As TCP encompasses a more complex process (e.g., flow and congestion control), it is left to be performed by the Contiki-OS and only the port verification process is going to be analyzed.

Security-dependent Features

When the network requires security features, such as data encryption, security mechanisms with strong cryptographic algorithms must be implemented. In the current state-of-art, IPsec is the standard method to secure communications over the Internet. Despite being a viable option [157], on its original form cannot be deployed in resource-constrained devices, mainly due to its heavy processing requirements. A Lightweight version of IPsec [158], was proposed and deployed under the Contiki-OS network security layer. According to its authors, despite performing well in low-end devices, such mechanism can still be highly improved when supported by hardware. This is due to the security algorithms, such as the Advanced Encryption Standard (AES), which requires heavy computations. A hardware-based implementation of such features are way faster than pure software implementations. According to [159], when processing 512-byte datagrams over a single hop, the overhead of pure software authentication header is 65 % which decreases to 12 % with the help of a cryptographic co-processor.

Application Layer

Because the use of a standardized communication stack also specifies application protocols to be used, such as the CoAP and MQTT, they can also be fully or partially extended to hardware-assisted blocks. This is however, out of the scope of this thesis.

Qualitative Analysis

Table 3.1: Qualitative analysis between heterogeneous solutions.

	HaloMote [104]	***Cookies WSN*** [105]	***PowWow*** [106]	***CUTE mote***
Application Support	bare metal	bare metal	Contiki-OS	**Contiki-OS**
Real-Time Support	No	No	No	**U/D**
IoT-ARM Stack	No	No	Yes	**Yes**
Radio IEEE 802.15.4	AT256FRTR2	CC2420	CC2420	**CC2520**
MCU Arch.	8-bit AVR	TI MSP430	TI MSP430	**32-bit ARMv7-M**
Acceleration	App-specific	App-specific, crypto-engines	App-specific	**App-specific, crypto-engines**
Net. Acceleration	MAC filter	MAC filter	MAC filter, ARQ & FEC	**MAC filter, ARQ & FEC**
Advanced Net. Acceleration	No	No	No	**MAC, IPv6, IPsec, TCP/UDP**
Device Security	No	No	No	**Yes**
Data Security	N/P	ECDSA, SH-1, MD5	N/P	**Yes** [1]

[1]Both algorithms and protocols: AES, SHA, HMAC, ECDH, ECDSA, IPsec, SSL, TLS, SSH.

Table 3.1 summarizes a qualitative analysis between the heterogeneous architectures previously identified and discussed in Chapter 1 and the CUTE mote, displaying their differences on the same features. In this analysis the architectural options that include a soft-core MCU, i.e., the contributions given by *Vera-Salas et al.* [107], *Nyländen et al.* [108] and *Stelte* [109] have been discarded, as they are not suitable for the desired solution. Concerning the RT support on the CUTE mote, it is currently under development (U/D) as the RIOT is being prepared to be also supported, beside the Contiki-OS. In regard to the other parameters, the CUTE mote clearly highlights its superiority.

3.4 Contiki-OS Evaluation

In this preliminary phase, it was evaluated the impact of the verification and processing of a received IEEE 802.15.4 Data frame and the subsequent payload containing a 6LoWPAN frame, used to retrieved the corresponding IPv6 packet. The performed experiments mainly concern the time taken by the software-based tasks and their impact on the overall system load. That is to say, the system availability when performing such tasks in the most important network-related operations, such as the reception of an IPv6 packet and further accepting/rejection at the Network and Transport layers, according to the standard requirements such as the address filtering and IPv6 packet compression.

3.4.1 API Evaluation

For the API valuation, it was measured the execution time taken from reading a received IEEE 802.15.4 Mac Protocol Data Unit (MPDU) from the radio transceiver until it is delivered to the Network layer. Next, the same experiment was repeated for the Network layer, measuring the execution time taken from reading a IEEE 802.15.4 MPDU from the MAC layer, until is delivered to the Network layer. Such execution time also includes the Data frame processing by the 6LoWPAN adaptation layer, needed to retrieve the IPv6 packet from the IEEE 802.15.4 Data frame. The overhead due to the OS was also taken in consideration. It consists of: (1) interrupting the OS execution to register a new event; (2) run pending OS processes; (3) run the process responsible to transfer the packet from the radio RX buffer to the MAC and Network layers. The performance evaluation was calculated based on the average time of receiving 1000 valid packets from a trusted sender, with a packet sending rate (PSR) of 8 packet per second (pkt/s).

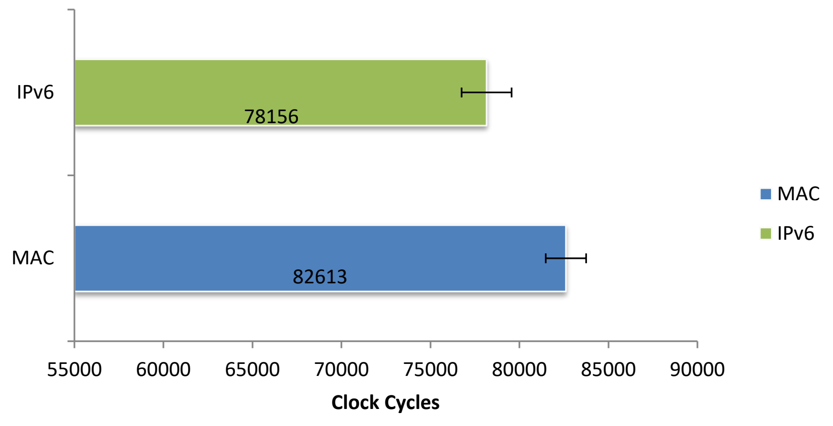


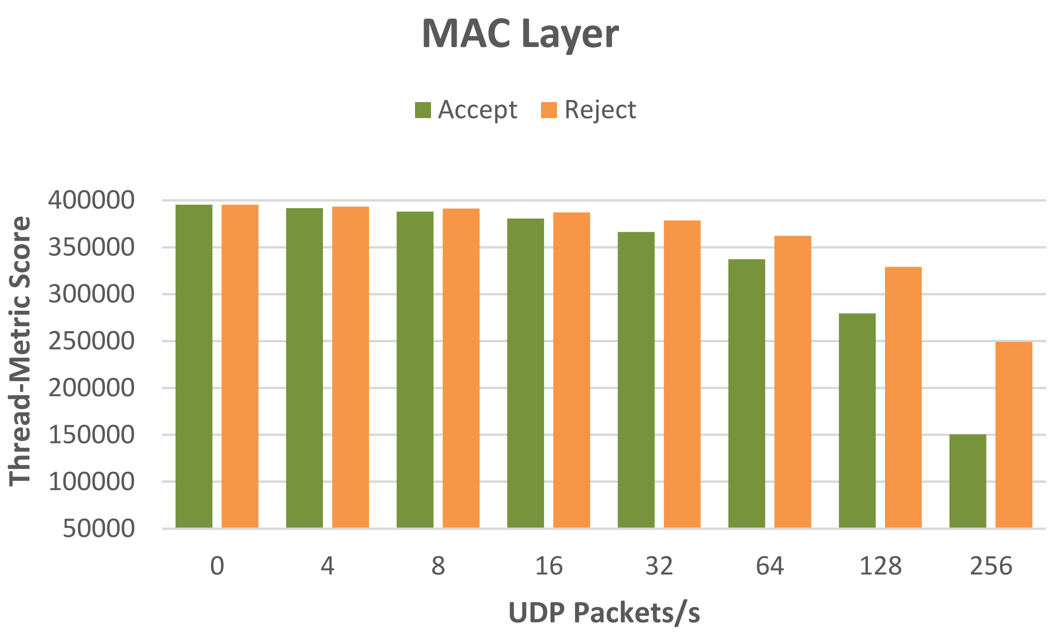
Figure 3.4: MAC and IPv6 API evaluation.

From the given results, depicted in Figure 3.4, the number of clock cycles needed to process and filter one received MPDU is, on average, 82613. On the other hand, to process an IPv6 packet the number of needed clock cycles is, on average, 78156. Such results can be possibly improved if the Contiki-OS could resort hardware accelerators to help in performing the packet filtering tasks and further processing of the received IEEE 802.15.4 data frames (as performed at the radio transceiver level), as well as subsequent IPv6 packet extraction, processing and verification.

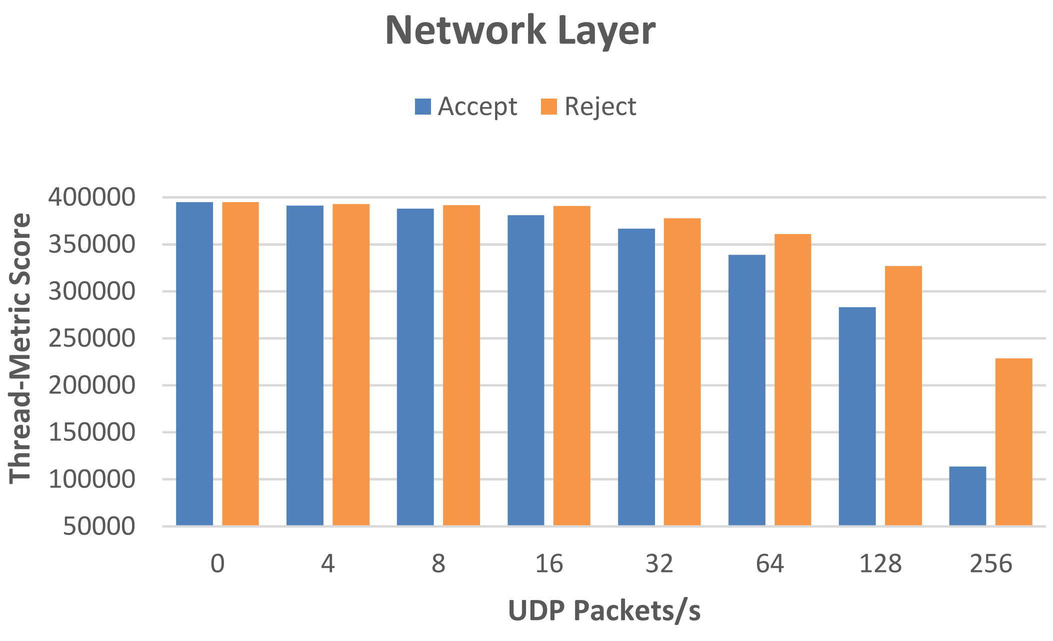
3.4.2 Thread-Metric Evaluation

This test consists in running the cooperative context switching benchmark from the Thread-Metric Benchmark Suite [152] in order to evaluate how the software-based filtering and processing tasks, at MAC and Network layers of the communication stack, impact the normal OS execution. This benchmark consists of creating five Contiki-OS processes, which output a score value which can represent the systems availability. A higher score denotes more system availability to execute application tasks, rather than OS services. On the other hand, a lower score represents less availability as the central processing unit (CPU) is busy processing scheduled OS services/events.

Figure 3.5a and Figure 3.5b depict the benchmark result for the filtering and processing tasks of an IEEE 802.15.4 data frame (and the resulting IPv6 packet) for the accepting and rejecting situations. This means, when a packet is received and



(a) Software-based packet processing: MAC Layer.



(b) Software-based packet processing: Network Layer.

Figure 3.5: Thread-Metric evaluation.

intended to the CUTE mote, it is processed and accepted to be delivered to upper layers, otherwise it is discarded. The tests were performed with a simple network setup consisting of one node exchanging data with the CUTE mote through a simple UDP connection. Data was sent at different PSR values (4, 8, 16, 32, 64, 128 and 256 pkt/s), both for the accepting and rejecting situations. In order to cause the packet reject situation, the destination IP was set to be a different one, rather than the CUTE mote address. A real-case network topology, with more than only two nodes, was implemented and better explained later in Chapter 4.

For each received packet, the OS must trigger and register new events into the scheduler. Since the Contiki-OS implements an event-based kernel, the scheduler polls every time all the active system processes, running only those which associated events were triggered, thus, increasing the number of events, the system's availability is expected to decrease. Also, enabling new events to be processed by the OS will decrease the system's predictability, as running the new scheduled processes responsible to read and filter any invalid MPDU will delay the execution of other system processes.

From the obtained results, for both analyzed layers, it can be concluded that for a PSR of 0 pkt/s, the score is at its highest level, as there are no packets being received by the node. Increasing the PSR the score tends to decrease, since the OS is busy attending the active system processes to handle the incoming MPDU frames and the corresponding IPv6 packets. The received IPv6 packets intended to another destination are meant to be discarded, thus, the benchmark score is higher than the score when the packets are to be accepted and processed by upper layers. Offloading such tasks to the RCU, would not only improve the score when the packets are meant to be accepted (taking benefit from the hardware acceleration), but the benchmark score is expected to be at its highest level when the packets are meant to be rejected. This is because the OS will not create new processes for the newly arrived IEEE 802.15.4 frames, as they are processed in hardware. Despite both graphics having the same look, the score is even lower when the processing is needed at the Network layer. This is mainly due to the overhead introduced by the 6LoWPAN adaptation layer and the heavy task performed by the IPHC mechanism.

3.5 CUTE Mote Power Characterization

Microsemi's flash-based architectures offer a very low-power solution for low duty cycle applications, with zero in-rush power (during power up) and very low power operations due to the Flash Freeze technology on the SmartFusion2 SoC family. This enables the CUTE mote to be used in typical very low-power applications, benefiting from all the advantages the deployed heterogeneous architecture can provide. The Flash Freeze power mode consists of an ultra low-power standby mode that sets the

FPGA fabric in a low-power quiescent state in which the state of the internal large static random-access memory (LSRAM), μRAM, and flip-flops are preserved. Entry into Flash Freeze typically occurs in less than 100 μs and exiting from this mode to I/O's operational typically occurs in less than 200 μs. This can be simply done by using a system service API call to the system controller and/or using any external trigger.

Enabling this technology into the FPGA fabric, combined with the ARM Cortex-M3 low-power modes, results in a great design choice for heterogeneous low-power motes that can directly compete with typical COTS solutions based on homogeneous WSN sensor nodes. The Flash Freeze control can be integrated into the CUTE mote and Contiki-OS in applications that only operate reactively or periodically. This can be done in the following ways:

1. Integration of wake-up radios in the heterogeneous architecture;
2. Activation of radio duty cycling protocols in Contiki-OS;
3. Take benefit from the Contiki's low-power mode operation.

Wake-up Radios

The design option of integrating wake-up radios into the architecture, such as used by the CookieWSN [105], can be applied to trigger the Flash Freeze control module. This way the system keeps operating into a very low power state while there are no communications or tasks needed to be performed by the node. When a node desires to communicate, the wake-up radio will enable the regular operation mode again.

Radio Duty Cycling Protocols

Low-power radio hardware is not always enough to fulfill the low power requirements of some applications. Aiming to promote energy saving modes, Contiki-OS provides several duty cycling mechanisms that can be used to turn off the IEEE 802.15.4 radio transceiver when not in use. The purpose of a power-saving duty cycling protocol is to keep the radio disabled, while providing enough *rendezvous* points for two nodes to be able to communicate with each other. Such *rendezvous* points in a duty cycling protocol can be achieved in several ways, but usually a time-synchronized mode is preferred.

In Contiki-OS, such RDC protocols can be found at the MAC layer of the stack, more precisely in a sub-layer called RDC layer. By default Contiki-OS provides three duty cycling MAC protocols: ContikiMAC [137], X-MAC [138] and LPP [160]. ContikiMAC is a low-power listening-based protocol, such as the B-MAC [161], but with better power efficiency. Contiki's X-MAC is based on the original X-MAC protocol, but with a significant set of improvements and extensions which contributes with reduced power consumption and helps in maintain good network conditions. Contiki's Low-Power Probing (LPP) is based on the original LPP protocol but with enhancements that improve power consumption, as well as provide mechanisms for sending broadcast data.

Aiming to save energy, the RDC layer tackles only the radio transceiver, controlling its utilization in the wireless network. In most of IoT applications, the device only enters into regular operation modes when the radio needs to send or receive messages. Therefore, the overall energy consumption on the mote can be reduced if during the non-operational modes the device enters into the Flash Freeze mode, going periodically to the operation mode during the radio active time according to the RDC protocol in use. Despite of Contiki-OS providing such energy saving mechanisms at the radio level, for the testing purposes of the deployed accelerators, the RDC layer is disabled and the CSMA/CA mode (in the radio transceiver) is used instead.

Contiki-OS Low-power Operation Modes

Contiki-OS provides an API to enable low-power modes when all pending OS events are serviced. This is done by calling `lpm_enter()` on every scheduler run. Low-power modes are dependent from the MCU architecture and the device's platform, thus, it must be implemented into Contiki-OS accordingly. When such modes are activated, the OS goes to the lowest supported power mode. In the CUTE mote hardware platform, such mode corresponds to the Flash Freeze sleep mode. Waking-up the system from this mode is done by resorting a sleep timer, which is internally controlled by the Contiki-OS according to the scheduler behavior.

Power Characterization

Table 3.2 summarizes the power characterization of the CUTE mote with two main low-power modes, active mode and Flash Freeze mode, running at 32 MHz. When running in normal operation the power consumption is, on average, 56.52 mW and

Table 3.2: CUTE mote hardware characterization.

Parameter	***Value***	
SoC	SmartFusion2 SoC FPGA 90K LE	
MCU Architecture	32-bit	
Instruction Set	Cortex-M3	
Clock (MHz)	32	
V_{CC} (V)	1.2	
$P_{FlashFreeze}$ (mW)	8.23	
P_{active} (mW)	56.52	
IPv6 packet soft-processing	**Accept**	**Reject**
$t_{average}$ (μs)	254	292
$P_{average}$ (mW)	56.52	56.52
$E_{average}$ (nJ)	14356	16503

when the Flash Freeze mode is activated this value drops to 8.23 mW. Also, the energy needed to process one IPv6 packet in software is on average 14356 nJ and 16503 nJ for the accepting and rejecting situations, respectively. After offloading such tasks to hardware the energy consumption is expected to decrease. Since the duration of the active and Flash Freeze modes depends on the mote configuration and application requirements, the overall energy consumption cannot be directly calculated. Such task can only be performed after the CUTE mote being used in a real application scenario and the network requirements well defined.

3.6 Conclusions

This chapter proposed and described a heterogeneous architecture for endpoint devices in the IoT edge network. The architecture combined with the SmartFusion2 hardware platform forms the CUTE mote, which is described as a customizable and trustable end-device, specially tailored for low-power IoT applications. The architecture explores its heterogeneity by combining an MCU, which runs the Contiki-OS, and an RCU attached to an IEEE 802.15.4 radio transceiver. The RCU allows the development of accelerators for the network stack which can be used by the Contiki-OS for energy saving and performance purposes.

Despite these evaluations are important to characterize the CUTE mote and test the feasibility of the solution, the main focus of this thesis concerns the evaluation of accelerators that can be securely deployed on the RCU of the proposed architecture, which can benefit from network-related accelerators at two levels: (1) by accelerating complex network-related tasks; (2) subsequent reduced system load by offloading software-based tasks to the accelerators. Since the main processing time concerns the processing of IEEE 802.15.4 Data frames and subsequent IPv6 packet processing, the evaluations are to be performed when the CUTE mote is in active mode, thus the mote will not enter low-power modes, such as the Flash Freeze. Security-related requirements are provided by the hardware platform, which complies with all the concerns previously discussed in Chapter 1. In the next chapters, the evaluated tasks and processes are deployed and tested on the RCU. This will help to understand the benefits of offloading selected tasks to hardware by comparing their performance with the software-based approach.

Chapter 4

Accelerating the MAC Sub-layer

The 6LoWPAN enables traditional WSN devices to be IP-connected, allowing IPv6 packets to be sent and received over IEEE 802.15.4-based networks. However, bringing IPv6 connectivity to low-end wireless sensor nodes leads to considerable device resources utilization, e.g., CPU and energy, caused by the increased amount of data transferred over the network that needs to be handled. In order to tackle and reduce these overheads, this chapter proposes the MAC layer accelerator (MLA), an IEEE 802.15.4 hardware accelerator to be deployed on the heterogeneous architecture presented in the Chapter 3, which mainly targets endpoint devices in the IoT network. Specially tailored for handling the MAC Data frames, the MLA implements the *First-*, *Second-* and *Third-level of filtering*, as specified by the IEEE 802.15.4 standard, performing all the processing and filtering of a received MPDU before it reaches the OS network stack. The MLA also supports extra features, such as the detection of multiple receptions of the same frame.

This chapter is structured as follows: Section 4.1 gives a brief overview over the IEEE 802.15.4 standard and its evolution over the years, while Section 4.2 presents the protocol Data frame, which is the main targeted component of the MLA. Afterwards, the MLA and its internal modules and functionalities are presented and explained in Section 4.3. The achieved results presented in Section 4.4, obtained from a typical IoT network scenario, show the benefits of including such an accelerator on the RCU of the heterogeneous architecture, offering nearly 17% of overhead reduction. All filtering functionalities are executed by the accelerator in order to discard unneeded frames, which avoids unnecessary interrupts to the OS and increases the system availability up to 59%. Finally, Section 4.5, concludes this chapter.

Related Publications

Ideas and findings presented in this chapter resulted in the following publication:

- **T. Gomes**, S. Pinto, F. Salgado, A. Tavares and J. Cabral, "*Building IEEE 802.15.4 Accelerators for Heterogeneous Wireless Sensor Nodes*," in **IEEE Sensors Letters**, vol. 1, no. 1, pp. 1.4, Feb. 2017.

4.1 Introduction

Maintained by the IEEE 802.15 working group [162], the IEEE 802.15.4 standard specifies the PHY and the MAC layers for LR-WPAN. With a high maturity level and widely used by technologies like ZigBee-1.0 [163] and -2006 [164], it has become the de facto standard for the MAC layer. It is also adopted by the IoT-ARM communication stack and other IoT reference architectures, providing wireless connectivity and promoting interoperability for IoT low-end devices in the edge network.

Since it was defined and released in 2003, the IEEE 802.15.4 standard [165] has been continually facing revisions and improvements over the years, always with the goal to provide a framework to the communication protocols of emerging wireless technologies, both at the MAC and PHY layers. Aiming to embrace a wide range of application scenarios, a broad number of MAC protocols [166] have been proposed. Regarding the way they access the medium, such protocols can be classified into two different categories:

1. **Contention-Based protocols**, like carrier-sense multiple access (CSMA) and Multiple Access with Collision Avoidance (MACA), where nodes contend for the channel in various ways in order to acquire the channel and transmit data;
2. **Contention-Free MAC Protocols**, like the time division multiple access (TDMA) and code division multiple access (CDMA), where nodes preallocate transmission resources in the network.

Contention-Based protocols are characterized for having simple implementations, however, they are weak in terms of scalability, mainly caused by the increase number of collisions when the number of participating nodes increases. On the other hand, Contention-Free protocols provide good scalability as they eliminate the issue of collisions by slotting the channel resources. However, they are more complex in terms of protocol setup and implementation. These improvements and features at the MAC layer only define different ways by which the shared medium is accessed and controlled. Contiki-OS, as seen in Chapter 2 and Chapter 3, supports and provides MAC protocols in these two categories.

On the recently updated version of the IEEE 802.15.4-2015 standard [167], the Time Synchronized Channel Hopping (TSCH) protocol (maintained by the IEEE 802.15.4e working group) was added to the PHY layer. It uses a channel-hopping mechanism that aims to tackle multi-path fading and external interference problems. Initially developed and commercialize by Dust Networks [168], this protocol, called Time Synchronized Mesh Protocol (TSMP) [169] is now part of the standard and mainly targets the challenges brought by IoT-based networks, such as the high volumes of exchanged data and the large number of participating devices. In turn, the TSCH originated the late 6top Protocol (6P) [170], which enables distributed scheduling over 6TiSCH (IPv6 over the TSCH mode of IEEE 802.15.4e) networks, showing that it is possible to provide both high reliability and energy-efficiency to the low-power smart devices. These improvements only tackle the PHY layer.

Commercial Off-The-Shelf Solutions

From available COTS solutions, several low-power IEEE 802.15.4 compliant radio systems already implement hardware functionalities that aim to improve the overall system performance of the network node. Implementing both the PHY and MAC features, these solutions enable low-power connectivity among smart IoT devices. The CC2520 [171], an IEEE 802.15.4 -2006 compliant radio transceiver from Texas Instruments, available standalone or included on the latest SoC devices with single or multi-standard wireless support, implements a very low-power PHY layer with an ultra-low current consumption on the transmitting and receiving modes. Concerning the MAC layer, it implements the *Microcontroller Support system*, which integrates critical portions of the RX requirements from the IEEE 802.15.4-2003 and -2006 standard, in dedicated hardware acceleration blocks. This approach reduces the MCU interruption rate, simplifies the software that handles the frame reception, and provides the results with minimum latency. These features include a *Frame filtering and processing engine*, which performs two main tasks:

1. **Processing of received frames**, which enables several PHY features such as the detection and removal of the received PHY synchronization header and the automatic frame check sequence (FCS) verification. The first is composed by the preamble and the start frame delimiter (SFD), while the latter attaches the retrieved Received Signal Strength Indicator (RSSI) and Link Quality Indication (LQI) fields to the received frame. At the MAC layer, the implemented features encompass the MAC Header (MHR) processing, that is able to filter and process incoming frames, e.g., by checking the frame version field, frame type, PAN ID, Source and Destination addresses, rejecting the ones intended to another destination, as specified in the IEEE 802.15.4-2006 revision, Section 7.5.6.2, Third-level of filtering [172].

2. **Automatic transmission of the ACK frame**, which consists in automatically sending the ACK frame to a sender when a valid frame is received and an ACK is requested.

Other well-known devices, such as the AT86RF231 [173] from Microchip, provide full IEEE 802.15.4-2003 and -2006 support and incorporate a *MAC Hardware Accelerator*, which performs automated acknowledgments, implement the CSMA/CA mechanism and can execute packet retransmissions. The AT86RF231 also includes an automated FCS verification mechanism to extract the packet integrity and to retrieve the RSSI and LQI field values. For the MHR handling, similarly the CC2520, it contains an *Automatic Address Filtering* block that is able to perform basic filtering tasks such as the PAN ID, Frame Type and Destination Address.

Other COTS solutions include, on the same chip, not only the IEEE 802.15.4 radio transceiver but also an MCU. These SoC solutions are mostly used for low-end devices due to their low-power capabilities and the low-budget characteristics. The EM357 [174] from SiliconLabs and the CC2538 [175] from TI are also available IEEE 802.15.4-compliant SoC solutions that include on the same chip the radio transceiver and a 32-bit ARM Cortex-M3. Moreover, other hardware accelerators are included on the same SoC, or provided separately, such as cryptographic blocks for securing communications and maintain data integrity [176,177]. As these algorithms perform in hardware, they provide fast execution of heavy cryptographic tasks. Table 4.1 summarizes, in terms of the power characteristics and device type, some commercially IEEE 802.15.4 compliant transceivers with MAC hardware blocks available in the market and widely used in traditional WSN and IoT solutions.

Table 4.1: IEEE 802.15.4 compliant devices with acceleration engines.

Vendor	***Product***	***RX Sensitivity***	***TX Current @ 0 dBm***	***RX Current***	***Type***
Texas Instr.	CC2520	-98 dBm	25.8 mA	18.8 mA	radio only
	CC2538	-97 dBm	24.0 mA	20.0 mA	SoC
Atmel	AT86RF231	-101 dBm	14.0 mA	12.4 mA	radio only
SiliconLabs	EM357	-100 dBm	27.5 mA	25.0 mA	SoC

FPGA-based Solutions

Recent software-defined radio (SDR) solutions, consisting of radio communication systems where components that have been typically implemented in hardware are now defined by means of software, have also being contributing with FPGA-based implementations of the IEEE 802.15.4 standard. In [178] it is proposed a sensor gateway with a reconfigurable PHY layer that makes it possible to switch, from IEEE 802.15.4 to IEEE 802.11 links, accordingly to the required speed and network parameters. Also targeting SDR systems, another FPGA-based open-source implementation has been proposed in [179]. However, it only deploys the PHY layer of the IEEE 802.15.4 in the FPGA.

Other FPGA implementations, for the MAC layer, consist in accelerating the data transmission by reducing the data frame size by using robust data compression techniques performed by dedicated hardware blocks [180]. More solutions, based on reconfigurable sensing hardware blocks with dedicated data gathering and compression systems, can be found in recent literature [107, 108], already discussed in previous chapters. However, and despite all this range of solutions, they fail in providing a reconfigurable hardware solution that combines the filtering scheme specified by the standard, along with other features related to the Data frame type of the IEEE 802.15.4 protocol, for heterogeneous architectures used in IoT endpoint devices.

Despite all the improvements at MAC and PHY layers, provided by their respective Internet Engineering Task Force (IETF) working groups and other relevant research, these contributions do not tackle the way data is exchanged and formatted, which is independent from any MAC and PHY protocol in use. From available COTS solutions, and focusing on the present hardware filtering capabilities, although they provide all the three levels of filtering (as specified by the standard), they fail into bring more filtering options, such as the support for multiple PAN and more Source and Destination MAC addresses. Moreover, deploying such standard on a heterogeneous architecture makes possible to add new capabilities, such as the detection of the reception of multiple frames mechanism, which is not yet implemented in any available COTS solution and only provided by the OS network stack.

This chapter describes an IEEE 802.15.4 hardware accelerator, which is implemented and deployed on the heterogeneous architecture presented in Chapter 3. It performs the tasks previously identified and proposed to be offloaded to hardware and, considering the best contribution of this chapter, it serves as a foundation for deploying upper layer stack features that rely on the IEEE 802.15.4 standard such as the 6LoWPAN protocol. Such task would be impossible to perform by only using the IEEE 802.15.4 features provided by COTS solutions. This implementation mainly focus on the filtering tasks since other features, e.g., the automated ACK system, are already offered by the radio transceiver and further improvements are not achievable.

4.2 IEEE 802.15.4 Data Frame

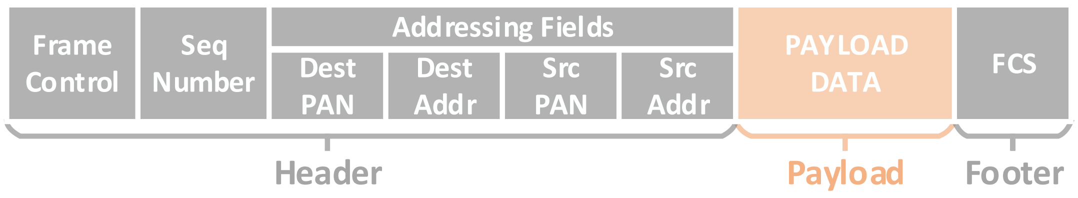


Figure 4.1: General IEEE 802.15.4 Data frame type.

Figure 4.1 depicts an IEEE 802.15.4 Data frame, which will gather special focus in this chapter. The other frame types, i.e., ACK, MAC Command and Beacon, will not be covered. The Data frame type is composed by three main sections: the MHR, the MAC payload and the MAC Footer (MFR). All of these sections contain at least one parameter field.

1. **MHR**:
    - Frame Control Field (FCF): It contains information about the MAC Data frame, which defines the type, addressing fields, and other control flags.
    - Sequence Number (SN): The SN field specifies the sequence identifier, e.g., Beacon Sequence Number (BSN) and Data Sequence Number (DSN) of the frame.
    - Addressing Fields: These contain the information about the Destination and/or Source PAN address, as well as the Destination and Source MAC addresses.
2. **MAC Payload**: The data payload is variable and can use up to 127 bytes per frame, according to the application requirements or (other) in-line protocols.
3. **MFR**: Contains the FCS, which follows the last MAC Payload byte. The FCS is calculated over the MPDU, using a 16-bit CRC algorithm.

The FCF, the SN and the FCS are required to be part of every frame whereas the addressing fields, the auxiliary security header and the MAC Payload might not be included in a frame type. Using the IEEE 802.15.4 Data frame illustrated by Figure 4.2 as an example (taken from a message exchange between two nodes in the

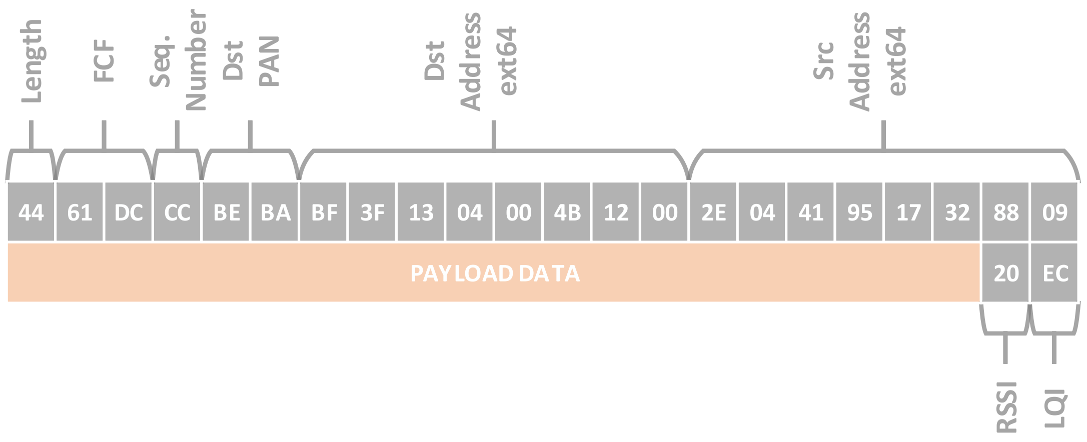


Figure 4.2: Example of an IEEE 802.15.4 Data frame.

same wireless personal area network (WPAN)), the MHR and the MFR sections can be analyzed and its corresponding data retrieved. The FCF field value is `0xDC61` (further analyzed in Section 4.2.1) and the SN field value is `0xCC`. For the Addressing fields, the following information is carried on this section of the MHR:

- **Destination PAN Address**: Indicates the address of the destination PAN, to where the MAC Destination address belongs. This field is always present, whereas the frame should reach the same PAN or not. In this example, the PAN address was set to `0xBABE`;

- **Destination Address (ext64 mode)**: Specifies the MAC Destination address the frame must reach. Its representation is determined by the FCF and, in this example, corresponds to the address `00:12:4B:00:04:13:3F:BF` (64-bit). When the frame is intended for broadcasting messages to the nodes belonging to the same WPAN, a 16-bit version (short mode) is used instead, with the address value `0xFFFF`;

- **Source Address (ext64 mode)**: Corresponds to the MAC Source address that has sent the frame. Its representation is also determined by the FCF and, in this example, is `09:88:32:17:95:41:04:2E` (64-bit).

From the MFR it can be extracted the RSSI and the LQI values:

- **RSSI**: This field is part of the MFR section and indicates the power present on the received radio frame, measured by the analog front-end of the radio transceiver. This value can be used for several purposes, including distance estimation between two nodes and localization schemes. However, its value can vary greatly due to the wireless nature of an LR-WPAN and this should always be taken into consideration;

- **LQI**: Also part of the MFR section, it is a metric that represents the quality of the communication link and it is computed based on the received RSSI and the number of errors detected on the link.

4.2.1 IEEE 802.15.4 Header Fields

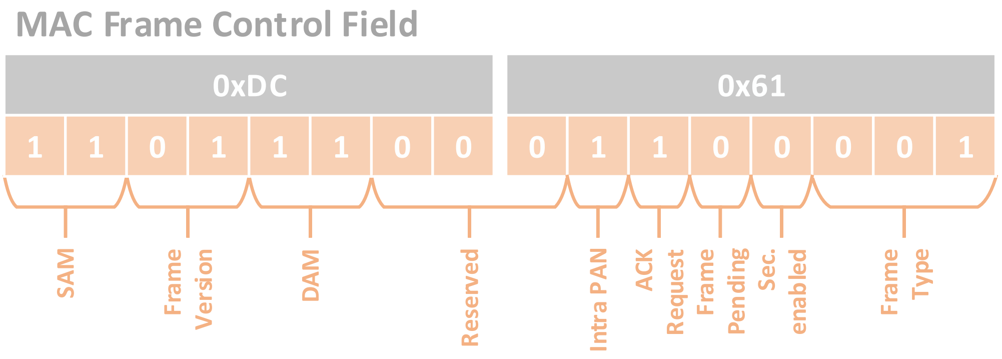


Figure 4.3: IEEE 802.15.4 Frame Control Field encoding.

The FCF, Figure 4.3, dictates how the MAC frame is formatted and how its information must be derived:

- **Frame Type**: Indicates the type of the received frame. It mainly uses four types: Beacon (`0b000`), Data (`0b001`), Acknowledgment (`0b010`), and MAC Command (`0b011`).

- **Security Enabled**: This bit indicates if the MAC sublayer is using security. If so, the Auxiliary Security Header (ASH) header must also be present.

- **Frame Pending**: Indicates if the sender node has more data pending for the destination node.

- **ACK Request**: Indicates if the sender requires an ACK.

- **PAN ID Compression**: Indicates if only one PAN identifier field has to be present. This happens when both sender and receiver are on the same WPAN, and thus, same source and destination PAN address.

- **Destination Address Mode**:
  - `00`: Both PAN and Address fields are not present.
  - `01`: Reserved.
  - `10`: Destination Address field contains a short address (16-bit).
  - `11`: Destination Address field contains an extended address (64-bit).
- **Frame Version**: Refers to the version of the IEEE 802.15.4 protocol in use.
- **Source Address Mode**:
  - `00`: Both PAN and Address fields are not present.
  - `01`: Reserved.
  - `10`: Source Address field contains a short address (16-bit).
  - `11`: Source Address field contains an extended address (64-bit).

Analyzing the given fields, and following the selected software-based tasks identified and analyzed on the Chapter 3, the accelerator will focus on the frame type, frame version, the SN and the Addressing fields, which will enable the deployment of features provided by the MLA.

4.3 MAC Sub-layer Accelerator (MLA)

Figure 4.4 depicts the heterogeneous architecture, which now integrates the network accelerator with IEEE 802.15.4-standard features. The accelerator performs previous software-based tasks, selected from the Contiki-OS and offloaded to hardware. Traditionally, after a received IEEE 802.15.4 Data frame being delivered to the network stack, the software processing of an MPDU consists in verifying the MAC Header, which contains on the FCF, Figure 4.3, the information relative to the received frame. Subsequently, the Addressing fields have to be processed in order to verify, for instance, if the received frame matches the receiver's PAN address, the Destination address or if a broadcast message was received. The SN field is also checked and used to avoid multiple processing when several receptions of the same frame occurs, caused by multi-path or when it is (un)intentionally sent by a source in the network. This verification allows the OS to accept or discard the re-

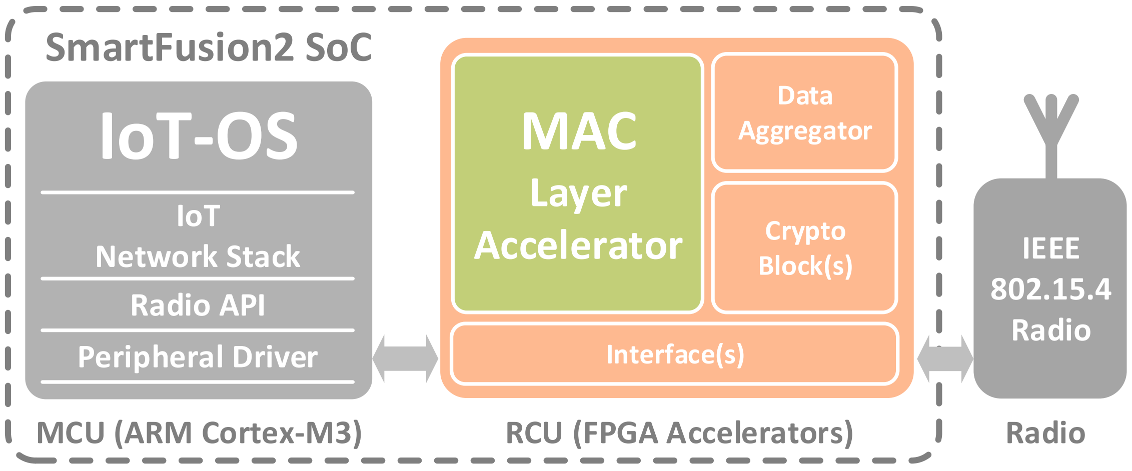


Figure 4.4: Adding the MLA to the heterogeneous architecture.

ceived frames at the MAC layer level, before they are forwarded to the upper layers, avoiding worthless further processing.

Despite the hardware support provided by the radio devices, these tasks are still performed in software by Contiki-OS, whereas the packet is meant to be accepted or discarded. For deploying and testing the MLA, software features such as the PAN and source/destination address verification, a frame version checker, and a Duplicate Frame Detector (DFD) module were developed and deployed on the RCU of the heterogeneous architecture.

The MLA connects to the MCU via the AMBA APB3 interface and the IEEE 802.15.4 radio transceiver using an SPI bus. It can be configured at runtime as it allows the OS to dynamically add/remove addresses to/from the comparing registers, as well as to define the number of packets to be held by the duplicate MPDU detector. When the radio transceiver holds a new MPDU on the RX buffer, the MLA is responsible to transfer it to its own RX buffer in order to be processed. Thereafter, the MLA will accept or discard the received frame according to the configured rules and matching addresses on the respective address registers. Also, if the MPDU passes the filtering processing it can still be discarded if the DFD module signals multiple receptions of the same frame. The MLA only interrupts the OS if the MPDU is accepted to be forward to the network stack. Hence, unnecessary event triggering and subsequent processing by the corresponding OS processes (e.g., Radio process and TCP/UDP process) is avoided, preventing also undesired wake-up calls to the MCU when it is power saving modes. For statistical purposes, the MLA also stores the number of dropped/accepted packets, as well as the number of detected duplicated packets.

4.3.1 General Architecture

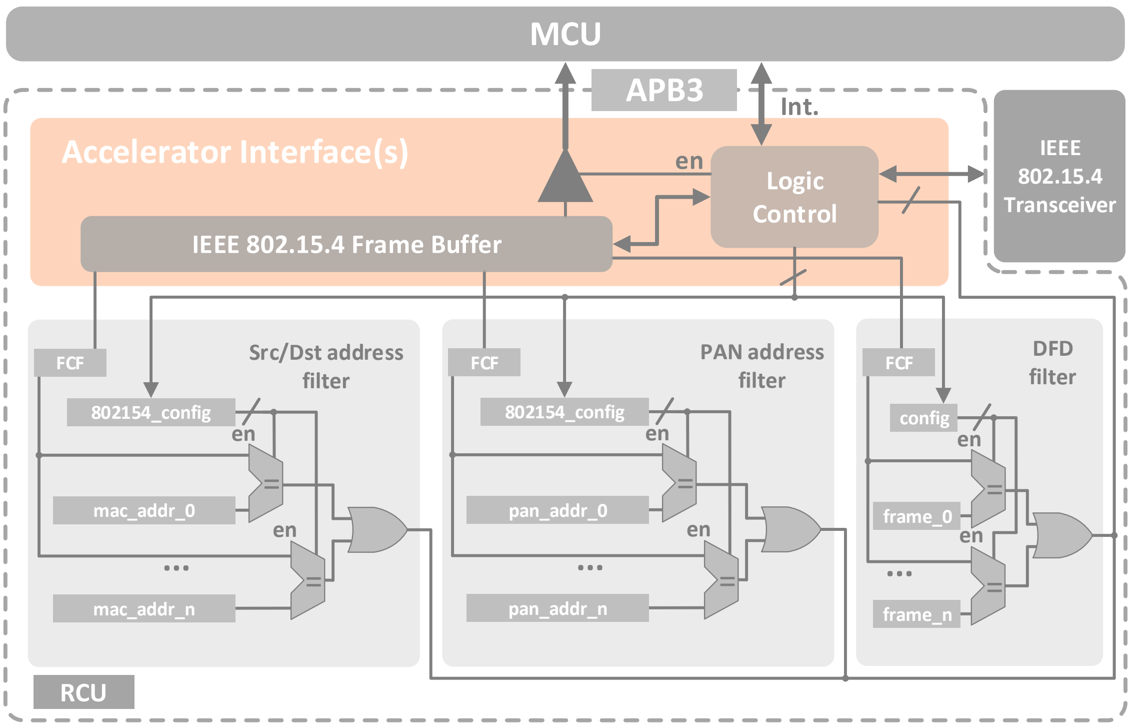


Figure 4.5: RTL design of the MLA in the heterogeneous architecture.

The RTL design of the MLA is depicted in Figure 4.5 and it is mainly composed by the following modules: (1) PAN Filter module; (2) Destination and Source Address Filter modules (only represented once, but deployed separately); (3) the DFD module. These modules are further explained in detail in the next sections.

4.3.2 Deployed Filtering Modules

PAN Filter Module

The MLA implements a PAN filter which verifies the PAN field of the received packet and compares its value to a set of addresses stored on a preconfigured address list. If the received value matches any from the list, the PAN Filter module signalizes it to the logic control module. More than one comparing address can be added to the list, which can be useful for the inter-PAN routing or when a (router) device has more than one network interface on different WPAN.

Destination and Source Address Filter Modules

A Destination and Source Address Filter module was also deployed on the MLA. It analyzes the packet's MAC Destination and Source address fields and compares them with a set of addresses stored in a list of addresses. If the fields match any address according to their source/destination values, an output is triggered to the logic control module, flagging the matching addresses. Then, accordingly to the configuration, the MPDU is discarded or accepted.

Duplicate Frame Detector Module

The DFD functionality checks the SN field value and verifies if this value, combined with the MAC source address, was detected on previous received packets. This means that, if a Data frame passes other filtering rules, it can still be discarded if this module detects the multiple reception of the same frame. This is done by checking the Sequence number and the source address of the received Data frame and comparing it with (a configurable number of) the last received frames.

4.3.3 Peripheral Interface

The deployed accelerator is a memory-mapped peripheral, interfaced with the MCU through the internal APB3 bus. Therefore, it requires a memory base address, matching the target hardware design, from where the registers of the accelerator instance are initialized and accessed. Figure 4.6 depicts the address space for the deployed accelerator and respective registers, while Listing 4.1 includes part of the file `core_accelerator_regs.h`, which contains the register definitions required for accessing the peripheral. The `NETWORK_ACC_BASE_ADDR` defines the peripheral base address, which in this case corresponds to `0x30000000`, matching the SmartFusion2 M2S090TS SoC hardware design. The `FILTER_REG`, corresponds to the peripheral configuration register, which is used to enable or disable the peripheral, as well as the desired filtering features. If all bits are set to zero, `FILTER_REG` is set to `FILTER_DISABLE`, the peripheral is disabled and bypasses all data exchange between the MCU and the CC2520. Otherwise, the peripheral is enabled and each bit field has the following definition (as shown in Figure 4.7): `PAN_ADDR` enables/disables the peripheral to filter received packets by the PAN ID; `SRC_ADDR` enables/disables the peripheral to filter the received IEEE 802.15.4 Data frames according to the MAC

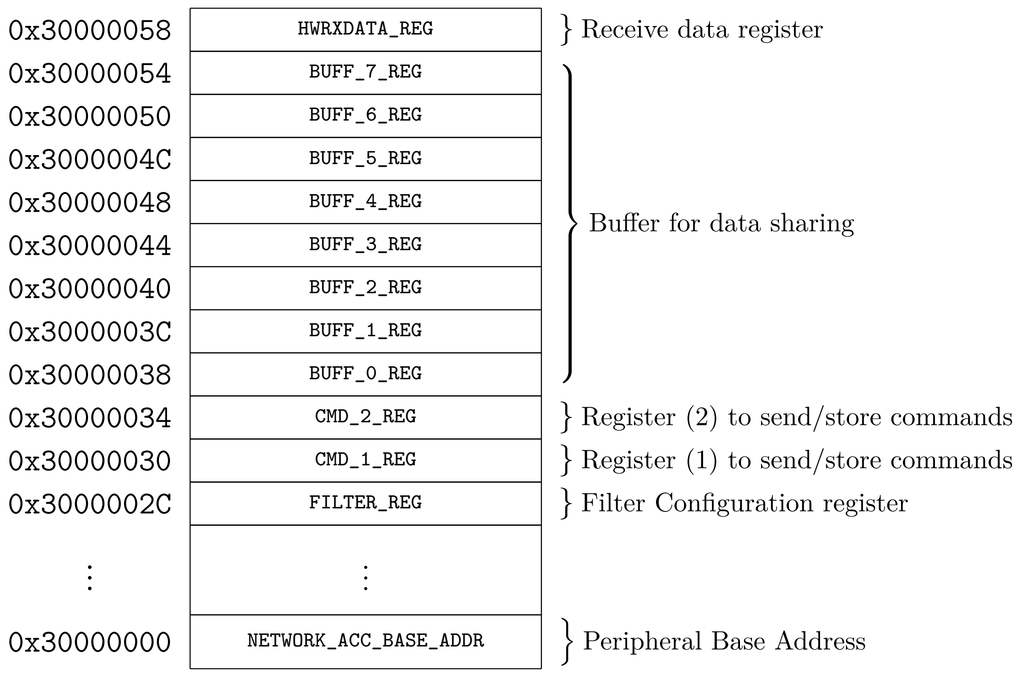


Figure 4.6: Peripheral memory address space.

source address; DST_ADDR enables/disables the peripheral to filter IEEE 802.15.4 Data frames according to the MAC destination address; and DFD enables/disables the DFD mechanism, which can detect the multiple reception of the same frame, up to a configured number of frames. By default DFD is configured to store the information for the last 32 MAC Data frames. The remaining bits are kept reserved to be later used for upgradability purposes if more features are need to be added to the network accelerator.

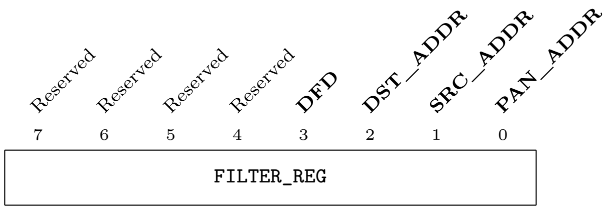


Figure 4.7: FILTER_REG register field.

The CMD_1_REG and CMD_2_REG registers are used to send commands to the peripheral. CMD_1_REG is used to send a read command for retrieving the number of MPDU frames that were filtered according to a certain field. It can be used for statistical purposes only or made available to an intrusion detection system (IDS) system, as it will be explained in the next chapter.

Listing 4.1: File core_accelerator_regs.h with the register definitions required for accessing the peripheral.

```
...
#define NETWORK_ACC_BASE_ADDR   0x30000000UL
/* Core Registers */
/*---------------------------------------------------------------------*/

/* Configuration Register */
#define FILTER_REG_OFFSET       0x2Cu

#define FILTER_DISABLE          0x00u   //disable the peripheral

#define PAN_ADDR_MASK           0x01u   //enable pan address filtering
#define SRC_ADDR_MASK           0x02u   //enable dst address filtering
#define DST_ADDR_MASK           0x04u   //enable src address filtering
#define DFD_MASK                0x08u   //enable dfd module
#define FREE4_MASK              0x10u   //remaining bits are reserved

/* Command 1 Register */
#define CMD1_REG_OFFSET         0x30u

#define PAN_PCKT_MASK           0x01u   //filtered pan packets
#define DST_PCKT_MASK           0x02u   //filtered dst packets
#define SRC_PCKT_MASK           0x03u   //filtered src packets
#define DFD_PCKT_MASK           0x04u   //dfd frames
#define CMD1_FREE4_MASK         0x05u   //remaining bits are reserved

/* Command 2 Register */
#define CMD2_REG_OFFSET         0x34u

#define WR_PAN_MASK             0x01u   //write pan address
#define WR_DST_MAC_MASK         0x02u   //write mac dst address
#define WR_SRC_MAC_MASK         0x03u   //write mac src address
#define CMD2_FREE3_MASK         0x04u   //remaining bits are reserved
/*---------------------------------------------------------------------*/
/* Buffer[8] Register */
#define BUFF_0_REG_OFFSET       0x38u
...
#define BUFF_7_REG_OFFSET       0x54u
/*---------------------------------------------------------------------*/
/* RXData Register */
#define HWRXDATA_REG_OFFSET     0x58u
...
```

The bit fields of `CMD_1_REG`, Figure 4.8, can be used as follows: `PAN_PCKT` is used to request the number of filtered MPDU frames according to their PAN ID, the fields `DST_PCKT` and `SRC_PCKT` are used to request the number of filtered packets according to their destination and source MAC addresses, respectively, and `DFD_PCKT` is used

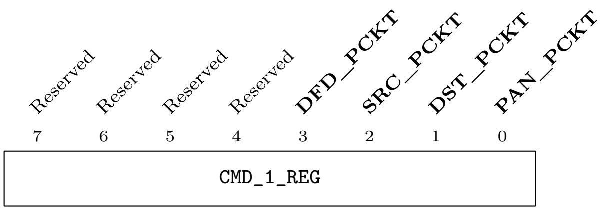


Figure 4.8: `CMD_1_REG` register field.

to request the number of duplicated frames detected. After sending the request, the reply is stored in the BUFF_0_REG to BUFF_7_REG registers and accessed through the peripheral driver provided. The remaining bit fields of the CMD_1_REG register are available for further added features.

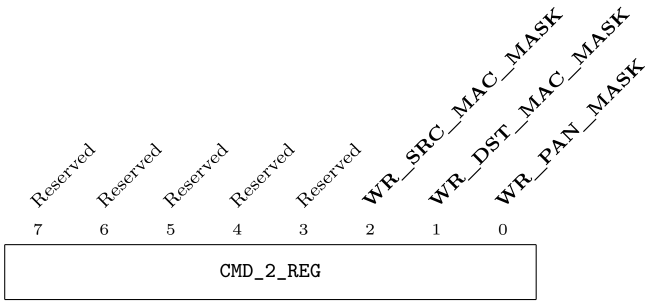


Figure 4.9: CMD_2_REG register field.

The CMD_2_REG is used to store addresses to the corresponding address list stored in the peripheral as internal hardware registers. The bit fields of CMD_2_REG, Figure 4.9, are used as follows: WR_PAN_MASK, WR_DST_MAC_MASK and WR_SRC_MAC_MASK fields indicate, respectively, the PAN, MAC destination and MAC source addresses that are going to be sent and stored in the peripheral. After sending the command, the address values are sent through the BUFF_0_REG to BUFF_7_REG transfer buffer. Registers BUFF_0_REG to BUFF_7_REG are used as a buffer for data exchange and, finally, the HWRXDATA_REG register is used for normal data transfers, e.g., the accepted IEEE 802.15.4 frame, from the peripheral IEEE 802.15.4 Frame Buffer to the MCU.

4.3.4 MLA-compliant API

A software device driver was developed to provide a standard programming interface to the network accelerator. It provides a set of functions for controlling the hardware accelerator by managing each configurable register and its bit field, while hiding these details from the programmer. This driver was tailored and mapped with Contiki-OS system calls, so it can be used transparently by any application or OS internal process. For instance, Listing 4.2 contains a code snippet that shows how the PAN ID is set in the system main function, during the initialization. The radio abstraction layer, Line 14 from Listing 4.2, provides flexibility and simplifies the integration of the peripheral into the OS.

Listing 4.2: Simple example for configuring the PAN address to filter.

```c
static void set_rf_parameters(void)
{
...
#if FILTER_PAN
  PRINTF("\nRadio: enable FILTER_PAN\n");
  uint8_t paddr[2];
  paddr[0] = IEEE802154_PANID & 0xff;
  paddr[1] = IEEE802154_PANID >> 8;

  radio_filter_result = radio_frame_filter_set(PAN_ADDR_MASK, paddr);
  filter_config |= PAN_ADDR_MASK;
#endif /* FILTER_PAN */
...
  NETSTACK_RADIO.set_value(RADIO_PARAM_RX_MODE, filter_config);
}
```

By its turn, the radio abstraction layer will call the software device driver services to set the desired values, e.g., the PAN address. Listing 4.3 displays a code snippet of the function used to set the PAN address. This function uses the HAL functions provided by Microsemi's Libero in the exported firmware. As an example, Listing 4.4 shows how the service is used to set a value into a given register.

Listing 4.3: Device driver API exemple.

```c
radio_filter_t radio_frame_filter_set(uint8_t param, uint8_t* value)
{
    ...
    switch(param)
    {
       case PAN_ADDR_MASK:
         HAL_set_8bit_reg(NETWORK_ACC_BASE_ADDR, BUFF_0, value[0]);
         HAL_set_8bit_reg(NETWORK_ACC_BASE_ADDR, BUFF_1, value[1]);
         HAL_set_8bit_reg(NETWORK_ACC_BASE_ADDR, CMD2, WR_PAN_MASK);
        ...
         return RESULT_OK;
       case SRC_ADDR_MASK:
        ...
         return RESULT_OK;
       default:
         return RESULT_NOT_SUPPORTED;
    }
}
```

Listing 4.4: Microsemi's HAL functions.

```c
/******************************************************************************
 * The macro HAL_set_8bit_reg() allows writing a 8 bits wide register.
 * BASE_ADDR:   A variable of type addr_t specifying the base address of the
 *              peripheral containing the register.
 * REG_NAME:    A string identifying the register to write. These strings are
 *              specified in a header file associated with the peripheral.
 * VALUE:       A variable of type uint_fast8_t containing the value to write.
 */
#define HAL_set_8bit_reg(BASE_ADDR, REG_NAME, VALUE) \
          (HW_set_8bit_reg( ((BASE_ADDR) + (REG_NAME##_REG_OFFSET)), (VALUE) ))
```

4.4 System Evaluation

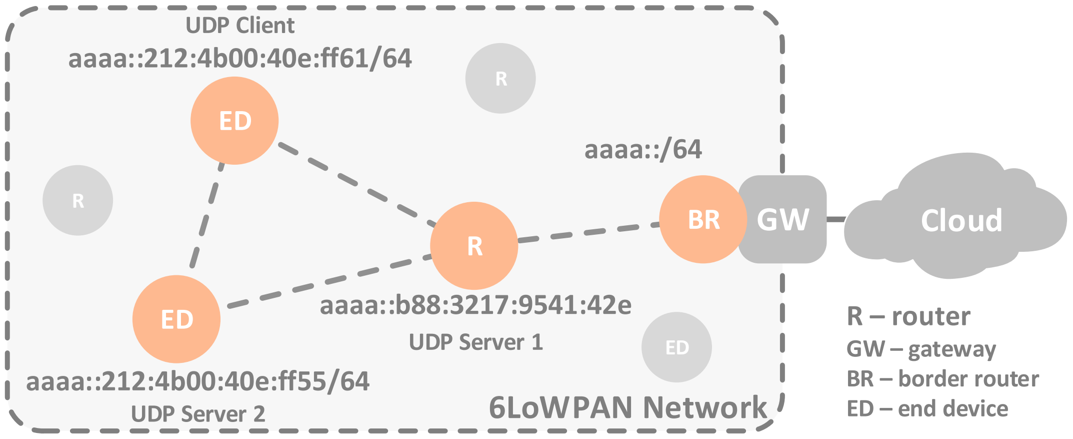


Figure 4.10: Test scenario.

Figure 4.10 depicts the network topology used for conducting the evaluation experiments, where all devices run the Contiki-OS. It consists of a border router (BR) which connects two different networks (interfaces the Internet and the 6LoWPAN network), one Router (R) running a UDP server application (UDP Server 2) and two endpoint devices (ED) with the routing capabilities disabled. These last two run a UDP Client and Server applications, respectively. For the BR, the R and the ED running the UDP client, the Contiki-OS was deployed on a CC2538 SoC and performs all the network packet processing. The SmartFusion2 SoC, which deploys the heterogeneous architecture, runs the UDP Server 1 with the MLA accelerator. The UDP Server 1 uses a connection on UDP ports 4000/4001, while the UDP Server 2 uses ports 5000/5001. The UDP client creates one UDP socket with each server. In order to enable the evaluation of the MLA, that is, comparing the performances between hardware and software-based tasks, all basic filtering features provided by the radio transceiver were disabled. This simple setup performs a typical IoT application (from the sensor application perspective) with high-rate data sampling and packet sending over the network. The UDP connection allows the test of the MLA as the 6LoWPAN relies on the IEEE 802.15.4 Data frames. Hence, when an MPDU is filtered and processed by the MLA the OS is not interrupted and the UDP Server will not receive the frames.

Aiming to evaluate the performance and system availability, two different experiments were conducted. Experiment 4.4.1 targets the Contiki MAC layer APIs and assesses the benefits of the hardware packet filtering over the native Contiki-OS execution. Experiment 4.4.2 runs the Thread-Metric benchmark, along with the UDP Server at the application level, in order to evaluate how the MLA accelerator decreases the OS overhead in terms of system availability to execute other running processes.

4.4.1 MAC Sub-layer: API Evaluation

This experiment evaluates the overall system performance of hardware and software-based MAC Layer APIs for the packet filtering and processing, measuring the elapsed execution time from reading a received MPDU from the radio transceiver until it is delivered to the Network layer. The overhead caused by the OS was also taken into consideration. It consists of: (1) interrupts to the OS execution to register a new event; (2) run pending processes; (3) run the process responsible to transfer the packet from the radio RX buffer to the MAC sub-layer. The performance evaluation is carried out based on the average time of receiving 1000 valid packets from a trusted sender, with a PSR of 8 packet per second (pkt/s). With the MLA disabled, the number of clock cycles needed to process and filter one received MPDU is, on average, 82613, while with the MLA enabled, this value is reduced to 68465, which represents a system's overhead decrease of nearly 17% (1.21 $\times$ speed-up) per received packet. The standard deviation (2δ) is, respectively, 1230 and 978, which means, the hardware implementation is more predictable than the software. Figure 4.11 illustrates the aforementioned results.

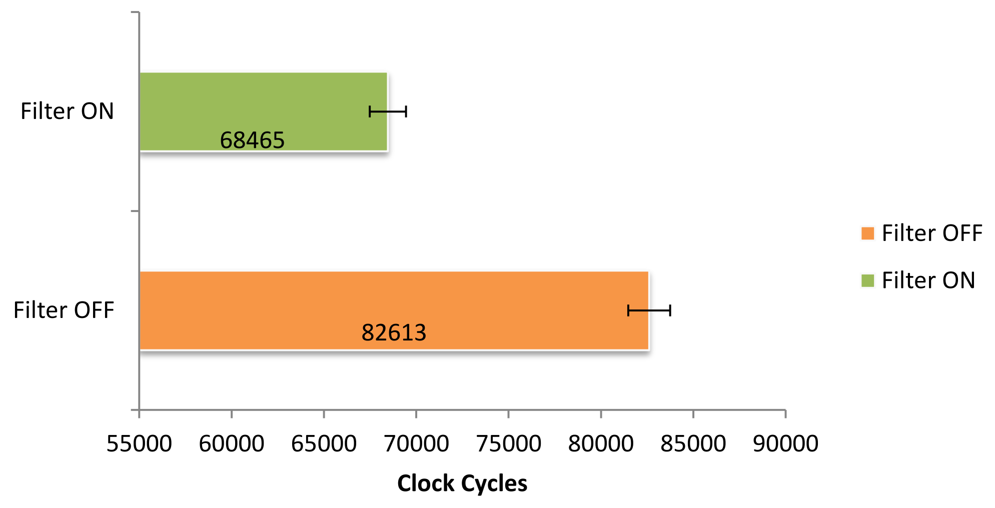


Figure 4.11: OS API evaluation with the MLA enabled and disabled.

4.4.2 Thread-Metric Evaluation

With the MLA disabled, the UDP Client is configured to send packets to the UDP Server at different sending rates (4, 8, 16, 32, 64, 128 and 256 pkt/s), varying the packet discard rate (PDR) value by 10 percentage points at a time, i.e., the number of packets to be discarded by the MLA (0% to 100%). The same experiment was repeated with the MLA enabled. Since Contiki-OS implements an event-based kernel, the scheduler polls all the active system processes, running only those which associated events were triggered. Thus, increasing the number of events, the system's availability is expected to decrease. Enabling new events to be processed by the OS will decrease the system's predictability, since the execution of new scheduled processes, responsible for reading and filtering invalid MPDUs, will delay the execution of other OS processes.

MLA Accelerator Disabled

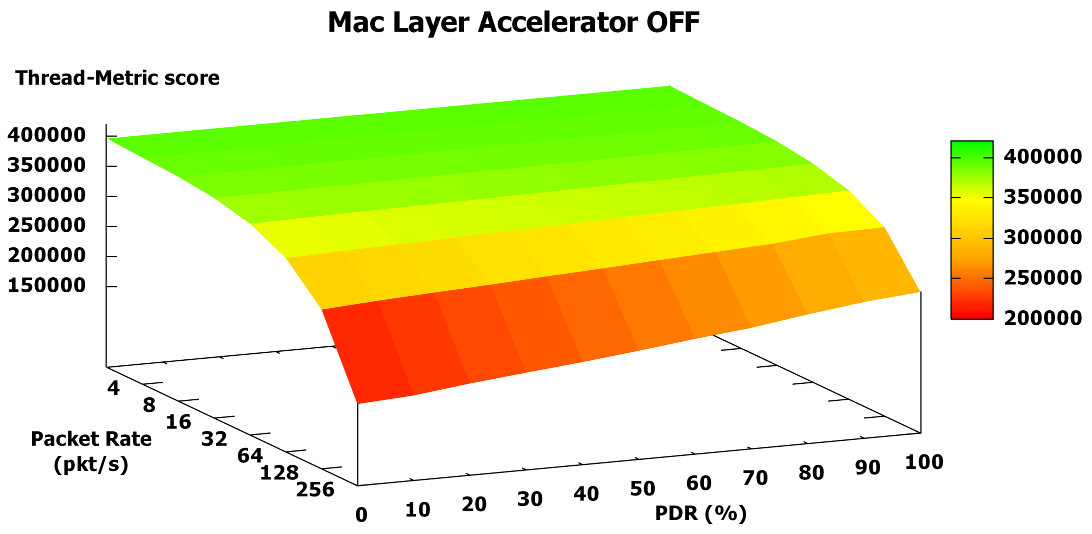


Figure 4.12: Thread-Metric benchmark score with the MLA disabled.

Figure 4.12 depicts the benchmark score results with the MLA disabled. The highest and lowest scores are 150691 and 395273 respectively, representing the highest and lowest PSR for a PDR value of 0%. The score decrease indicates the lowest system's availability as the radio process event is being triggered every time a packet is ready to be read from the radio RX buffer. Increasing the PDR will slightly increase the score. For the best case scenario, i.e., with a PDR value of 100%, the score is 249353 at a PSR of 256 pkt/s and 395273 at a PSR of 0 pkt/s. This is due to the packet rejection being performed at the MAC layer, which means the OS will not trigger the process to handle the new incoming packets and the benchmark process will run earlier. However, it is still visible the impact in the OS's performance when packets are being rejected.

MLA Accelerator Enabled

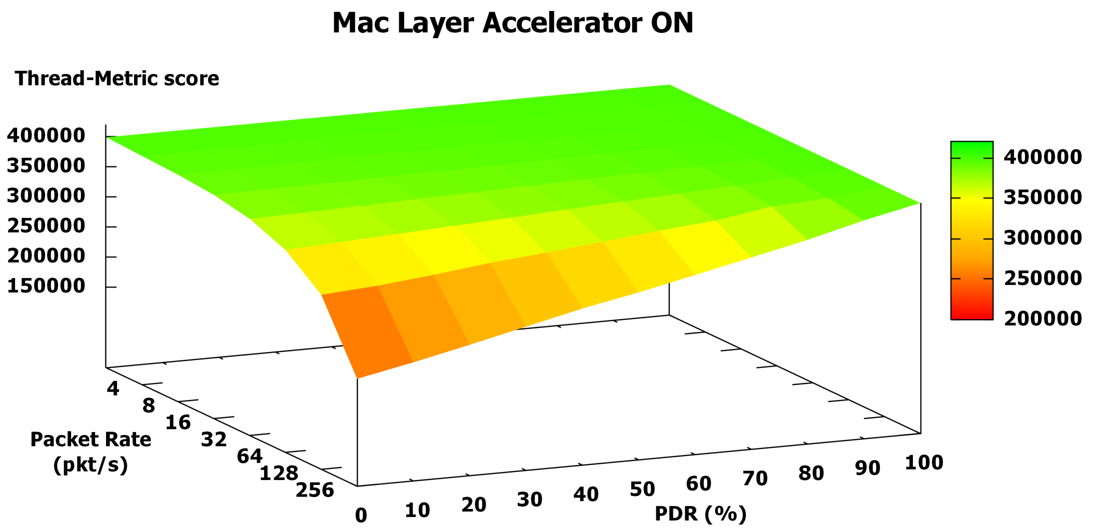


Figure 4.13: Thread-Metric benchmark score with the MLA enabled.

Similarly to Figure 4.12, Figure 4.13 depicts the benchmark score results with the MLA enabled. The highest and lowest benchmark scores are 193484 and 398134 respectively, corresponding to a PSR of 256 pkt/s and 0 pkt/s. The benchmark score increase is directly correlated with the performance gain achieved by the MLA, as discussed in experiment 4.4.1. Increasing the PDR, the benchmark score value increases very fast and at the highest PDR, the variation of the PSR shows no impact on the benchmark score. This is due to the packets being filtered and discarded by the MLA and, consequently, the OS is not interrupted to process dummy frames. Comparing the benchmark score results from both experiments in the best case scenario, i.e., receiving packets at the highest sending rate (PSR of 256 pkt/s) with all frames to be discarded (PDR of 100%), the MLA offers an overhead reduction of 59.6%.

RCU Resources Utilization

Table 4.2 shows the synthesis results, taken from the Microsemi Libero SoC tool, for the MLA accelerator along with its filtering modules, on the RCU side of the heterogeneous architecture. The obtained results are expressed in terms of 4-input

Table 4.2: Synthesis results for the M2S090TS SoC.

Module	***4LUT***	***DFF***
`network_acc_0`	1916	1895
`pan_filter_0`	64	73
`mac_src_filter_0`	300	265
`mac_dst_filter_0`	159	265
`dfd_filter_0`	194	297
Total (out of 86184)	2633 (3.06%)	2795 (3.24%)

Look-Up Tables (LUT) and D Flip-Flop (DFF) used, for each deployed module. The `network_acc_0` is the most costly (4LUT and DFF) since it is responsible to interface the radio transceiver, the adjacent filtering modules, the MCU APB3 bus, as well as to accommodate the IEEE 802.15.4 Frame Buffer. For the remaining modules, the deployment costs are extremely reduced, representing around 3% of the available RCU resources.

4.5 Conclusions

This chapter presented a co-designed hardware/software IEEE 802.15.4 MAC layer accelerator, the MLA, for heterogeneous endpoint devices that target IoT applications. It is responsible for processing and filtering the received IEEE 802.15.4 Data frames, as specified in IEEE 802.15.4-2006 standard, Section 7.5.6.2 (the three levels of filtering). The unwanted MPDUs are processed and discarded by the hardware accelerator, prior the MCU, without interfering with the normal OS execution or interrupting the MCU, that be kept in energy-saving modes. The hardware offloading offered a systems overhead reduction of nearly 17%, while the hardware filtering may reach a reduction of 59.6% when all the received packets are meant to be discarded. The obtained results reveled the benefits of integrating network accelerators, at the MAC Layer of the communication stack, on a heterogeneous architecture, which may contribute for a reduced systems latency and increased predictability. The offered functionalities can be used by any network protocol for Low power and Lossy Networks (LLN) as long as it relies on the IEEE 802.15.4 standard.

Following the standardized IoT-ARM network stack, and since the 6LoWPAN relies on the IEEE 802.15.4 standard, the accelerator must be expanded to other stack layers, since on its current form it does not provide enough contribution to the state-of-the-art available COTS solutions. Hereafter, and since the IPv6 addresses are usually inferred from the IEEE 802.15.4 MAC Header, further accelerators will be deployed for handling the IPv6 address filtering and UDP/TCP port checking. A complete energy characterization for the proposed hardware accelerators is addressed in the next chapter (where all the developed modules are present), in order to compare the heterogeneous solution with related work and analyze the impact of bringing network accelerators to such kind of heterogeneous architectures.

Chapter 5

Accelerating the Network Layer

Bringing IPv6 connectivity to resource-constrained devices in the IoT network results in large amounts of data exchanged between devices, leading to unneeded received packets to be processed and consequent reduced system availability to perform other tasks. This chapter presents the 6LoWPAN accelerator (6LA), an enhanced network accelerator for IP-based IoT networks, that is able to process and filter IPv6 packets received by an IEEE 802.15.4 radio transceiver without the MCU intervention. The 6LA offers nearly 13.2% overhead reduction for the packet processing and filtering tasks, while guaranteeing full system availability when unwanted IPv6 packets are received. An overhead reduction up to 42% can be achieved when all packets are discarded. This contribution is meant to be deployed side-by-side with the MLA on the heterogeneous architecture presented in Chapter 3, which includes on the same architecture, FPGA technology beside an MCU and an IEEE 802.15.4-compliant radio transceiver. Apart from evaluating the energy cost of such accelerator on the heterogeneous architecture, where the processing of packets to be rejected achieved an energy reduction over 99%, this chapter also shows how future generations of radio transceivers can benefit from the proposed solution with an extra cost of 15718 standard cells to their application-specific integrated circuit (ASIC) implementation.

The remaining of this chapter is organized as follows: Section 5.1 introduces this chapter, while Section 5.2 goes through an overview over the 6LoWPAN protocol. Section 5.3 explains the system architecture with its respective integrated accelerators, their functionalities and the provided API and Section 5.4 discusses the system evaluation, where performance experiments and energy analysis tests were realized. Section 5.5 concludes this chapter.

Related Publications

Ideas and findings presented in this chapter resulted in the following publications:

- **T. Gomes**, F. Salgado, S. Pinto, J. Cabral and A. Tavares, "*Towards an FPGA-based network layer filter for the Internet of Things edge devices*," 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, 2016, pp. 1-4.
- **T. Gomes**, F. Salgado, S. Pinto, J. Cabral and A. Tavares, "*A Hardware 6LoWPAN Packet Filter for IoT Edge Devices,*" in **IEEE Internet of Things Journal** (Under Review).

5.1 Introduction

Bringing IPv6 connectivity to IEEE 802.15.4-based networks, which are mainly characterized by the low-bandwidth communications channels and resource-constrained devices, is not straightforward. For instance, due to the IPv6 default minimum MTU size, a non-fragmented IPv6 packet would be too large to fit in an IEEE 802.15.4 MPDU. Also, the 40 bytes size IPv6 header would waste the limited available PHY bandwidth. To tackle these issues, the 6LoWPAN working group has defined an effective adaptation layer between IPv6 and IEEE 802.15.4 MAC levels (RFC6282 and RFC4944 [51,181]) allowing IP packets to be sent over LLN. The major concerns on maintaining the regular IPv6 services, e.g., IPv6 addresses auto-configuration, link-layer subnet broadcast support in shared networks, adoption of new and lightweight application protocols and reduced routing overhead with efficient routing protocols, were always taken into great consideration. In addition, the inherent security aspects, e.g., information confidentially and integrity on data collection and message exchange, are completely preserved.

Aiming to speed-up application-related operations and to reduce the complexity of processing overhead caused by heavy processing algorithms, hardware-accelerated solutions for IoT-enabled devices are already available, as it was presented and discussed in Chapter 1. Regarding the current state-of-the-art on the 6LoWPAN standard, it is proposed by [182–187] some enhancements at the protocol level, e.g., Neighbor Discovery and Network managing services. However, this research only focus on the network services aspects without considering the handling of the transmitted/received IPv6 packets. Hardware-accelerated approaches targeting the network packet processing overhead, such as the processing and filtering of the received IPv6 packets in IoT-enabled low-end devices, have never been attempted.

The main contributions of this chapter are: (1) an efficient and reconfigurable 6LoWPAN accelerator, the 6LA, specially tailored to retrieve, process, filter and detect multiple receptions of IPv6 packets on heterogeneous endpoint devices in IoT networks; (2) a performance, energy and hardware resources evaluation of the 6LA; (3) the deployment of the 6LA on the heterogeneous architecture and its integration into the CUTE mote, presented on Chapter 3. Software APIs were also developed in order to integrate and evaluate the 6LA with the Contiki-OS. These APIs enable a seamless integration with any available IoT OS. Since the 6LoWPAN relies on the IEEE 802.15.4 standard, the 6LA also integrates features from the MAC accelerator presented on Chapter 4.

5.2 6LoWPAN Adaptation Layer

Starting from a maximum physical layer packet size of 127 octets, provided by the IEEE 802.15.4 Data frames, and a maximum frame overhead of 25 octets, the remaining maximum frame size at the media access control layer is 102 octets. When used, Link-layer security imposes further overhead, which in the maximum case (21 octets of overhead in the AES-CCM-128 case, versus 9 and 13 for AES-CCM-32 and AES-CCM-64, respectively) leaves only 81 octets available. This is far below the IPv6 packet size of 1280 octets, thus an efficient compression and fragmentation and reassembly mechanism is mandatory. The 6LoWPAN adaptation layer defines encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4-based networks.

All the 6LoWPAN datagrams transported over a IEEE 802.15.4 MPDU are prefixed by a group of headers, each one identified by a type field. By defining an efficient header compression mechanism [181], where redundant information can be inferred from other layer payloads, large header fields such as the 128-bit long IPv6 addresses can be easily calculated from the IEEE 802.15.4 headers. This way, IPv6 packets and regular network traffic can be efficiently sent over IEEE 802.15.4-based networks. The drawback caused by this mechanism is the overhead added to the packet processing, since the header fields cannot be directly retrieved and therefore must be, most of the times, derivate from the MAC layer. Also, and since on the network edge the devices are mostly battery powered, the high traffic load induced by the IPv6 heavily affects the overall energy consumption. Reducing the number of packets to be processed by the device's CPU, while maintaining the application requirements, would result in a more efficient device operation and thus, an increased battery lifetime.

5.2.1 6LoWPAN Frame

As specified by the 6LoWPAN adaptation layer, the IPv6 frames are carried on an IEEE 802.15.4 Data frame payload. Figure 5.1 depicts an example of an UDP Data frame sent between two nodes carried out in an IEEE 802.15.4 data frame payload, where the IPv6 headers are compressed and must be derivate from the MAC layer.

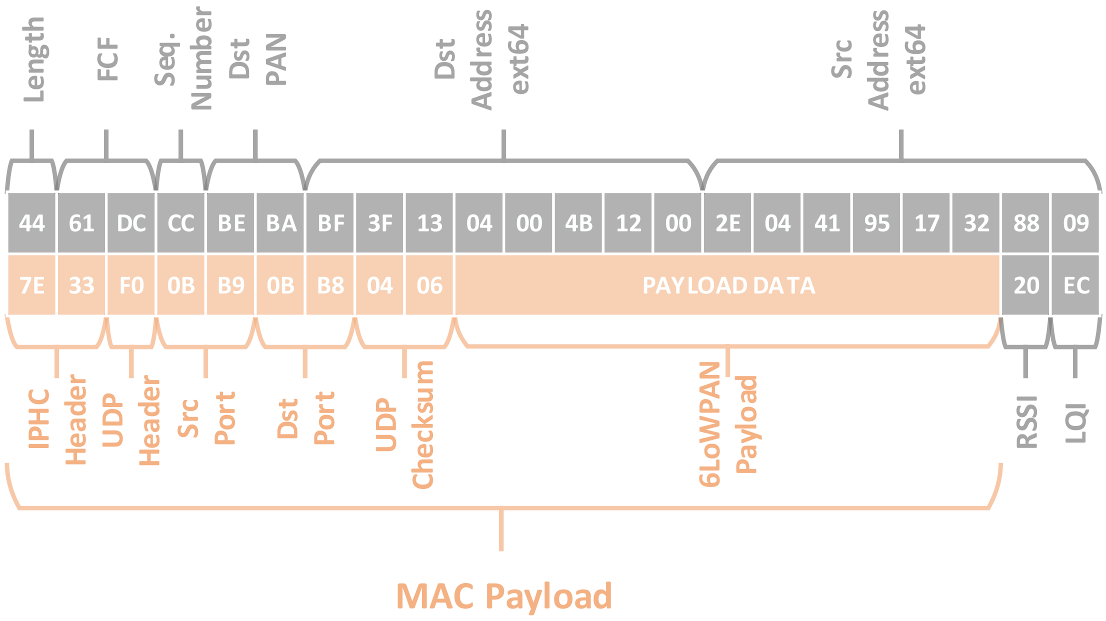


Figure 5.1: 6LoWPAN frame format.

For the 6LoWPAN header, there are four possible categories, regarding their role on the network:

1. **No 6LoWPAN**: when the MAC payload is not 6LoWPAN compliant and must be discarded at the adaptation layer;
2. **Dispatch Header**: used for Header compression purposes while keeping the backwards compatibility with former encapsulation and compression schemes, e.g., the *LOWPAN_HC1*;
3. **Mesh Addressing Header**: used for forwarding IEEE 802.15.4 frames at link-layer;
4. **Fragmentation Header**: used when a datagram does not fit into a single frame and must be fragmented.

Table 5.1 summarizes the four categories with the first two bits and the following subcategories with the remaining bits.

According to Table 5.1, the example frame illustrated in Figure 5.1 consists of an IEEE 802.15.4 MAC frame with a 6LoWPAN payload. The first two bytes (`0x7E33`) correspond to the Header Compression mechanisms, which indicates that the frame is a 6LoWPAN using a `LOWPAN_IPHC` Header Compression scheme and the following bits must be used to retrieve all the corresponding header fields.

Table 5.1: 6LoWPAN Headers.

	First 2 bits	***Following bit combinations***	
No 6LoWPAN	00	xxxxxx	Any combination
Dispatch	01	000000	Additional Dispatch byte follows
		000001	Uncompressed IPv6 Addresses
		000010	LOWPAN_HC1 compressed IPv6
		010000	LOWPAN_BC0 broadcast
		1xxxxx	LOWPAN_IPHC compressed IPv6
Mesh Addressing	10	xxxxxx	Any combination
Fragmentation	11	000xxx	First Fragmentation Header
		111xxx	Subsequent Fragmentation Header

5.2.2 LOWPAN_IPHC Encoding

Within the same WPAN, Header Compression mechanisms are expected to be often used and the headers easily obtained without explicit indication by the source node. For the sake of simplicity, only relevant fields and how they can be retrieved from the encoded bits are explained. Remaining `LOWPAN_IPHC` base encoding bits are explained on Section 3.1 of the RFC6282 [181]. If some of the IPv6 header fields have to be carried in-line, they follow the `LOWPAN_IPHC` encoding rules. In the best case, the `LOWPAN_IPHC` can compress the IPv6 header down to 2 bytes in an IPv6 link-local communication (i.e., a direct single-hop communication). When a packet is routed through multiple hops, `LOWPAN_IPHC` can compress the IPv6 header down to 7 bytes.

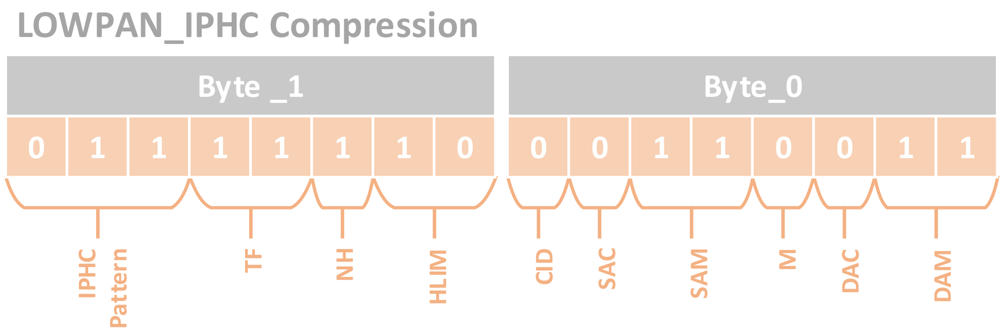


Figure 5.2: LOWPAN_IPHC encoding format.

Taking the 6LoWPAN example frame from Figure 5.1, the information obtained on Figure 5.2 can be retrieved by decoding the 16-bit LOWPAN_IPHC field, which starts with the **LOWPAN_IPHC Pattern** (`0b011`).

Some parameters and compressed header fields, are retrieved in the following way:

- The **Traffic Class & Flow Level (TF)** information that, in this case, states that the Traffic Class and Flow Label are elided (`0b11`).
- The 6LoWPAN *Payload Length* can be inferred either from the *MAC Frame Length* or from the *Datagram Size Field* in the fragmentation header (if present).
- The **Hop LIMit (HLIM)**, set to a well-known value by the source node, indicates that the HLIM field is compressed and the hop limit value is 64, corresponding to the bit fields set to `0b10`.
- **Next Header (NH)** field: the 6LoWPAN provides compression formats for different next-headers, which are identified by a variable-length bit-pattern immediately following the `LOWPAN_IPHC` compressed header. Each next-header in the original IPv6 packet will be present in the compressed one in the same order and it will be encoded with the appropriate `LOWPAN_NHC` format. In this example, the **NH** field is compressed, `0b1`, and encoded by using the `LOWPAN_NHC` mechanism. For the packet displayed in Figure 5.1, the NH field corresponds to an UDP Header (`0xF0`).
- **Source Address Compression (SAC)** field: the `LOWPAN_IPHC` encoding mechanism performs effective compression of unique local, global, and multicast IPv6 addresses. Addresses assigned to 6LoWPAN interfaces are formed with an Interface Identifier derived directly from MAC addresses. Regarding the Source address, for this example, the SAC field indicates that Source address compression uses stateless compression (`0b0`).
- The **Source Address Mode (SAM)** bits state how to retrieve the source address field. If SAC=0 and SAM=11, the Source address is fully elided. The first 64 bits of the address are the link-local prefix padded with zeros and the remaining 64 bits are computed from the encapsulating header (e.g., IEEE 802.15.4 or IPv6 source address). The resulting link-local IPv6 address is **`[fe80:0000:0000:0000:0b88:3217:9541:042e/64]`** and it can be calculated using the SAC and SAM information. It also can be represented by **`[fe80::b88:3217:9541:42e/64]`**, if represented in the IPv6 compressed version.
- The **M** field indicates if the Destination address is a multicast address. In this case is `0b0` and states that the Destination address is a unicast address.

- **Destination Address Compression (DAC)** field: like in the SAC encoding bits, the DAC indicates the Destination address compression used, and in this case, it uses stateless compression (`0b0`). The Destination Address Mode (DAM) works similarly to SAM. If M=`0`, DAC=`0` and DAM=`11`, the address is fully elided. The first 64 bits of the address are the link-local prefix padded with zeros and the remaining 64 bits are computed from the encapsulating header. `[fe80:0000:0000:0000:0212:4b00:0413:3fbf/64]` is the resulting link-local IPv6 address, calculated using the DAC and DAM information. It can be simplified as `[fe80::212:4b00:413:3fbf/64]`.

Performing the heavy task of retrieving the header fields, before processing them, requires processing capabilities and system availability. Considering that on the same WPAN several nodes are in communication range of each other, most of the traffic received by a node is intended for another, resulting in a redundant packet processing, since unwanted received packets will be dropped. Offloading the `LOWPAN_IPHC` header decompression task from the OS to an accelerator block, alleviates the overhead caused by the processing of the 6LoWPAN headers, while avoiding unnecessary processing when unwanted packets are discarded.

5.3 6LoWPAN Accelerator (6LA)

In order to assess and explore the possibilities and challenges of a network packet processor for low-end IoT devices, the hardware accelerator was developed and deployed on heterogeneous architecture presented in Chapter 3. Figure 5.3 depicts

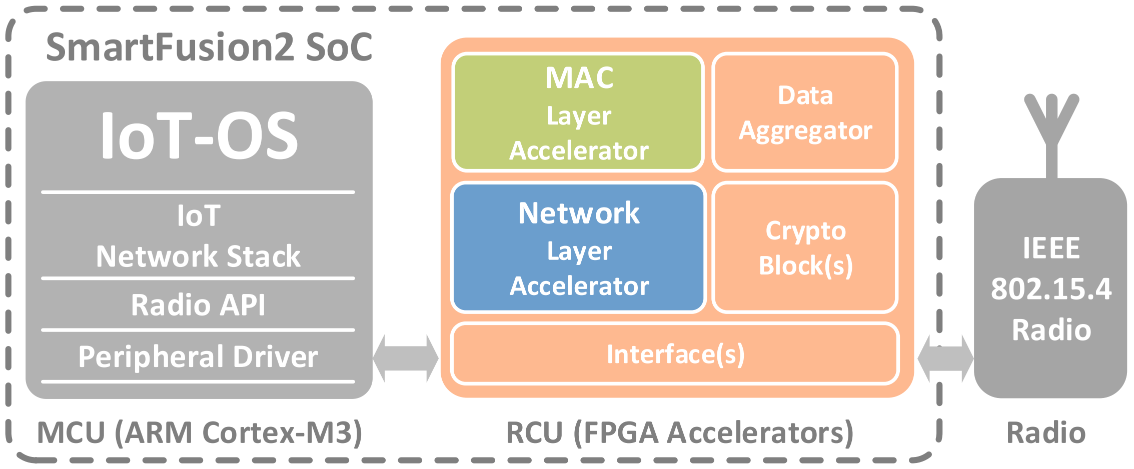


Figure 5.3: Adding the 6LA to the heterogeneous architecture.

the IEEE 802.15.4 and 6LoWPAN accelerators, deployed side-by-side. It provides several protocol-related filtering and processing mechanisms, which can help in alleviating processing load from the MCU and assist software-based tasks such as intrusion detection systems.

5.3.1 General Architecture

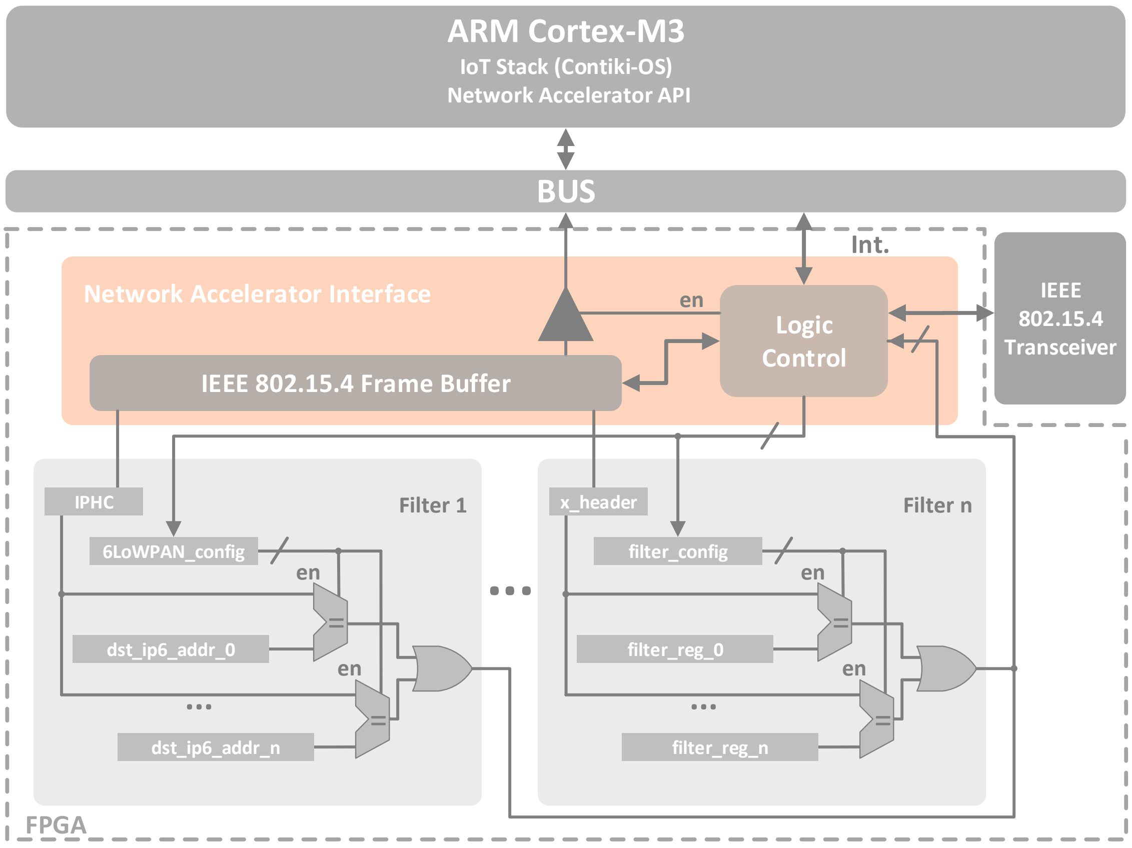


Figure 5.4: Architectural RTL representation of the 6LA.

The system RTL model is depicted in Figure 5.4. Globally, it is composed by the network accelerator interface, which connects the hard-core processor via the APB3 protocol and the IEEE 802.15.4 transceiver using the SPI protocol bus. Different filter modules can be independently configured and connected to the network accelerator block to perform the desired filtering/processing operation, accordingly to the application requirements. This building blocks approach provides scalability to the solution since it enables new filtering modules to be easily added to the 6LA. Regarding the received packets, these blocks can integrate processing and filtering tasks for the MAC and network layer (6LoWPAN), as well as performing specific security-related tasks such as packet encryption/decryption algorithms, resorting the most recent security mechanisms. Creating filter modules in a building blocks approach provides design-time scalability by mitigating the development and the integration of new filtering modules to the heterogeneous architecture.

When the radio transceiver receives and holds a valid packet, the network accelerator interface is triggered to transfer the packet to the IEEE 802.15.4 Frame Buffer. Then, the enabled filtering/processing modules will take action and trigger an output to the network accelerator interface if the packet passes or fails the settled rules. If the packet passes, it is transferred to the OS in order to be processed in the upper stack layers. If the packet fails the filter rules, it is discarded and the current processor's flow is not interrupted, leaving it available to run other tasks. Components of the network accelerator interface, e.g., configuration, status registers, packet parameters, filtering statistics, can be accessed at any time by the OS through the developed API. Because the 6LoWPAN relies and uses features from the IEEE 802.15.4 MAC protocol, modules from the MAC sub-layer accelerator were needed to be integrated and deployed side-by-side. In this way, the MAC accelerator can be used as standalone when the IoT network implements a different Transport data protocol, rather than the 6LoWPAN, over the IEEE 802.15.4 standard as well as when such features are not available on the radio transceiver hardware.

5.3.2 Deployed Modules

Header Compression and Packet Handling

In order to enable and provide packet filtering solutions, the IPv6 packet must be previously retrieved from the MAC Data frame payload. Therefore, when an IEEE 802.15.4 Data frame is stored in the IEEE 802.15.4 Frame Buffer, it is automatically decompressed into an IPv6 packet and all the fields set available to the OS and the filtering modules. The decompression mechanisms are based on the algorithm explained previously in Section 5.2.2.

Address Filtering

The 6LoWPAN packet filter is composed by a source and destination address filter modules. Both can be configured with a desired number of IPv6 addresses to be compared with the received packet. If the received packet matches any of the configured addresses (e.g., Link-Local and Global addresses), an output for the network accelerator interface module is triggered. For vital network protocols, e.g., mDNS, RPL routing messages, and ICMPv6 messages, the 6LoWPAN modules will always flag a valid packet. This way, the network nodes can keep up-to-date information about the network topology while maintaining the network integrity. For handling the IPHC mechanisms of the IPv6 protocol, these modules are able to retrieve all the existing fields and calculate the IPv6 addresses, when compressed, from the corresponding MAC Addressing fields. Offloading this task from the OS alleviates the overhead caused by the heavy processing of the 6LoWPAN headers.

TCP/UDP Filter

When the received packet contains a UDP/TCP payload, this module verifies the local and remote ports. When combined with the previous modules it turns the network accelerator into a powerful IPv6 UDP/TCP filter that is able to discard packets if there are no bind connections with the node's IP address. If the configured socket is listening on different ports than the received on the UDP/TCP packet fields, the packet is discarded. When the node is configured as a router, the module can still forward the packet to the OS in order to generate an ICMPv6 "Destination Unreachable Message" to notify the sender. This can be disabled if port scan probes are undesired.

Other 6LoWPAN modules in development

There are other crucial features that are required by the 6LoWPAN in order to maintain the IPv6 network working in a more efficient way. Besides compressing the IPv6 addresses in an efficient manner, the `LOWPAN_IPHC` also introduces an extension that is used to identify IPv6 Global Addresses that are commonly used in the network, e.g., a remote server address. Thus, the addresses have not to be carried in-line every time a message is exchanged between a node and the remote server. For that purpose, the `LOWPAN_IPHC` introduces the Context IDentifier (CID) Extension that expects a conceptual context is shared between the node that compresses a packet and the node(s) that needs to expand it. However, how the contexts are shared and maintained is out of the scope of the standard specification [181]. The specification enables a node to use up to 16 contexts and the context used to encode the source address has not to be the same as the context used to encode the destination address. If the CID field is set to 1 in `the LOWPAN_IPHC` encoding, then an additional octet will extend the LOWPAN_IPHC encoding following the DAM bits, but before the IPv6 header fields that are carried in-line. The additional octet identifies the pair of contexts to be used when the IPv6 source and/or destination address is compressed. The context identifier reserves 4 bits for each address type, supporting up to 16 contexts, where context 0 is the default context. A Context Table (CT) maps the most frequent Internet address prefixes to context identifiers of several bits, which are used in the packets generated by the border router. The CT design and how the tables are maintained are not specified. However, few approaches to dynamically update each CT, and how the information is disseminated over the network have been proposed by [185,186]. Since the dynamic algorithm applied is heavy, and with the increased number of global host that might need to communicate with the nodes inside the 6LoWPAN network, it is also proposed the addition of a module to the 6LoWPAN accelerator in order to handle these tables and how the information is disseminated. Such task, in the Contiki-OS, is being under evaluation and proposed to be offloaded.

5.3.3 Peripheral Interface

After deploying the 6LA on the RCU, and since it shares the same memory space with the MAC accelerator, it was necessary to update the peripheral interface created in Section 4.3.3 from Chapter 4. The update concerns the registers FILTER_REG, CMD_1_REG and CMD_1_REG bit fields on the core_accelerator_regs.h file. Regarding the FILTER_REG, Figure 5.5, the reserved bit fields were updated as follows:

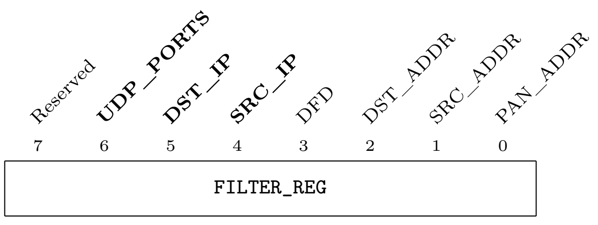


Figure 5.5: FILTER_REG register field.

SRC_IP, DST_IP and enables/disables, respectively, the filtering functionalities of source/destination IPv6 addresses (can be Local-Link, Global, etc.), and UDP_-IP adds the functionality to additionally filter the packet by the UDP remote/local) ports in use. Regarding the CMD_1_REG register, Figure 5.6, the reserved bit fields are now used as follows: SRCUDP_PCKT, DSTUDP_PCKT, SRCIP6_PCKT_MASK and DSTIP6_PCKT_MASK are used to request the number of filtered packets for source and destination UDP ports and source and destination IPv6 addresses, respectively.

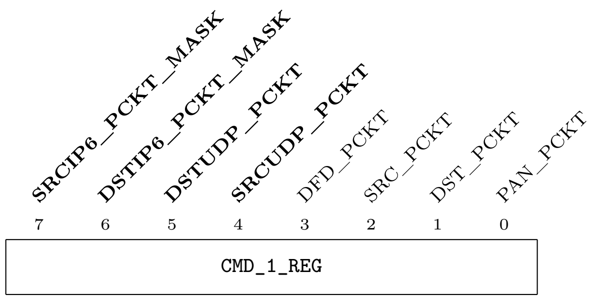


Figure 5.6: CMD_1_REG register field.

For the CMD_2_REG register, Figure 5.7, there were added new bit fields as follows: WR_UDP_PORTS_MASK is used to set a new pair of local/remote ports to be added to the filter. New ports were added to the BUFF_0_REG, BUFF_1_REG, BUFF_2_REG and BUFF_3_REG, to be next added to the list of remote/local ports. The same procedure is used to add source and destination IPv6 addresses. Because the IPv6 uses 16 octets, the transfer occurs in two steps, first the network prefix is set by asserting the WR_SRCIP6_PREFIX_MASK bit and followed by the 64-bit IPv6 address when the WR_DSTIP6_ADDR_MASK bit is set. Listing 5.1 summarizes the file with the performed updates.

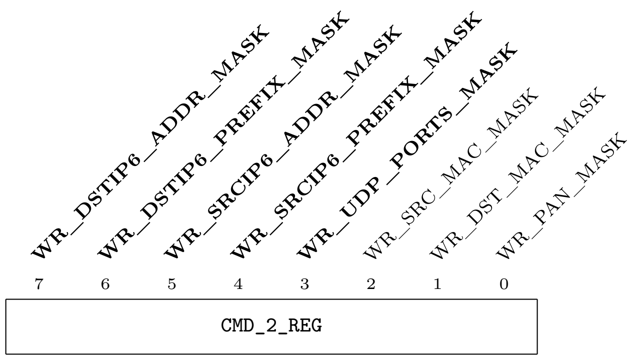


Figure 5.7: CMD_2_REG register field.

Listing 5.1: Updated core_accelerator_regs.h file.

```c
...
#define NETWORK_ACC_BASE_ADDR   0x30000000UL
...
#define FILTER_REG_OFFSET       0x2Cu
...
#define SRC_IP6_MASK            0x10u   //enable src IPv6 address filtering
#define DST_IP6_MASK            0x20u   //enable dst IPv6 address filtering
#define UDP_PORTS_MASK          0x40u   //enable UDP remote/local filtering

/* Command 1 Register */
#define CMD1_REG_OFFSET         0x30u
...
#define SRCUDP_PCKT_MASK        0x05u   //filtered src udp packets
#define DSTUDP_PCKT_MASK        0x06u   //filtered dst udp packets
#define DSTIP6_PCKT_MASK        0x07u   //filtered dst ipv6 packets
#define SRCIP6_PCKT_MASK        0x08u   //filtered src ipv6 packets

/* Command 2 Register */
#define CMD2_REG_OFFSET         0x34u
...
#define WR_UDP_PORTS_MASK       0x04u   //write udp ports
#define WR_SRCIP6_PREFIX_MASK   0x05u   //write src ipv6 prefix
#define WR_SRCIP6_ADDR_MASK     0x06u   //write dst ipv6 address
#define WR_DSTIP6_PREFIX_MASK   0x07u   //write src ipv6 prefix
#define WR_DSTIP6_ADDR_MASK     0x08u   //write dst ipv6 address
...
```

5.3.4 6LA-compliant API

The software driver developed to access the network accelerator in Chapter 4 was updated to support the new functionalities of the filter. As an example, Listing 5.2 depicts a code snippet for a UDP port filtering configuration. In short, after the application starts, it sets an UDP connection with the defined local/remote ports.

Listing 5.2: Simple example for configuring the UDP ports in the peripheral registers.

```c
...
#define UDP_REMOTE_CONN_PORT    4001
#define UDP_LOCAL_CONN_PORT     4000
...
  server_conn = udp_new(NULL, UIP_HTONS(UDP_REMOTE_CONN_PORT), NULL);
  udp_bind(server_conn, UIP_HTONS(UDP_LOCAL_CONN_PORT));
...
  udp_ports[0] = UDP_LOCAL_CONN_PORT & 0xff;
  udp_ports[1] = UDP_LOCAL_CONN_PORT >> 8;
  udp_ports[2] = UDP_REMOTE_CONN_PORT & 0xff;
  udp_ports[3] = UDP_REMOTE_CONN_PORT >> 8;
...
  //write UDP ports to buffer
  filter_config |= UDP_PORTS_MASK;
  radio_frame_filter_set(UDP_PORTS_MASK, udp_ports);
  //write the filter config
  NETSTACK_RADIO.set_value(RADIO_PARAM_RX_MODE, filter_config);
 ...
```

Afterwards, it writes the local/remote ports to the peripheral buffer registers followed by the new filter configuration which is written to the `FILTER_REG` register. From now, the network accelerator will reject any UDP connection if the local/remote ports mismatch the configured ones. The rest of the software driver follows the same principle as depicted before in Chapter 4, Section 4.3.4.

5.3.5 Handling Security in 6LoWPAN

Data Security

As security is becoming a major requirement for most of IoT applications, current technology must provide mechanisms that can protect, verify and authenticate data exchanged between trusted nodes [45]. However, the traditional security mechanisms that are widely used on the Internet, e.g., IPsec, are too heavy to be integrated on small-constrained devices. Some approaches have already been proposed and deployed at two different layers of the IoT network stack, as discussed in [55]. For the MAC layer, the IEEE 802.15.4e standard specifies mechanisms to achieve data encryption and authentication. Despite the last versions of the standard providing security, keys management schemes and authentication policies are not specified, being these issues addressed in the upper layers. For securing the 6LoWPAN communications, and since available IPv6 protocol stacks use IPsec to secure data exchange, Compressed IPsec [159] is proposed as a 6LoWPAN extension for IPsec support. It mitigates the usage of IPsec over the IoT low-end devices while keeping the existing end-points on the Internet unmodified. The Compressed IPsec was implemented in the Contiki-OS μIP stack and its impact was evaluated in terms of memory footprint, packet size, energy consumption and performance under different configurations. Such contribution suggests that future IPsec systems should use cryptographic algorithms such as AES-CBC-128 for encryption and AES-XCBC-MAC-96 mode for authentication. Although these algorithms are suitable to integrate into the low constrained devices, when deployed as software-only, they demand a high level of resources, such as CPU, memory requirements and energy, thus the efficiency of the IPsec can still be improved by resorting hardware support. The future work pointed by the Compressed IPsec motivate the 6LoWPAN IPsec extension features integration on the CUTE mote, which is not only architecturally compliant, but also complementary to the developed solution. Despite the CUTE mote architecture being able to support security mechanisms for handling the secure

6LoWPAN packets, by easily adding security blocks that resort hardware cryptographic modules, their implementations and evaluations, are out of the scope of this thesis, but will be addressed in the future.

DoS Security

The strong connectivity of IoT environments requires a holistic, end-to-end security approach, addressing security and privacy risks at all abstraction levels [54]. Exposing resource-constrained endpoint devices to the Internet opens many doors to attackers to exploit device vulnerabilities by creating Denial of Service (DoS) events to reduce, disrupt or completely eliminate the network communication and device availability [54, 188]. There can be different kinds of DoS attacks observed at the MAC layer such as misbehavior and selfish attacks. Malicious nodes manipulate the MAC protocol parameters such as back-off time, network allocation vector value and short inter frame space, or flood the network with a large number of dummy packets. Under such scenarios, the attacker nodes can capture entire network bandwidth causing legitimate nodes to fail to communicate with other nodes, consequently decreasing the throughput of the nodes significantly. A proper DoS detection mechanism allied to an efficient packet filter can contribute for more efficient and reliable communication systems. Given this, the proposed work can trigger future research focused on hardware DoS detection mechanisms for low-end devices.

In [188], Suricata, an open-source IDS system, was used to create an IDS for 6LoWPAN-based networks and implemented over Contiki-OS. Suricata helps creating packet actions (i.e., pass, drop, reject, alert) for a specific protocol (UDP, TCP, ICMP, etc.). Rules are based on packet's source address and respective source port, destination address with corresponding destination port, followed by other rule options. Such options provide the flexibility to the IDS and novel rule options can be developed to extend the detection parameters. For instance, in the case of a UDP flood attack, an alert can be triggered if the number of received packets is above a defined threshold, e.g., 50 packets per second. The developed filtering features deployed under the 6LA, such as the UDP/TCP port filtering, combined with the IDS, can help preventing DoS attacks in a variety of scenarios such as an UDP flooding attack. Taking benefit from the available hardware, the additional overhead caused by the IDS can also be mitigated by the 6LA. Despite related, integrating such functionalities with the IDS is out of the scope of this thesis, but proposed as future work.

5.4 System Evaluation

For evaluating the 6LA, a similar experiment setup to the one presented in Figure 4.10 of Chapter 4, was conducted. The 6LoWPAN filtering (with the source and destination filter modules) along with the IEEE 802.15.4 MAC filters were enabled (needed to extract the IPv6 addresses from the MAC Headers). In addition, the DFD module was enabled, despite not affecting the results. The PAN filter as well as the TCP/UDP modules were disabled. The accelerator was configured to accept packets to its own IPv6 addresses, i.e., Global and Link-Local.

5.4.1 Network Layer: API Evaluation

This first experiment evaluates the system's performance, comparing both hardware and software implementations for the packet processing and filtering. To evaluate the achieved performance due to offloading filtering software functions to hardware, micro-benchmarks were executed. These were used to measure the execution time needed to read a packet from the radio transceiver and deliver it to the network layer. The performance evaluation is calculated based on the average time for receiving 1000 packets by the UDP Server 1. Figure 5.8 shows the obtained results for software and hardware filtering when a valid IPv6 packet is received. With the filter disabled, the number of clock cycles required to process and filter one packet is on average 78156. When the filter is enabled, this value is reduced to 67806. This represents a system's overhead reduction of 13.24% (1.15× speed up), per received packet.

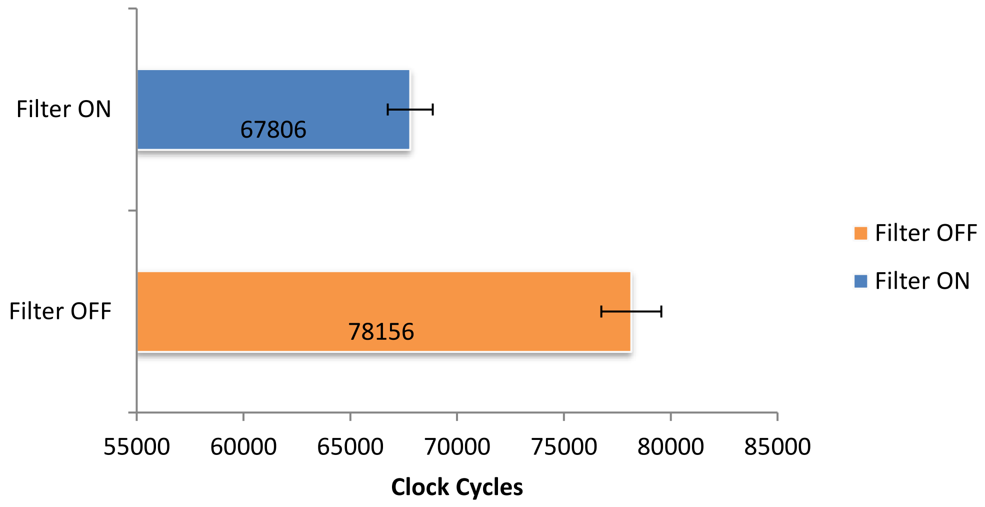


Figure 5.8: API performance evaluation.

5.4.2 Thread-Metric Evaluation

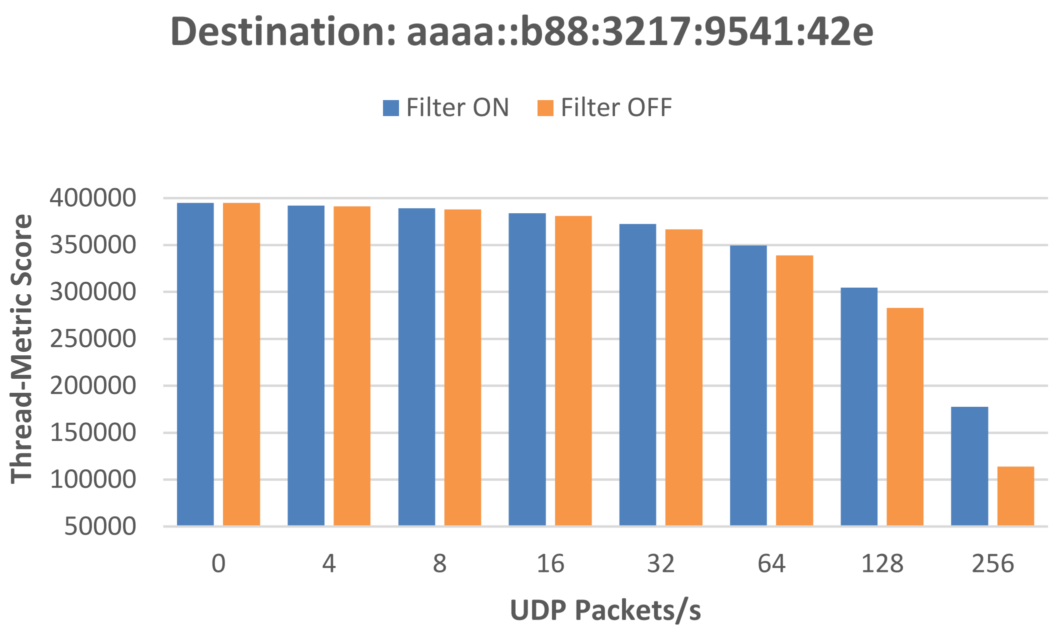


Figure 5.9: UDP Client sending packets to UDP Server 1.

UDP Client to UDP Server 1

Figure 5.9 depicts the results obtained from running the benchmark on the UDP Server 2 (with the 6LA enabled and disabled) with packets being sent from the UDP Client to the UDP Server 1. The UDP Client was configured to send messages at different PSR values, varying from 0 to 256 pkt/s. When PSR is 0, the score is at its highest value, nearly 395093 with the 6LA enabled and disabled, which represents the highest system availability since there are no packets being processed. Increasing the PSR value, the score tends to decrease exponentially, since the OS is processing and discarding new packets, intended to another node. However, with the 6LA enabled, the score is slightly higher (35.9% better compared with the previous score, for the highest value of PSR). This is due to the performance gain, achieved by the packet filtering and discarding being processed by the hardware.

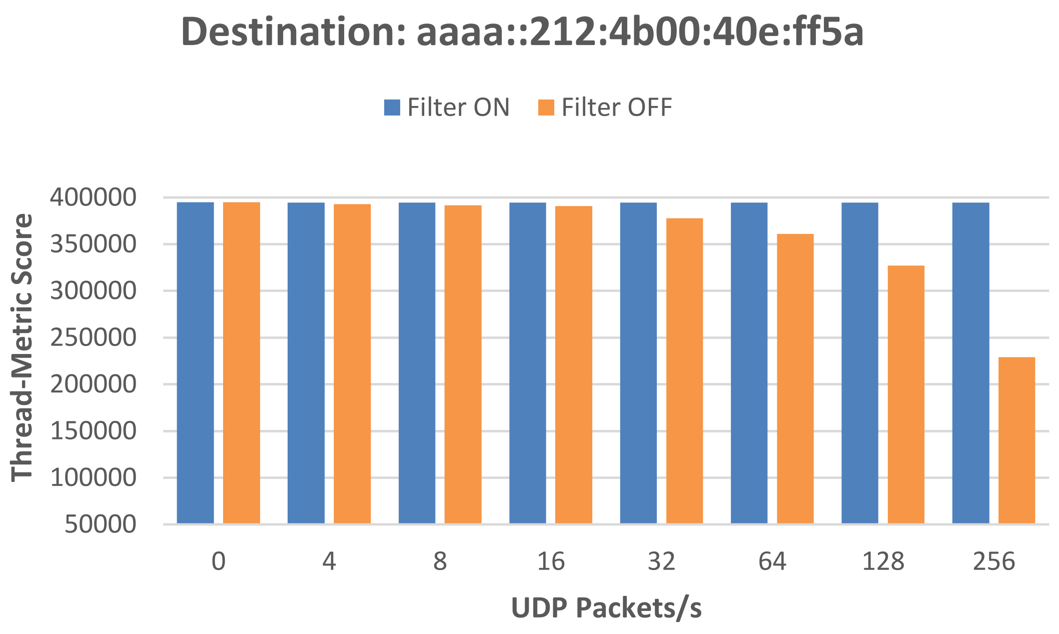


Figure 5.10: UDP Client sending packets to UDP Server 2.

UDP Client to UDP Server 2

Figure 5.10 shows the results obtained from running the benchmark with the accelerator filter features enabled and disabled, when the packets are sent from the UDP Client to the UDP Server 2, at different values of PSR. This experiment aims to evaluate the impact on the benchmark (that runs on the Server 1), when the packets are meant to be discarded (sent from UDP Client 1 to UDP Server 2). When the PSR value is 0, the score is also at its highest value, around 395046 for the 6LA enabled and disabled, since there are no packets being transmitted in the network. With the 6LoWPAN filter disabled, increasing the PSR tends to exponentially decrease the score, but at a lower rate when compared with the previous experiment. This is due to packets being received and dropped by the OS. Although these packets are meant to be discarded, the OS still has to flag an OS process to read and compute new packets. With the 6LA enabled, the score is not affected due to the packets being processed and discarded by the network accelerator. This represents full availability of the OS services when no packets are intended to be received and processed by the node. Comparing the benchmark scores, for the worst case scenario, with the 6LA enabled, the obtained score is about 394684 while with the 6LoWPAN filter disabled it is around 228947. This result represents an increase of nearly 42% on system availability.

5.4.3 Energy Consumption Analysis

To validate the solution in terms of energy consumption, the heterogeneous platform was evaluated and compared with a COTS solution, typically used in WSN and IoT applications. The performed evaluation focused on the energy needed for processing a single 6LoWPAN packet (to be accepted or rejected) on each platform, comparing the software solution with the 6LA, which is only available in the proposed heterogeneous architecture. Both platforms were configured to run at the same clock speed (32MHz) for their typical voltage supply (3.3V for the CC2538EM and 1.2V for the SmartFusion2 core), running the Contiki-OS with the UDP server application compiled under the same conditions (GNU Toolchain 5.4.1 with the optimizations flag -O2). When running the experiments, only the IPv6 filtering was handled and the filtering of the UDP source/destination ports was not performed. A full energy characterization of the developed platform and further comparison with related systems, goes beyond of the scope of this thesis and it is proposed as future work.

From the obtained results (presented in Table 5.2) when comparing only the software approaches, the energy required to process one 6LoWPAN packet is around 9445 nJ (processing time of nearly 210 μs) for the CC2538 and 14356 nJ for the SmartFusion2 (average processing time of 254 μs), with the 6LA disabled. This small divergence is related to the architectural differences of the CPU, as well as the overall differences on the SoC of both platforms. Moreover, for the overall power consumption, exploring the Flash Freeze control on the SmartFusion2, which en-

Table 5.2: Energy consumption analysis per received packet on different IoT low-end platforms.

Parameter	***Device***					
	CC2538 (Acc. N/A)		***SmartFusion2*** (Acc. OFF)		***SmartFusion2*** (Acc. ON)	
Compiler	ARM-GCC 5.4.1		ARM-GCC 5.4.1		ARM-GCC 5.4.1	
MCU Architecture	32-bit		32-bit		32-bit	
Instruction Set	Cortex-M3		Cortex-M3		Cortex-M3	
Clock (MHz)	32		32		32	
V_{CC} (V)	3.3		1.2		1.2	
Packet operation	Accept	Reject	Accept	Reject	Accept	Reject
$t_{average}$ (μs)	210	270	254	292	221.4	**0.40**
$P_{average}$ (mW)	44.98	44.98	56.52	56.52	54.95	**54.95**
$E_{average}$ (nJ)	9445	12144	14356	16503	12168	**22**

ables the FPGA to be disabled when not in use, can reduce the energy consumption required for processing received packets. The superiority of the proposed solution is emphasized when the 6LA (deployed on the RCU) is exploit on the heterogeneous architecture. When a 6LoWPAN packet is meant to be discarded, the processing time is nearly 0.4 μs, with an average energy consumption of around 22 μJ, which represents an energy reduction over 99%, if compared with the software processing of a rejecting packet. Despite the accelerator assisting the software processing, when the packet is meant to be accepted, the processing time is around 221 μs with an energy consumption of around 12168 μJ. This happens because Contiki-OS is not optimized to use hardware accelerators and the heterogeneous architecture does not take the benefits (at this stage of development) of the Flash Freeze control. Addressing these issues in the future will result in a better energy consumption on the heterogeneous architecture, when compared with the native software-only solution.

5.4.4 RCU Resources Utilization

Table 5.3 shows the synthesis results of the implemented network accelerator with MAC and 6LoWPAN filter modules connected. Results are expressed in terms of 4LUT and DFF used, for each deployed filter. The *network_acc_0* module is the most costly in terms of resources (4LUT and DFF) as it is responsible to interface the radio transceiver and the desired filter modules, as well as to accommodate the *IEEE 802.15.4 Frame Buffer*. Regarding the filter modules, 6LoWPAN filters are the most expensive in terms of FPGA fabric, mainly due to the implementation of the IPHC header processor. The *ip6_src_filter_0* resources (in terms of 4LUT) are less when compared with the *ip6_dst_filter_0* due to the synthesis tool optimizations. This

Table 5.3: Synthesis results obtained from Libero SoC v11.7.

	Module	***4LUT***	***DFF***
	network_acc_0	1916	1895
MAC	*pan_filter_0*	64	73
MAC	*mac_src_filter_0*	300	265
MAC	*mac_dst_filter_0*	159	265
MAC	*dfd_filter_0*	194	297
6LoWPAN	*ip6_src_filter_0*	159	521
6LoWPAN	*ip6_dst_filter_0*	1916	521
6LoWPAN	*udp_ports_filter_0*	1813	137
	Total (out of 86184)	6521 (7.57%)	3974 (4.61%)

happens because some resources, such as the registers that store the received IEEE 802.15.4 Data frame, can be shared between these two modules and thus, reduced to only one slot of resources instead of one for each module. For the remaining filter modules, the deployment costs are reduced.

5.4.5 ASIC Resource Estimation

Next generation of IEEE 802.15.4 transceivers can also benefit from the developed 6LoWPAN accelerator. Despite being deployed and tested on the RCU of a heterogeneous architecture, it can be integrated on an ASIC. As the FPGA technology is commonly used for prototyping and used as proof of concept prior ASIC implementations, it is possible to predict the hardware cost of the accelerator for ASIC deployments from the RTL design. The gate-level synthesis was conducted using the Synopsys Design Compiler for the AMS 0.35μm CMOS technology with a 32 MHz clock speed. The result from the synthesis tool regarding the 6LA RTL design, which evaluated the ASIC cost in terms of the number of standard cells (from Corelib library) needed, is 15718. This value indicates the extra cost of adding such accelerator to ASIC implementations of IEEE 802.15.4 radio transceivers.

5.5 Conclusions

This chapter described the 6LA, a co-designed hardware/software 6LoWPAN accelerator for filtering and processing IPv6 packets carried on IEEE 802.15.4 Data frames. Its building-block approach offers design-time scalability by easing the integration of new customized modules and features. The contributions are meant to be deployed on the developed heterogeneous architecture for IoT-based networks, which use an IEEE 802.15.4 compliant radio transceiver for data exchange and are mainly present on IoT low-end devices. The achieved results show that the software offloading allows a system's overhead reduction of nearly 13.24% while the packet discarding by the 6LA may reach a reduction of 42%, achieved when all the received packets are meant to be rejected. The performed evaluations highlighted the superiority of the 6LA (which reduced the energy consumption of processing received packets from 16503 μJ to nearly 22 μJ) when the packets are processed and discarded by the hardware. The contribution can also be deployed on future generations of IEEE 802.15.4 radio transceivers at a cost of 15718 cells for the 6LA.

As future work, new modules for handling security should be fully implemented and tested on the current heterogeneous architecture. Hereafter, since the accelerator was only tested with non-router devices, the implementation should be tested and included in a router. In IP-based networks, where the packets are mainly forward to intermediate nodes until they reach the destination, the accelerator should also explore the implementation of an RPL module in order to forward packets to their destination without the OS intervention. Furthermore, while security has emerged as a major design goal for smart connected devices, future work will not only focus on securing communications data (through the 6LoWPAN extension to IPsec), but also when in processing and at rest. To achieve that, the applicability of recent work, the IIoTEED [49], to resource-constrained low-end devices, supported by the new generation of ARM Cortex-M TrustZone-based processors, must be researched.

As the system complexity increases, the design and time-to-market metrics tend to decrease. Despite using a building-block approach, adding features to the proposed 6LA will later add an extra developing cost in terms of design-time scalability and software verification. Promoting system-design automation and reconfigurability to the system through a DSL, will contribute for a fast and better development. These topics are next discussed in Chapter 6.

Chapter 6

Enabling Design Automation Through a Modeling DSL

With the increased complexity of the IoT low-end devices, mainly due to their connectivity and interoperability requirements, the development and configuration of an embedded OS for such devices is not straightforward. The complexity of the communication requirements is usually mitigated by the OS, as it already incorporates an IoT-compliant network stack. However, the configuration of such stack requires major knowledge on the code structure, leading to additional development time, particularly when the network comprises several wireless nodes and individual configurations, with subsequent firmware functionalities to be generated.

This chapter presents the EL4IoT framework, a DSL for embedded OSes that targets IoT low-end devices. It aims to reduce the development time by promoting a design automation tool that can configure and generate Contiki-OS code, ready to be compiled. Although leveraging the Contiki-OS architecture model, this work only re-factored and modeled the network stack, approaching the OS itself as one big component. The proposed DSL can also be extended to other embedded OSes or it can be integrated in other design automation tools. Section 6.1 introduces this chapter, while Section 6.2 presents the Elaboration Language (EL), going through the framework's workflow and language constructs. Section 6.3 presents the modeling of the Contiki-OS network stack following a composite design pattern, and Section 6.4 illustrates the implementation of a component and the EL4IoT internals. The system evaluation is discussed in Section 6.5 and Section 6.6 concludes the chapter.

Related Publications

Ideas and findings presented in this chapter resulted in the following publication:

- **T. Gomes**, P. Lopes, J. Alves, P. Mestre, J. Cabral, J. L. Monteiro, and Adriano Tavares, "*A Modeling Domain-Specific Language for IoT-enabled Operating Systems,*" IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, 2017.

6.1 Introduction

Existing IoT embedded OSes, e.g., Contiki-OS, mainly aim to promote a lightweight implementation of a standardized network stack for the low-end devices. However, their configuration and deployment is still complex, mainly due to the devices' hardware heterogeneity and the high variability of the OS and network stack. The task of configuring and customizing network parameters, such as the PAN value and device's addresses (MAC and IPv6), the OS services and protocols, e.g., 6LoWPAN and CoAP, can be mitigated by enabling design automation through the development of a tool that allows full system configuration and code generation, according to user needs and application requirements. The applicability of such tool can be explored in a way that generating firmware for several nodes in an IoT network, while providing mechanisms for code verification and validation, can be performed by automated systems and/or embedded systems designers without deep knowledge of the OS and/or the network stack.

Several approaches targeting design automation by providing DSLs to model a desired system have already been undertaken in the recent years. In [62] is proposed a low-level DSL for dynamic code-generation in binary translation systems, enabling code snippets to be added during compile time. The code is then generated by the translator on demand at runtime and integrated into the translated application code. Targeting the development of FPGA-based systems, several well-known DSLs have been proposed to smooth the system design both at the software and RTL layers [63]. These DSLs aim to simplify the development of building hardware/-software co-designed systems, which requires high levels of expertise on different domains. SensorScript [64] proposes a business-oriented DSL for sensor networks, that aims to provide a model that avoids to overwhelm any user with all the data gathered from the sensor network, regardless if it is actually required by the user. This allows users to easily search, query and aggregate information from the available sensors. Targeting IoT-based applications and aiming to relieve designers from the complexity and the heterogeneity of the WSN nodes, the DSL-4-IoT [65] was recently proposed. It provides a visual model based language that, using a high level of abstraction, allows different node configurations to be deployed over a WSN environment. However, the level of abstraction provided DSL-4-IoT does not specify the low-level configurations such as the network settings (e.g., addressing and radio channel parameters) and protocols in use.

This chapter presents the EL4IoT framework, a modeling DSL for IoT-enabled OSes. The EL4IoT allows the configuration and automatic generation of code for low-end devices in IoT applications that require an IoT-compliant network stack to provide interoperability and seamless connectivity to the Internet. The main contributions of the EL4IoT are: (1) a DSL for modeling embedded OSes targeting the Internet of Things (Contiki-OS); (2) reduced modeling efforts over 4 layers of the Contiki-OS network stack; (3) the development of a design automation and code verification tool for embedded systems designers, promoting its integration with design automation tools.

6.2 DSL for IoT Endpoint Devices

A DSL is a programming language with limited expressiveness which, in contrast with general purpose languages, targets a specific domain by providing constructs to solve its specific problems [59]. An ideal DSL allows specifying what is required to be computed, while relegating its implementation to automated tools. Its usage is quite appealing since it promotes a simpler and faster development, while providing higher gains in expressiveness, ease of use and productivity [60]. Developing a DSL is quite hard, since it requires high levels of domain knowledge and technical expertise. However, once well designed and created, it tends to pay off all the inherent development efforts [61]. There is a growing interest in DSLs for generative programming (GP) [189] and model driven development (MDD) [190] programming styles, as both provide higher levels of abstraction, leveraging software reuse and fast software development. While GP targets the automatic system generation according to a defined specification [189], MDD is an approach in which extensive models are created (before, during or after source code is written) to describe system's architecture abstracting implementation details, easing development and testing purposes. Both MDD and GP rely on software reuse and complex code generation. Thus, a DSL must provide constructs to enable the mapping between models and code that will be generated [191].

The EL is a modeling DSL designed to be an efficient GP tool, while approaching MDD. Based on the Service-Component Architecture (SCA) standard, it mainly targets the code generation automation from the source files of a designated system. SCA specifies that various system components may be assembled by the form of service-oriented architecture (SOA) components, following a composite pattern [192].

The key concepts of the SCA standard are: *Composite*, *Component*, *Service*, *Reference*, *Property* and *Wire*. According to the standard, it is possible to create reference architectures identifying the system components and interactions between them, as well as their configurable properties. The EL was developed using the Xtext and Xtend frameworks, widely adopted when developing a DSL [59]. Xtext is an Eclipse framework used to create the language grammar, which dictates how the parser and the Abstract Syntax Tree is created, while Xtend is a general-purpose programming language that is translated into a comprehensible Java representation [59]. It is interchangeable with Java code and used to implement language validators, code generation software, and some other Eclipse language specific tools (e.g., quick fixes). In this chapter the aspects of developing a DSL are not covered. The chapter aims to technically present the EL as a solution for software modeling, validating its usability through a real use case, i.e., the modeling of the Contiki-OS communication stack. The following sections will cover EL's workflow and framework architecture, in addition to its constructs and rules.

6.2.1 EL4IoT - EL Framework Overview

The framework workflow, depicted in Figure 6.1, encompasses four main stages: *Modeling*, *Elaboration*, *Configuration* and *Code Generation*. During each stage, the artifacts that will be used in the next stages are created. During the *Modeling* stage, the main goal is to create an architectural model, according to the SCA standard, that will be later used as a reference architecture. During the architectural model creation, system's components must be identified, as well as the dependencies between them, allowing the specification of well-defined interfaces and properties. After its creation, it must be compiled. If the compilation process succeeds, the compiler should generate, for each component, its Java representation, elaboration stub class, the configuration XML files, and an architecture-specific Java Elaborator. All these artifacts are then used in the following stages.

The *Elaboration* stage encompasses the provision of annotated source files and the implementation of the elaboration classes' behavior (using the elaboration stub classes). Once implemented, elaboration files dictate how the source code must be generated. For each component, more than one implementation may be available, as well as its respective elaboration. Only one elaboration class per component will be executed in the Elaborator, as specified by the configuration files. Also, an API is available to ease the annotation process (find and replace), within the respec-

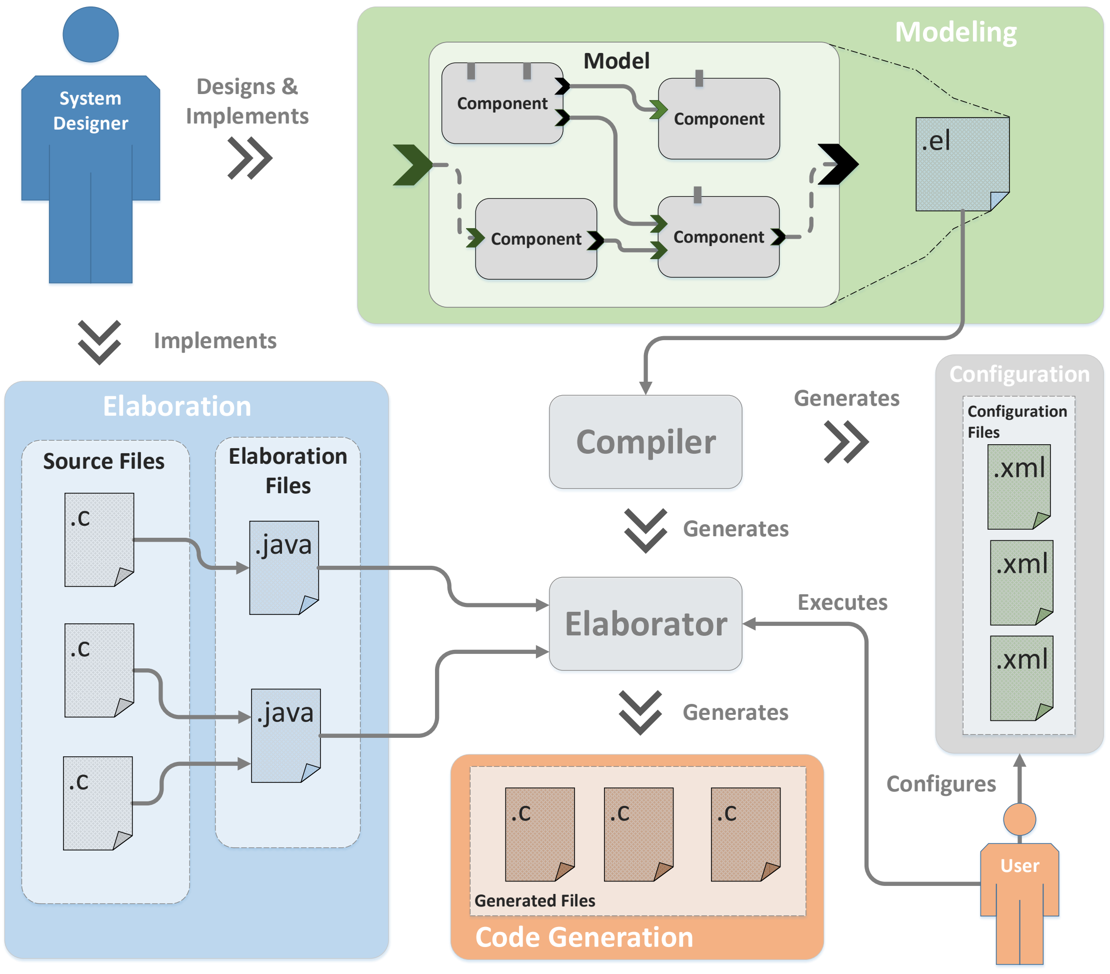


Figure 6.1: Framework workflow.

tive source code files (e.g., function calls, property values), and also to fetch other properties from the configuration files.

Previously generated XML artifacts (configuration files), contain the values for all the component properties, which may be changed during the *Configuration* stage, to modify the system configuration and its subsequent code generation. These files also specify which component elaboration will be loaded into the Elaborator process. In addition, each elaboration may have its own implementation-specific properties. Since such properties are not available in the reference architecture's configuration files, another XML file should be provided by the elaboration developer. Once properly configured, the generated elaborator must be executed. This process will fetch each component's properties and will load elaboration classes through Java reflexion, according to which rules are specified in the XML files. As the result, the *Code Generation* should be according to corresponding Elaboration Classes.

Three different actors, which interact with the system at different stages of the code generation process, are identified:

- The Architect: the individual with technical knowledge and specific domain expertise, that is responsible for translating system characteristics into a model;
- The Component Designer: the one with technical expertise, which provides the annotated source code files and implements the elaboration classes.
- The End User: the final user that will benefit from the provided resources to configure and generate application-specific code. Usually, the end user only focus on setting up the configuration files (defining properties values and choosing the elaboration file to be imported) before invoking the Elaborator.

6.2.2 EL's Constructs and Operations

As previously said, the EL is a DSL that allows the description of an architectural model according to SCA specified concepts. An EL file, depicted in Listing 6.1, is defined with the .el extension and contains three types of top-level constructs: (1) components, (2) interfaces and (3) language descriptions. Each component is described as an aggregation of subcomponents, properties, and its relations with other components in the form of services and/or references. It is also described as having one language type, which should be its own implementation language.

Listing 6.1: EL code snippet from the component *sixlowpan*.

```
final ("sixlowpan.java") component sixlowpan (C){
    subcomponents:
        fragmentation fragmentator
        compression compressor
        sixlowpan_out output
        sixlowpan_in input

    promote service input.S_in as S_in
    promote service output.S_out as S_out
    promote reference input.R_tcpip_in as R_tcpip_in
    promote reference output.R_sp as R_sp
    bind input.R_decomp to compressor.S_d
    bind input.R_defrag to fragmentator.S_d
    bind output.R_comp to compressor.S_c
    bind output.R_frag to fragmentator.S_f
}
```

Table 6.1: Available EL's keywords.

Keyword	***Description***
`annotation`	Defines the character that limits the annotations.
`as`	Renames a promoted reference or service.
`bind`	Binds a reference to a service.
`bool`	Component's property data type.
`compile`	Tells to compiler which is the top level component.
`component`	Defines a component.
`final`	Defines that a component has a concrete elaboration.
`import`	Imports the content of the specified file.
`int`	Component's property data type.
`interface`	Defines a set of functions used by a service or pointed by a reference.
`is`	Inherits the specified component.
`float`	Component's property data type
`language`	Defines a language.
`promote`	Promotes a reference or service from a subcomponent to a component.
`properties`	Defines the properties set of a component.
`reference`	Defines the reference used in a promote or in a bind operation.
`references`	Defines the references set of a component.
`restrict`	Restricts the values that a property can take to a user's defined set.
`service`	Defines the service used in a promote or in a bind operation.
`services`	Defines the services set of a component.
`string`	Component's property data type.
`subcomponent`	Defines the subcomponents set of a component.
`to`	Connects a reference to a service in a bind operation.

The **Language construct** specifies the implementation language of components (e.g., C, C++ or Python), imposing restrictions to a given component, which can only relate with others described in the same language.

The **Interface construct** describes interfaces for which other components will associate. An interface declaration must always state the services it can provide. Components connect through bindings of services and references that follow the same interface type, using the keyword `bind`. Table 6.1 contains all the keywords provided by the EL DSL. For instance, in Listing 6.1 (line 12), there is a binding between the service *S_d* from the component *compressor* and the reference *R_decomp* for the component *sixlowpan*.

The **Component construct** is the most important and it consists of four sections:

- Properties: EL enables properties declaration for the basic programming types (i.e., char, int, string), in which it is possible to impose restrictions and to perform assignment operations.
- Subcomponents: Where the aggregated subcomponents are stated.

- Services and References: Here, interfaces are instantiated as services or references. The interfaces belonging to the Services section are those implemented by the component, while the ones inherent to the references section are dependencies from a given component that implements that interface. Services and references must always be connected through binding operations.
- Free: In this section, assignments can be made to properties where subcomponent interfaces are binded and/or promoted. This is done (either for services or for references) by using the keyword `promote`. It states that a subcomponent interface is going to be available in the top-level component, and it must be resolved later with a bind operation with the top-level component. Other important keyword is the `compile`, which states the top-level component on the hierarchy, from where the compilation process is started (in a top-down approach). This also defines the order of the classes invocation in the Elaborator program.

6.3 Modeling the IoT Stack

Modeling the Contiki-OS network stack (μIP stack) is not straightforward. Due to its complexity, it requires a high level of expertise on IoT-enabled network stacks for low-end devices, as well as knowledge on the OS itself. The μIP stack is composed by tightly coupled components, which must be priorly identified by performing source code analysis aided by the respective simulation. The identification of component interfaces and their interactions (through function invocation) requires a deep understanding of the stack implementation and its behavior. The first steps encompass the creation of a reference architecture of the network stack, followed by a comprehensive description of abstraction, lowering from the model to source code, and extending as well the Contiki-OS stack documentation.

Figure 6.2 depicts the top-level view of the resulting model, comprising the top-level components in a composite model, where each block refers to a component with well-known denomination: (1) MAC, (2) 6LoWPAN, (3) IPv6, (4) Transport Layer and (5) UDP/TCP Sockets for the application layer. Such view is expected since the network stack its being modeled, prior further development. The purple and green polygons define references and services, respectively, in a SOA approach. Semantically, the connectors between references and services denote the establishment of a function call dependency between components, where green polygons provide

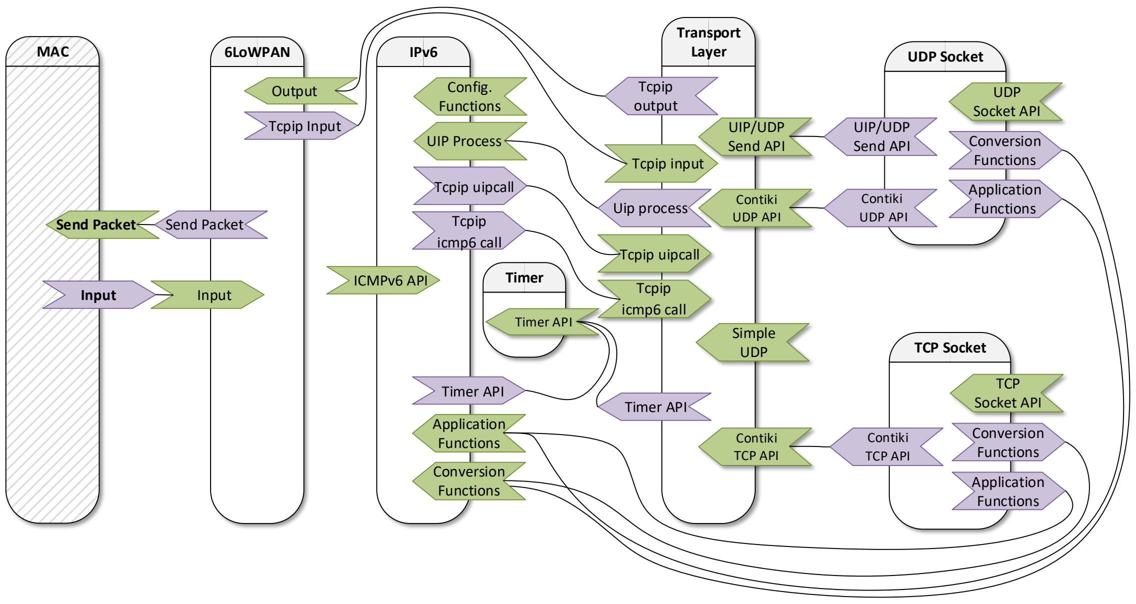


Figure 6.2: High-level composite model of the Contiki-OS network implementation.

the services required by references. The proposed model follows a layered approach, where top components are formed by other components as well. This approach inherently provides a changeable level of abstraction, depending on the embedded designer needs.

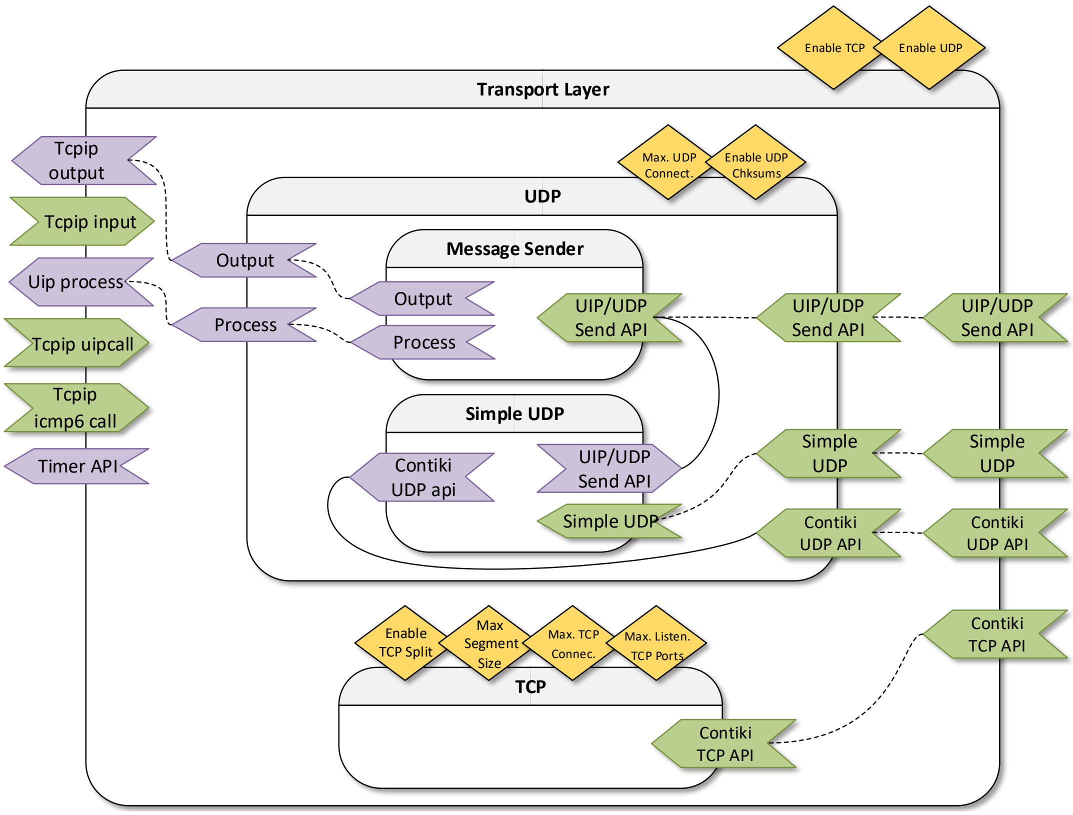


Figure 6.3: Internals of the Transport Layer component.

For the sake of simplicity and explanation purposes, only the *Transport Layer* and its internals are depicted in Figure 6.3. This component is composed by a UDP and TCP subcomponents, with their respective interfaces and properties, where the dashed lines depict the promoted relationships. In this example, it is only provided visibility to the internal services and references from outside the *Transport Layer* component. Logically, it uses all the modeled OS services available (to applications) and, therefore, encompasses all its available references. Properties are application-specific and cannot be represented in the reference model.

6.4 Implementation

After conceiving the model, it is translated to the EL DSL. Its code representation allows the generation of the supporting software, which consists on a framework that accesses model properties and interfaces during the final stage of code generation. This framework aims to promote design automation with code generation, leaving only the components selection and properties configuration, e.g., PAN address and TCP/UDP parameters, to user's choices.

Elaboration Properties

The EL seamless code generation process allows a transparent system configuration. The modeled system properties are defined in a configuration XML file, according to the user and application requirements. While EL avoids the traditional need for developers to write code, the generation of the final OS source files according to the model definition and configuration must be automatic and user-friendly. As aforementioned, EL uses software annotations, embedded in the OS source files through a marker defined by the meta-character `@@`.

Listing 6.2: UIP_CONF_TCP and UIP_CONF_UDP annotations.

```
//======================== Transport ========================
//----------- TCP
#define UIP_CONF_MAX_LISTENPORTS  @@UIP_CONF_MAX_LISTENPORTS@@

//----------- TransportLayer
#define UIP_CONF_TCP              @@UIP_CONF_TCP@@
#define UIP_CONF_UDP              @@UIP_CONF_UDP@@
```

This pattern is used to help in finding code where the annotation is later replaced by its corresponding value during the code generation process. Listing 6.2 depicts a code snippet with `UIP_CONF_TCP` and `UIP_CONF_UDP` annotations, which are used to enable TCP and UDP protocols, associated with the `Enable TCP` and `Enable UDP` model properties, respectively. The `UIP_CONF_MAX_LISTENPORTS` is defined by the TCP component inside its parent component (Transport Layer), as depicted in Figure 6.3, leveraging the ability of internal components to define their own properties.

The elaboration of components incorporates the logic of their respective code generation process (i.e., annotation substitutions) in the form of JAVA code. The component Transport Layer only contains the original Contiki-OS implementation, embedded in an elaboration. Further elaborations for this and other components of the stack are still under development. Listing 6.3 denotes the logic associated with the Transport Layer component elaboration.

Listing 6.3: Elaboration method of the Transport Layer component in Java.

```java
openAnnotatedSharedSource("contiki-conf-gen.h");

if(target.get_Enable_TCP())
    replaceAnotation("UIP_CONF_TCP", 1);
else
    replaceAnotation("UIP_CONF_TCP", 0);

if(target.get_Enable_UDP())
    replaceAnotation("UIP_CONF_UDP", 1);
else
    replaceAnotation("UIP_CONF_UDP", 0);

openAnnotatedSource("tcpip.c", "./contiki-3.0/core/net/ip");
openAnnotatedSource("tcpip.h", "./contiki-3.0/core/net/ip");
```

Briefly, a header file, that shared between several components, is opened and the `UIP_CONF_TCP` and `UIP_CONF_UDP` annotations are replaced by the values defined in the model, retrieved from the configuration XML file. Next, the source code of the implementation is generated in the final directory by calling the "openAnnotatedSource" method. The new generated file will be create after the annotations' replacement. All these functionalities are provided by an API, which eases the elaborator's implementation. The generate method is automatically invoked by the framework, while a fully configured and ready-to-compile Contiki-OS stack is being generated. For the example given by Listing 6.2, the resulting annotated file (generated by the Elaborator) is depicted by Listing 6.4.

Listing 6.4: Annotated file with `UIP_CONF_TCP` and `UIP_CONF_UDP` configuration directives.

```
//======================= Transport =======================
//----------- TCP
#define UIP_CONF_MAX_LISTENPORTS  8

//----------- TransportLayer
#define UIP_CONF_TCP              0
#define UIP_CONF_UDP              1
```

The TCP protocol is set to be disabled (`UIP_CONF_TCP = 0`) while the UDP protocol is configured to be enabled (`UIP_CONF_UDP = 1`), with a maximum number of eight UDP listening ports (`UIP_CONF_MAX_LISTENPORTS = 8`).

Elaboration Interfaces

Contrary to the "Properties" of the code generation (which is entirely a model-based process), interfaces use components' elaborations to provide services (as function calls) to connected references. That is to say, different components' elaborations for the same component might provide distinct implementations for the same service, represented by several functions calls. The implementation of Interfaces, including argument's meta-data passing, is still under development.

6.5 Evaluation Tool

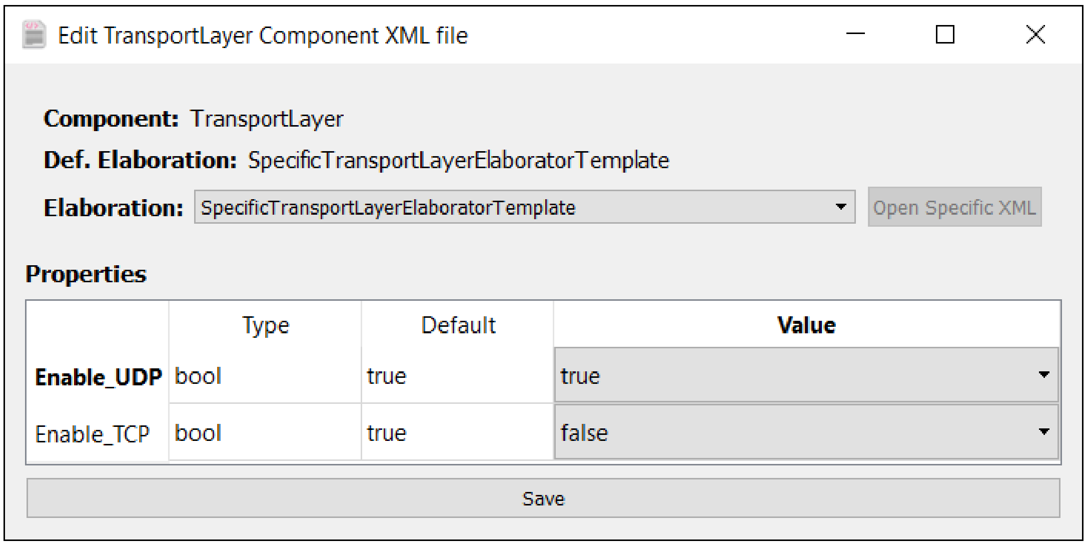
The Contiki-OS network stack model is used to demonstrate the flexibility provided by the modeling tool to automatically generate a full configured system ready to be compiled. While the reference model provides properties' abstraction from implementations by using components, easing the system configuration to users, it requires the specification of a real implementation by the system designer. Listing 6.5 denotes the XML configuration file for the Transport Layer component. The implementation of this component is provided by the `MySpecificTransportLayerElaborator`. Each Transport Layer sub-component has its own implementation. For instance, by disabling the TCP protocol, the final generated Contiki-OS code will not contain TCP related code.

Listing 6.5: XML configuration file for the Transport Layer component.

```xml
<component type="TransportLayer">
  <elaboration default="SpecificTransportLayerElaboratorTemplate">
  MySpecificTransportLayerElaborator
  </elaboration>
  <properties>
    <property type="bool" name="Enable_UDP" default="true">
      <value>
        <element></element>
      </value>
    </property>
    <property type="bool" name="Enable_TCP" default="true">
      <value>
        <element>false</element>
      </value>
    </property>
  </properties>
</component>
```

Figure 6.4 illustrates the XML component editor window that allows the end-user to easily edit the configuration XML files. In this example, two options are available for changing the `UIP_CONF_TCP` and `UIP_CONF_UDP` parameters, whose values will later replace the respective annotations during the code generation process. For each property, the application restricts such inputs, according to their type (defined by the EL keyword `type`) or possible range (set in the field *Value*), avoiding wrong user inputs, while keeping the stack functional when the values are not set.

The same procedure is used to specify the implementation of the `Application` component to be a UDP Server. This implementation provides its own configuration XML file, allowing the designer to specify implementation-related properties as well. In both XML files, every property requires a default value to seamlessly generate the final (compilable) Contiki-OS source code. The UDP server application is configured to create a connection with any UDP client, listening on the UDP port 4101 (instead of the default port 3001). The default response given by the server to any client was not changed by the tool.


Edit TransportLayer Component XML file

Component: TransportLayer

Def. Elaboration: SpecificTransportLayerElaboratorTemplate

Elaboration: SpecificTransportLayerElaboratorTemplate

Properties

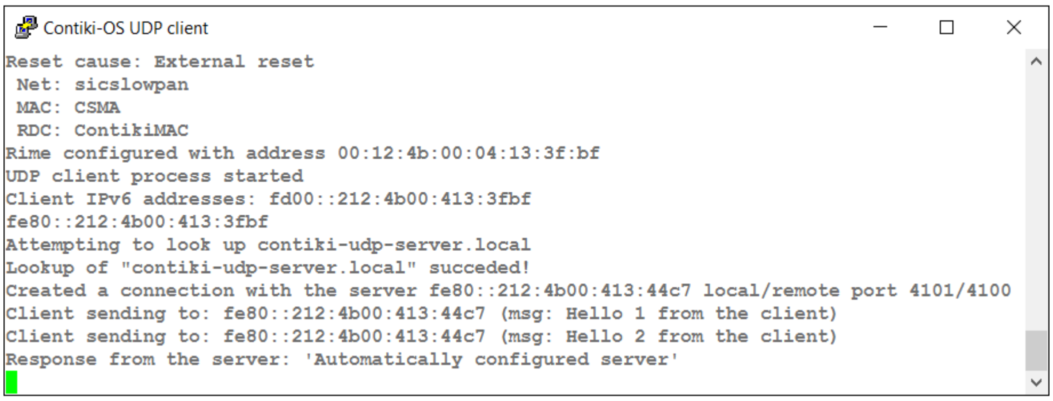
	Type	Default	**Value**
Enable_UDP	bool	true	true
Enable_TCP	bool	true	false

Figure 6.4: XML component editor with the Transport Layer component.

While in this example, Listing 6.6, the server is configured to be listening on UDP port 4101 (TCP related code was disabled from compilation), several distinct configurations can be achieved by using other model properties (not depicted in this test). After its generation, the source code for the UDP server was compiled and the resulting firmware was deployed on a CC2538EM board from Texas Instruments.

Listing 6.6: UDP Server application specific XML configuration file.

```xml
<component type="Application">
    <properties>
        <property type="int" name="LocalPort" default="3001">
            <restriction type="range">
                <botValue>1</botValue>
                <topValue>5000</topValue>
            </restriction>
            <value> <element>4101</element> </value>
        <property type="string" name="Message" default="Automatically
    configured server">
            <value> <element></element> </value>
        </property>
        <property type="string" name="Hostname" default="contiki-udp-server">
            <value> <element></element> </value>
        </property>
    </properties>
</component>
```



```
Contiki-OS UDP client
Reset cause: External reset
 Net: sicslowpan
 MAC: CSMA
 RDC: ContikiMAC
Rime configured with address 00:12:4b:00:04:13:3f:bf
UDP client process started
Client IPv6 addresses: fd00::212:4b00:413:3fbf
fe80::212:4b00:413:3fbf
Attempting to look up contiki-udp-server.local
Lookup of "contiki-udp-server.local" succeded!
Created a connection with the server fe80::212:4b00:413:44c7 local/remote port 4101/4100
Client sending to: fe80::212:4b00:413:44c7 (msg: Hello 1 from the client)
Client sending to: fe80::212:4b00:413:44c7 (msg: Hello 2 from the client)
Response from the server: 'Automatically configured server'
```

Figure 6.5: A simple UDP client connecting and exchanging messages with the automatically generated UDP Server.

This Evaluation Module (EM) was used in a simple client-server setup in order to test the developed tools. The server, using the same EM, is running a UDP client application that periodically exchanges messages with the automatically generated and configured, by the tool, UDP server. As it can be seen from Figure 6.5, which depicts the UDP server application output, received by the UDP client application. The client connects the specified UDP server ports and receives the default message, which is printed to the EM output.

6.6 Conclusions

This chapter presented the EL4IoT, a modeling DSL for embedded OSes that targets the low-end devices in the IoT. The Contiki-OS network stack was modeled and a DSL was developed, in order to allow the whole OS description (mainly its companion network stack) with a variable abstraction level supported by the composite pattern. The developed framework aims to promote design automation, mitigating the configuration task of the Contiki-OS when deploying an IoT-compliant network through automatic code generation. For the proof of concept, the Contiki-OS network stack was modeled into the DSL and, using the framework, generated a UDP server application, easily (re)configured without manually changing the source code. The performed evaluation show that, despite being hard to create a DSL for a specific domain, the modeling efforts tend to pay-off. With the increasing level of complexity and variability of IoT systems, these tools result in higher productivity and lower development time.

The benefits from such approaches in developing embedded software are endless. Hereafter, the EL4IoT framework should get some improvements, namely the code generation tool, with a more user-friendly interface that allows code generation and system configuration with reduced number of inputs from the user. Other features can be added to this tool, such as the automatic creation of the final firmware from the generated code. Currently under the development, the DSL is being improved with the usage of semantic technology to describe the domain knowledge, leveraging its development towards a model validation approach and reducing the elaboration development efforts. This will also contribute in improving the system scalability, as semantic technology can upgrade code generation and verification when used to guide the development environment.

Chapter 7

Conclusions and Future Work

In a world where everything is getting connected, device security and device interoperability are a paramount. This triggers several technological issues towards connectivity, interoperability and security requirements of IoT devices, from the sensor to the cloud. Regarding endpoint devices for IoT, this thesis proposed to tackle several aspects that aimed to contribute for an accelerated, secure and energy-aware end-device communication.

This chapter presents and discusses the conclusions that could be drawn from the results obtained by this thesis. The remaining of this chapter is organized as follows: Section 7.1 concludes the work developed throughout this thesis, while Section 7.2 discusses the identified limitations. Section 7.3 points some directions for future work and proposes a research roadmap to achieve it. Finally, Section 7.4 summarizes the list of publications that have directly or indirectly contributed for the realization of this thesis.

7.1 Conclusions

Pervasive computing and communication technologies of IoT have brought many challenges when developing wirelessly connected devices, especially at the network edge. Challenges on the embedded devices are due not only to the ever-growing amount of data to be handled, but are also due to the increasing importance of security and privacy requirements. Thus, tackling them is not straightforward. The need for energy-efficient wireless devices to connect the Internet seamlessly, while preserving their performance and security capabilities, demands for new architectural solutions at the hardware level. While the connectivity exposes the device to the Internet, which raises several security issues, deploying a standardized communication stack on endpoint devices highly increases the data exchanged over the network. Due to the inherent complexity of developing IoT systems, even at the network edge, this work was motivated by the broad question posed at the beginning of this thesis,

How to leverage an accelerated, secure and energy-aware IoT end-device communication?

to which several subquestions were addressed and a proper methodology proposed.

First and second questions concerned with the traditional low capabilities of endpoint devices, which could be enhanced by deploying acceleration blocks in heterogeneous hardware solutions. This was performed after identifying the best candidates to be offloaded from software and deployed under hardware accelerators. To tackle these questions, and following the proposed methodology, Chapter 3 contributed with a heterogeneous architecture for endpoint devices in the IoT network. The architecture, integrating a secure hardware solution with a low-power MCU and an RCU beside an IEEE 802.15.4-compliant radio transceiver, resulted in the CUTE mote, which is described as a customizable and trustable IoT end-device. Following a hardware/software co-design, the solution has proven to be a great choice for deploying not only application-specific hardware blocks, but also an accelerated network stack in a secure hardware platform, provided by the SmartFusion2 SoC.

The third question related to the endpoint communication improvements in low-end devices, for which a hardware/software co-design methodology was identified as the best approach to identify, from the OS network stack, the best offloading candidates. This resulted in the contributions on Chapter 4 and Chapter 5, which presented, respectively, hardware-accelerated solutions for the MAC sub-layer and the Network layer on the communication stack. Such improvements aimed to contribute with accelerators directly coupled to the MCU as standard memory mapped soft-hardware peripherals, which could be easily accessed through standard communication buses. The benefits of such accelerators revealed to be highly important, as they are able to contribute to an increased OS performance and opportunity to handle other tasks, even in high network traffic situations. The MAC sub-layer accelerator presented in Chapter 4, was built as the foundation for the 6LoWPAN accelerator, which is presented in Chapter 5, providing a performance increase of nearly 17% for the processing and filtering tasks, while the 6LoWPAN accelerator contributed with a performance increase of nearly 13% for the handling of the IPv6 packets extracted from the IEEE 802.15.4 Data frames.

Since device and data security play a key role when connecting low-end devices to the Internet, the fourth question could be answered with the contributions described on Chapter 3 and Chapter 5, where not only the network performance is tackled, but also the device and data security mechanisms are addressed. By employing IPsec security to the device communication through the usage of Compressed IPsec in Contiki-OS, the device can highly benefit from hardware acceleration for increasing the performance of such heavily computational algorithms. Moreover, with the increasing concern on the device security through communication's vulnerabilities, traditional IDS systems are being approached to endpoint devices as well. Such systems can take great benefit from the contributions from Chapter 5, where the accelerated and customized filtering features can be set available to any rule applied in the IDS. Device security starts from the silicon manufacturing phase, and that could be achieved by selecting a proper SoC, which is provided by the SmartFusion2, as discussed in Chapter 3.

Finally, for the fifth question, which raised concerns about the engineering efforts for creating secure, green and fast endpoint devices at the network edge, Chapter 6 contributes with a DSL that helps promoting and facilitating design automation tools. Such tools can contribute for a fast and reliable development of different accelerated solutions to individuals without expertise in the field. The benefits of developing DSL for embedded systems are endless. Well combined with code validation, simulation and firmware generation tools, they contribute for a more efficient and rapid development of endpoint devices for IoT.

7.2 Limitations

Despite the contributions provided by this thesis to the state-of-the-art, this work presents some limitations, which are expected to be overcome in the near future. Such limitations are identified as:

- Power consumption issues are still evident as, when in the normal operation mode, the CUTE mote consumes more power for the accepted IPv6 packets. The solution only takes benefit of the accelerated processing, in terms of power, when the IPv6 packets are rejected. Despite supporting the Flash Freeze low-power mode, the CUTE mote does not yet integrate a complete DPM system. A full power characterization is still pending and dependent of a real application scenario;

- As RT support is seen as a major requirement in most IoT applications, the CUTE mote can be considered limited in this sense. However, RIOT and Contiki-OS with RT support are scheduled to be soon integrated and supported by the CUTE mote. Still regarding the OSes, since they mainly target resource-constrained devices, their network stack support is sometimes tightly coupled (in a cross-layer approach) with services and OS internals, thus it is hard to remove the full dependency of some network tasks and functionalities from the OS. In Contiki-OS, this hampers the task of software offloading, resulting in some processing tasks being still repeated by the OS.

- Regarding the network accelerators, both for the IEEE 802.15.4 sub-layer and the 6LoWPAN adaptation layer, several improvements are still needed. With the recent features provided by the IEEE 802.15.4e standard, some energy and security-related issues are being addressed and could also be integrated into the CUTE mote. Concerning the 6LoWPAN, and since this layer comprises several protocols and services (rather than just the header compression mechanisms), several functionalities could benefit also from the hardware accelerator. As thoroughly stated during previous chapters, the 6LoWPAN accelerator could include the following functionalities: (1) full hardware support for the Compressed IPsec (cryptographic algorithms and the protocol) to control and handle the security when protected IPv6 packets are received; (2) support for other services and protocols that rely on the 6LoWPAN, such as the Neighbor Discovery protocol, a CID management algorithm and the RPL routing capabilities.

- For other layers, such as the Application layer, all the tasks and protocols, e.g., CoAP, are still performed in software. This solution lacks in the support of other standards and protocols that could benefit from acceleration modules. Other OS services, at the application level, could also take advantage from the developed accelerators, such as the Suricata IDS application. Although a smooth integration could be achieved, this is not yet integrated in the CUTE mote.

- Concerning with the developed framework used for configuring and generating source-code ready for compiling, code verification and automatic firmware generation are still not integrated with the developed framework, thus some hand-work is still needed by the system developer.

7.3 Future Work

Despite all the contributions developed and provided throughout the realization in this thesis, future work is still addressed and proposed by the forthcoming research roadmap:

- **Provide full RT support**: With the increasing requirements for RT support on the endpoint devices, the efforts of including other embedded OS rather than Contiki-OS, e.g., RIOT, will contribute for a heterogeneous endpoint device with RT capabilities;

- **Remove full dependency between stack-related tasks and the OS internals and processes:** Due to optimization purposes, some Network tasks are tightly coupled with Contiki-OS internals. Therefore, when developing hardware/software co-designed solutions, this difficult the task of offloading software components to hardware. Despite being a complex task, future work could comprise the refactoring of Contiki-OS source code in order to provide a hybrid solution, both with hardware and software support, which could be utilized accordingly. Concerning the network stack and the work developed in this thesis, this would result in removing the redundant processing in Contiki-OS of some tasks, which are already performed by hardware accelerators.

- **Support other 6LoWPAN protocols and services through hardware-acceleration engines**: Since the 6LoWPAN is the de facto standard for supporting the IPv6 protocol in WPAN and LLN, other features must be analyzed and offloaded to hardware, contributing for even more green, efficient and accelerated endpoint device communication. Such features comprise the full hardware support to Compressed IPsec security, a dynamic CID management algorithm, and RPL routing capabilities.

- **Integrate into the CUTE mote an IDS system**: Suricata, which was already tested with Contiki-OS and 6LoWPAN, could resort hardware accelerators and activate, on-the-fly, different and customized filtering features according to the dynamically established rules. This could contribute for a better DoS attack prevention from many and different attack sources/types, such as UDP flooding and port scanning probes.
- **Expand the usage of the proposed solutions to middle- and high-end devices**: Even devices with more performance and high processing capabilities, such as border-routers and gateways, where routing capabilities and technology translation are the most performed task, could benefit from the proposed solution. Accelerating such tasks would enhance the performance, security and interoperability of all devices in the 6LoWPAN network.
- **Expand the functionalities and applicability of the EL4IoT framework**: Despite being used mainly for code generation of the Contiki-OS, the developed DSL, which followed the MDD approach, can also easily be targeted toward test automation. Simulation-oriented artifacts can be extracted and/or implemented based on behaviors of known simulation tools, such as Cooja offered by Contiki-OS. In doing so, it contributed to a better endpoint system development. Although the Cooja simulator does not provide API for interfacing with external tools, the source-code is set available in open-source, therefore, its integration with the EL4IoT framework is still possible.

7.4 List of Publications

The work developed throughout this thesis has contributed to the following publications (both journal and conference proceedings):

- **T. Gomes**, J. Brito, H. Abreu, H. Gomes and J. Cabral, "*GreenMon: An efficient wireless sensor network monitoring solution for greenhouses*," 2015 IEEE International Conference on Industrial Technology (ICIT), Seville, 2015, pp. 2192-2197.

- **T. Gomes**, D. Fernandes, M. Ekpanyapong and J. Cabral, "*An IoT-based system for collision detection on guardrails*," 2016 IEEE International Conference on Industrial Technology (ICIT), Taipei, 2016, pp. 1926-1931.
- **T. Gomes**, S. Pinto, T. Gomes, A. Tavares and J. Cabral, "*Towards an FPGA-based edge device for the Internet of Things*," 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 2015, pp. 1-4.
- S. Pinto; J. Pereira; **T. Gomes**; M. Ekpanyapong; A. Tavares, "*Towards a TrustZone-assisted Hypervisor for Real Time Embedded Systems*," in **IEEE Computer Architecture Letters** , vol.PP, no.99, pp.1-1
- **T. Gomes**, F. Salgado, S. Pinto, J. Cabral and A. Tavares, "*Towards an FPGA-based network layer filter for the Internet of Things edge devices*," 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, 2016, pp. 1-4.
- S. Pinto, **T. Gomes**, J. Pereira, J. Cabral and A. Tavares, "*IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices*," in **IEEE Internet Computing**, vol. 21, no. 1, pp. 40-47, Jan.-Feb. 2017.
- S. Pinto, J. Cabral and **T. Gomes**, *"We-care: An IoT-based health care system for elderly people,"* 2017 IEEE International Conference on Industrial Technology (ICIT), Toronto, ON, 2017, pp. 1378-1383.
- D. Fernandes, **T. Gomes**, et al., "*On-body signal propagation in WBANs for firefighters personal protective equipment: Statistical characterization and performance assessment,*" 2017 IEEE International Conference on Industrial Technology (ICIT), Toronto, ON, 2017, pp. 1360-1365.
- **T. Gomes**, S. Pinto, F. Salgado, A. Tavares and J. Cabral, "*Building IEEE 802.15.4 Accelerators for Heterogeneous Wireless Sensor Nodes*," in **IEEE Sensors Letters**, vol. 1, no. 1, pp. 1-4, Feb. 2017.
- F. Salgado; **T. Gomes**; S. Pinto; J. Cabral; A. Tavares, "*Condition Codes Evaluation on Dynamic Binary Translation for Embedded Platforms,*" in **IEEE Embedded Systems Letters**,vol. 9, no. 3, pp. 89-92, Sept. 2017.

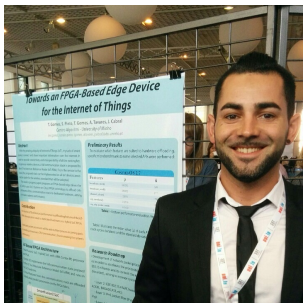
- **T. Gomes**, P. Lopes, J. Alves, P. Mestre, J. Cabral, J. L. Monteiro, and Adriano Tavares, "*A Modeling Domain-Specific Language for IoT-enabled Operating Systems,*" IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, 2017.

- F. Salgado, A. Martins, D. Almeida, **T. Gomes**, J. L. Monteiro, and Adriano Tavares, "*MODELA DBT: Model-Driven Elaboration Language Applied to Dynamic Binary Translation,*" IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, 2017.

- **T. Gomes**, F. Salgado, A. Tavares and J. Cabral, "*CUTE Mote, A CUstomizable and Trustable End-device for the Internet of Things,*" in **IEEE Sensors Journal**, vol.PP, no.99, pp.1-1.

- **T. Gomes**, F. Salgado, S. Pinto, J. Cabral and A. Tavares, "*A Hardware 6LoWPAN Packet Filter for IoT Edge Devices,*" in **IEEE Internet of Things Journal** (Under Review).

About the Author



Tiago Gomes is a researcher at the Embedded Systems Research Group at Centro ALGORITMI, University of Minho, Portugal. He holds an MSc in Telecommunications engineering from the same University. His research interests include embedded systems, hardware/software co-design for resource constrained wireless devices, wireless protocols for low-rate wireless personal area networks and network protocols for the Internet of Things low-end devices. For the last 5 years, as part of his academic pursuit, Gomes has visited several Universities, e.g., Aalto University (Finland), Würzburg University (Germany), Jilin University (China) and Asian Institute of Technology (Thailand), under PhD Exchange and ERASMUS Programmes. Gomes has more than 20 publications, both including articles on international journals and conference papers.

Bibliography

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002.

[2] V. C. Gungor and G. P. Hancke, “Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches,” *IEEE Transactions on Industrial Electronics*, vol. 56, pp. 4258–4265, October 2009.

[3] 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, pp. 88–97, ACM, 2002.

[4] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng, “Review: From wireless sensor networks towards cyber physical systems,” *Pervasive and Mobile Computing*, vol. 7, pp. 397–413, August 2011.

[5] L. Mainetti, L. Patrono, and A. Vilei, “Evolution of wireless sensor networks towards the Internet of Things: A survey,” in *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, pp. 1–6, September 2011.

[6] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A Survey,” *Computer Networks*, vol. 54, pp. 2787–2805, October 2010.

[7] National Intelligence Council (NIC), “Disruptive Civil Technologies - Six Technologies with Potential Impacts on US Interests Out to 2025 - Conference Report CR 2008-07,” tech. rep., National Intelligence Council (NIC), July 2008.

[8] Gartner, Inc., “Gartner’s 2015 Hype Cycle for Emerging Technologies,” tech. rep., Gartner, Inc., August 2015.

[9] L. Tan and N. Wang, "Future internet: The Internet of Things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, pp. V5–376–V5–380, August 2010.

[10] D. Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything," tech. rep., Cisco Systems, Inc, 2011.

[11] Intel Corporation, "A Guide to the Internet of Things." [Online]. Available: `https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html`, Date accessed June, 8 2017.

[12] J. Bradley, J. Barbier, and D. Handler, "Embracing the Internet of Everything To Capture Your Share of $14.4 Trillion," tech. rep., Cisco Systems, Inc, 2013.

[13] J. Macaulay, L. Buckalew, and G. Chung, "Internet of Things in Logistics," tech. rep., DHL Trend Research & Cisco Consulting Services, 2015.

[14] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, "The Internet of Things: Mapping the Value Beyond the Hype," tech. rep., McKinsey Global Institute, 2015.

[15] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, pp. 22–32, February 2014.

[16] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An Information Framework for Creating a Smart City Through Internet of Things," *IEEE Internet of Things Journal*, vol. 1, pp. 112–121, April 2014.

[17] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT Considerations, Requirements, and Architectures for Smart Buildings - Energy Optimization and Next-Generation Building Management Systems," *IEEE Internet of Things Journal*, vol. 4, pp. 269–283, February 2017.

[18] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the Implementation of IoT for Environmental Condition Monitoring in Homes," *IEEE Sensors Journal*, vol. 13, pp. 3846–3853, October 2013.

[19] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 2233–2243, November 2014.

[20] L. Catarinucci, D. de Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone, "An IoT-Aware Architecture for Smart Healthcare Systems," *IEEE Internet of Things Journal*, vol. 2, pp. 515–526, December 2015.

[21] M. A. Al-Taee, W. Al-Nuaimy, Z. J. Muhsin, and A. Al-Ataby, "Robot Assistant in Management of Diabetes in Children Based on the Internet of Things," *IEEE Internet of Things Journal*, vol. 4, pp. 437–445, April 2017.

[22] T. C. Chiu, Y. Y. Shih, A. C. Pang, and C. W. Pai, "Optimized Day-Ahead Pricing With Renewable Energy Demand-Side Management for Smart Grids," *IEEE Internet of Things Journal*, vol. 4, pp. 374–383, April 2017.

[23] X. Huang and N. Ansari, "Content Caching and Distribution in Smart Grid Enabled Wireless Networks," *IEEE Internet of Things Journal*, vol. 4, pp. 513–520, April 2017.

[24] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark, "Connected Vehicles: Solutions and Challenges," *IEEE Internet of Things Journal*, vol. 1, pp. 289–299, Aug 2014.

[25] Micrium, "IoT for Embedded Systems: The New Industrial Revolution." [Online]. Available: `https://www.micrium.com/iot/devices/`, Date accessed June, 8 2017.

[26] A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, and S. Meissner, *Enabling Things to Talk*. Springer New York, 2013.

[27] Intel Corporation, "The Intel IoT Platform - Architecture Specification." White Paper, April 2015.

[28] oneM2M, "one M2M: The interoperability enabler for the entire M2M and IoT ecosystem." White Paper, January 2015.

[29] Cisco Systems, Inc, "The Internet of Things Reference Model." White Paper, January 2014.

[30] Oracle, "Wireless Communication Standards for the Internet of Things." White Paper in Enterprise Architecture, October 2009.

[31] Paul Fremantle, "A Reference Architecture for the Internet of Things." White Paper, Version 0.9.0, October 2015.

[32] Martin Bauer et all, "Deliverable D1.5 - Final architectural reference model for the IoT v3.0," tech. rep., Internet of Things - Architecture, January 2015.

[33] Cees Links, GreenPeak Technologies CEO, "Wireless Communication Standards for the Internet of Things." White Paper, January 2015.

[34] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the Internet of things," *IEEE Internet Computing*, vol. 14, pp. 44–51, January 2010.

[35] Silicon Laboratories Inc., "Overcoming Challenges of Connecting Intelligent Nodes to the Internet of Things." White Paper, January 2012.

[36] Andre Foster, PrismTech, "Messaging Technologies for the Industrial Internet and the Internet of Things." White Paper, January 2015.

[37] Cisco Systems, Inc, "Integrating an Industrial Wireless Sensor Network with Your Plant's Switched Ethernet and IP Network." White Paper, January 2009.

[38] Texas Instruments, "Understanding Wireless Connectivity in the Industrial IoT." White Paper, January 2015.

[39] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys Tutorials*, vol. 15, pp. 1389–1406, March 2013.

[40] GSM Association, "IoT Security Guidelines: Overview Document." White Paper, Version 1.1, November 2016.

[41] GSM Association, "IoT Security Guidelines for Endpoint Ecosystems." White Paper, Version 1.1, November 2016.

[42] Symantec, "An Internet of Things Reference Architecture." White Paper, 2016.

[43] M. Vai, D. J. Whelihan, B. R. Nahill, D. M. Utin, S. R. O'Melia, and R. I. Khazan, "Secure Embedded Systems," *LINCOLN LABORATORY JOURNAL*, vol. 22, no. 1, pp. 110–122, 2016.

[44] Jorge Granjal and Edmundo Monteiro and Jorge Sá Silva, "Security in the integration of low-power Wireless Sensor Networks with the Internet: A survey," *Ad Hoc Networks*, vol. 24, no. Part A, pp. 264 – 287, 2015.

[45] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE Communications Surveys Tutorials*, vol. 17, pp. 1294–1312, thirdquarter 2015.

[46] Intel Corporation, "New Microarchitecture for 4th GenIntel Core Processor Platforms." Product Brief, 2013.

[47] Synopsys, Inc., "DesignWare tRoot Secure Hardware Root of Trust." Datasheet, 2015.

[48] ARM, "ARM Security Technology: Building a Secure System using TrustZone Technology." White Paper, 2009.

[49] S. Pinto, T. Gomes, J. Pereira, J. Cabral, and A. Tavares, "IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices," *IEEE Internet Computing*, vol. 21, pp. 40–47, January 2017.

[50] Jonas Olsson, Texas Instruments, "6LoWPAN Demystified." White Paper, October 2014.

[51] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, RFC Editor, September 2007. `http://www.rfc-editor.org/rfc/rfc4944.txt`.

[52] A. Aijaz and A. H. Aghvami, "Cognitive Machine-to-Machine Communications for Internet-of-Things: A Protocol Stack Perspective," *IEEE Internet of Things Journal*, vol. 2, pp. 103–112, April 2015.

[53] Gregory Guez, Maxim Integrated, "Why Hardware-Based Design Security is Essential for Every Application." White Paper, 2017.

[54] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, pp. 54–62, October 2002.

[55] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE Internet of Things Journal*, vol. 1, pp. 265–275, June 2014.

[56] C. Hennebert and J. D. Santos, "Security Protocols and Privacy Issues into 6LoWPAN Stack: A Synthesis," *IEEE Internet of Things Journal*, vol. 1, pp. 384–398, Oct 2014.

[57] N. Maruyama, T. Ishihara, and H. Yasuura, "An RTOS in hardware for energy efficient software-based TCP/IP processing," in *Application Specific Processors (SASP), 2010 IEEE 8th Symposium on*, pp. 58–63, June 2010.

[58] F. Hijaz, B. Kahne, P. Wilson, and O. Khan, "Efficient parallel packet processing using a shared memory many-core processor with hardware support to accelerate communication," in *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pp. 122–129, August 2015.

[59] L. Bettini, *Implementing Domain-Specific Languages with Xtext and Xtend.* Packt Publishing, 2013.

[60] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-specific Languages," *ACM Comput. Surv.*, vol. 37, pp. 316–344, December 2005.

[61] M. Fowler, *Domain Specific Languages.* Addison-Wesley Professional, 1st ed., 2010.

[62] M. Payer, B. Bluntschli, and T. R. Gross, "LLDSAL: A Low-level Domain-specific Aspect Language for Dynamic Code-generation and Program Modification," in *Proceedings of the Seventh Workshop on Domain-Specific Aspect Languages*, DSAL '12, pp. 15–20, ACM, 2012.

[63] N. Kapre and S. Bayliss, "Survey of domain-specific languages for FPGA computing," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–12, August 2016.

[64] A. Garnier, J. M. Menaud, and R. Pottier, "SensorScript: A Business-Oriented Domain-Specific Language for Sensor Networks," in *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 44–49, August 2015.

[65] A. Salihbegovic, T. Eterovic, E. Kaljic, and S. Ribic, "Design of a domain specific language and IDE for Internet of things applications," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 996–1001, May 2015.

[66] Zolertia, "RE-MOTE." [Online]. Available: `http://zolertia.io/product/re-mote/`, Date accessed June, 8 2017.

[67] Arago Systems, "WiSMote." [Online]. Available: `http://www.aragosystems.com/produits/wisnet/wismote/`, Date accessed June, 8 2017.

[68] Eistec, “Mulle.” [Online]. Available: http://www.eistec.se/mulle/, Date accessed June, 8 2017.

[69] Advantic Systems, “Tmote Sky.” [Online]. Available: https://telosbsensors.wordpress.com/, Date accessed June, 8 2017.

[70] OpenMote Technologies, “OpenMote.” [Online]. Available: http://www.openmote.com/, Date accessed June, 8 2017.

[71] SECO SRL, “UDOO.” [Online]. Available: https://www.udoo.org/, Date accessed June, 8 2017.

[72] Micro:bit Educational Foundation, “micro:bit.” [Online]. Available: http://microbit.org/, Date accessed June, 8 2017.

[73] P. Samundiswary, P. Priyadarshini, and P. Dananjayan, “Performance Evaluation of Heterogeneous Sensor Networks,” in *2009 International Conference on Future Computer and Communication*, pp. 264–267, April 2009.

[74] J. Valverde, A. Otero, M. Lopez, J. Portilla, E. de la Torre, and T. Riesgo, “Using sram based fpgas for power-aware high performance wireless sensor networks,” *Sensors*, vol. 12, no. 3, pp. 2667–2692, 2012.

[75] P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, “Review of Hardware Architectures for Advanced Encryption Standard Implementations Considering Wireless Sensor Networks,” in *Proceedings of the 7th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS’07, pp. 443–453, 2007.

[76] S. Peter, O. Stecklina, J. Portilla, E. de la Torre, P. Langendoerfer, and T. Riesgo, “Reconfiguring Crypto Hardware Accelerators on Wireless Sensor Nodes,” in *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, pp. 1–3, June 2009.

[77] T. F. Al-Somani and H. Houssain, “Implementation of GF(2m) Elliptic Curve cryptoprocessor on a Nano FPGA,” in *2011 International Conference for Internet Technology and Secured Transactions*, pp. 7–12, December 2011.

[78] E. Eryümaz, I. Erturk, and S. Atmaca, "Implementation of Skipjack cryptology algorithm for WSNs using FPGA," in *2009 International Conference on Application of Information and Communication Technologies*, pp. 1–5, October 2009.

[79] H. Houssain, M. Badra, and T. F. Al-Somani, "Hardware implementations of Elliptic Curve Cryptography in Wireless Sensor Networks," in *2011 International Conference for Internet Technology and Secured Transactions*, pp. 1–6, Dec 2011.

[80] X. Zhang, H. M. Heys, and C. Li, "FPGA implementation of two involutional block ciphers targeted to wireless sensor networks," in *2011 6th International ICST Conference on Communications and Networking in China (CHINACOM)*, pp. 232–236, August 2011.

[81] Y. Wang, S. Lu, and L. Cui, "Design and implementation of a SoC-based security coprocessor and program protection mechanism for WSN," in *IET International Conference on Wireless Sensor Network 2010 (IET-WSN 2010)*, pp. 148–153, November 2010.

[82] J. g. Tong, Z. x. Zhang, Q. l. Sun, and Z. q. Chen, "Design of Wireless Sensor Network Node with Hyperchaos Encryption Based on FPGA," in *2009 International Workshop on Chaos-Fractals Theories and Applications*, pp. 190–194, November 2009.

[83] N. Kimura and S. Latifi, "A survey on data compression in wireless sensor networks," in *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, vol. 2, pp. 8–13 Vol. 2, April 2005.

[84] W. Bajwa, J. Haupt, A. Sayeed, and R. Nowak, "Compressive wireless sensing," in *2006 5th International Conference on Information Processing in Sensor Networks*, pp. 134–142, April 2006.

[85] Y. Wang, A. Bermak, and F. Boussaid, "FPGA implementation of compressive sampling for sensor network applications," in *2nd Asia Symposium on Quality Electronic Design (ASQED)*, pp. 5–8, August 2010.

[86] C. J. Debono and N. P. Borg, "The Implementation of an Adaptive Data Reduction Technique for Wireless Sensor Networks," in *2008 IEEE International Symposium on Signal Processing and Information Technology*, pp. 402–406, December 2008.

[87] T. T.-O. Kwok and Y.-K. Kwok, "Computation and energy efficient image processing in wireless sensor networks based on reconfigurable computing," in *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, pp. 8 pp.–50, 2006.

[88] Y. Sun, L. Li, and H. Luo, "Design of FPGA-Based Multimedia Node for WSN," in *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–5, September 2011.

[89] J. Thoné, J. Verlinden, and R. Puers, "An efficient hardware-optimized compression algorithm for wireless capsule endoscopy image transmission," *Procedia Engineering*, vol. 5, pp. 208 – 211, 2010. Eurosensor XXIV Conference.

[90] A. Chefi, A. Soudani, and G. Sicard, "Hardware compression solution based on HWT for low power image transmission in WSN," in *ICM 2011 Proceeding*, pp. 1–5, December 2011.

[91] M. L. Kaddachi, A. Soudani, I. Nouira, V. Lecuire, and K. Torki, "Efficient hardware solution for low power and adaptive image-compression in WSN," in *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, pp. 583–586, December 2010.

[92] Y. E. Krasteva, J. Portilla, E. de la Torre, and T. Riesgo, "Embedded Runtime Reconfigurable Nodes for Wireless Sensor Networks Applications," *IEEE Sensors Journal*, vol. 11, pp. 1800–1810, September 2011.

[93] P. Latha and M. A. Bhagyaveni, "Reconfigurable FPGA based architecture for surveillance systems in WSN," in *2010 International Conference on Wireless Communication and Sensor Computing (ICWCSC)*, pp. 1–6, January 2010.

[94] P. Muralidhar and C. B. R. Rao, "Reconfigurable wireless sensor network node based on Nios core," in *2008 Fourth International Conference on Wireless Communication and Sensor Networks*, pp. 67–72, December 2008.

[95] S. Tanaka, N. Fujita, Y. Yanagisawa, T. Terada, and M. Tsukamoto, "Reconfigurable hardware architecture for saving power consumption on a sensor node," in *2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 405–410, December 2008.

[96] H. Hinkelmann, P. Zipf, and M. Glesner, “A Domain-Specific Dynamically Reconfigurable Hardware Platform for Wireless Sensor Networks,” in *2007 International Conference on Field-Programmable Technology*, pp. 313–316, December 2007.

[97] G. Chalivendra, R. Srinivasan, and N. S. Murthy, “FPGA based reconfigurable wireless sensor network protocol,” in *2008 International Conference on Electronic Design*, pp. 1–4, December 2008.

[98] G. G. Mplemenos, P. Christou, and I. Papaefstathiou, “Using reconfigurable hardware devices in WSNs for accelerating and reducing the power consumption of header processing tasks,” in *2009 IEEE 3rd International Symposium on Advanced Networks and Telecommunication Systems (ANTS)*, pp. 1–3, December 2009.

[99] Y. E. Krasteva, J. Portilla, J. M. Carnicer, E. de la Torre, and T. Riesgo, “Remote HW-SW reconfigurable Wireless Sensor nodes,” in *2008 34th Annual Conference of IEEE Industrial Electronics*, pp. 2483–2488, November 2008.

[100] V. Jeličić, T. Ražov, D. Oletić, M. Kuri, and V. Bilas, “Maslinet: A wireless sensor network based environmental monitoring system,” in *2011 Proceedings of the 34th International Convention MIPRO*, pp. 150–155, May 2011.

[101] J. Koskinen, P. Kilpeläinen, J. Rehu, P. Tukeva, and M. Sallinen, “Wireless Sensor Networks for infrastructure and industrial monitoring applications,” in *2010 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 250–255, November 2010.

[102] P. Turcza and M. Duplaga, “Low power FPGA-based image processing core for wireless capsule endoscopy,” *Sensors and Actuators A: Physical*, vol. 172, no. 2, pp. 552 – 560, 2011.

[103] A. de la Piedra, A. Braeken, and A. Touhafi, “Sensor Systems Based on FPGAs and Their Applications: A Survey,” *Sensors*, vol. 12, no. 9, p. 12235, 2012.

[104] A. Engel and A. Koch, “Heterogeneous Wireless Sensor Nodes that Target the Internet of Things,” *IEEE Micro*, vol. 36, pp. 8–15, November 2016.

[105] V. Rosello, J. Portilla, and T. Riesgo, “Ultra low power FPGA-based architecture for Wake-up Radio in Wireless Sensor Networks,” in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, pp. 3826–3831, November 2011.

[106] O. Berder and O. Sentieys, “PowWow : Power Optimized Hardware/Software Framework for Wireless Motes,” in *23th International Conference on Architecture of Computing Systems 2010*, pp. 1–5, February 2010.

[107] L. A. Vera-Salas, S. V. Moreno-Tapia, R. A. Osornio-Rios, and R. d. J., “Reconfigurable Node Processing Unit for a Low-Power Wireless Sensor Network,” in *2010 International Conference on Reconfigurable Computing and FPGAs*, pp. 173–178, December 2010.

[108] T. Nyländen, J. Boutellier, K. Nikunen, J. Hannuksela, and O. Silvén, “Reconfigurable miniature sensor nodes for condition monitoring,” in *2012 International Conference on Embedded Computer Systems (SAMOS)*, pp. 113–119, July 2012.

[109] B. Stelte, “Toward Development of High Secure Sensor Network Nodes Using an FPGA-based Architecture,” in *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC ’10, pp. 539–543, ACM, 2010.

[110] A. Engel and A. Koch, *Hardware-Accelerated Data Compression in Low-Power Wireless Sensor Networks*, pp. 167–178. Cham: Springer International Publishing, 2014.

[111] A. Engel, A. Koch, and T. Siebel, “A heterogeneous system architecture for low-power wireless sensor nodes in compute-intensive distributed applications,” in *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*, pp. 636–644, October 2015.

[112] J. Portilla, A. Otero, E. de la Torre, T. Riesgo, O. Stecklina, S. Peter, and P. Langendörfer, “Adaptable Security in Wireless Sensor Networks by Using Reconfigurable ECC Hardware Coprocessors,” *International Journal of Distributed Sensor Networks*, vol. 6, no. 1, p. 740823, 2010.

[113] J. Valverde, V. Rosello, G. Mujica, J. Portilla, A. Uriarte, and T. Riesgo, “Wireless Sensor Network for Environmental Monitoring: Application in a Coffee Factory,” *International Journal of Distributed Sensor Networks*, vol. 8, no. 1, p. 638067, 2012.

[114] J. Valverde, A. Otero, M. Lopez, J. Portilla, E. de la Torre, and T. Riesgo, “Using SRAM Based FPGAs for Power-Aware High Performance Wireless Sensor Networks,” *Sensors*, vol. 12, no. 3, pp. 2667–2692, 2012.

[115] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, “The first collision for full SHA-1,” tech. rep., CWI Amsterdam & Google Research, 2017.

[116] J. d. J. Rangel-Magdaleno, R. d. J. Romero-Troncoso, R. A. Osornio-Rios, E. Cabal-Yepez, and A. Dominguez-Gonzalez, “FPGA-Based Vibration Analyzer for Continuous CNC Machinery Monitoring With Fused FFT-DWT Signal Processing,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, pp. 3184–3194, December 2010.

[117] A. Ordaz-Moreno, R. J. Romero-Troncoso, and J. A. Vite-Frias, “Hardware signal processing unit for one-dimensional variable-length discrete wavelet transform,” in *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig’05)*, pp. 5 pp.–5, September 2005.

[118] S. Ishihara, Z. Xia, M. Hariyama, and M. Kameyama, “Architecture of a low-power FPGA based on self-adaptive voltage control,” in *2009 International SoC Design Conference (ISOCC)*, pp. 274–277, November 2009.

[119] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton, “Dynamic voltage scaling for commercial FPGAs,” in *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, pp. 173–180, December 2005.

[120] Levis, P. and Madden, S. and Polastre, J. and Szewczyk, R. and Whitehouse, K. and Woo, A. and Gay, D. and Hill, J. and Welsh, M. and Brewer, E. and Culler, D., *TinyOS: An Operating System for Sensor Networks*, pp. 115–148. Springer Berlin Heidelberg, 2005.

[121] A. Eswaran, A. Rowe, and R. Rajkumar, “Nano-rk: An energy-aware resource-centric rtos for sensor networks,” in *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, RTSS ’05, pp. 256–265, IEEE Computer Society, 2005.

[122] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet.* John Wiley & Sons, Ltd, 2009.

[123] S. Kent and K. Seo, “Security Architecture for the Internet Protocol,” RFC 4301, RFC Editor, December 2005. `http://www.rfc-editor.org/rfc/rfc4301.txt`.

[124] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, “Operating Systems for Low-End Devices in the Internet of Things: A Survey,” *IEEE Internet of Things Journal*, vol. 3, pp. 720–734, October 2016.

[125] Contiki-OS, “Contiki: The Open Source OS for the Internet of Things.” [Online]. Available: `http://www.contiki-os.org/`, Date accessed January, 24 2017.

[126] RIOT-OS, “RIOT: The friendly Operating System for the Internet of Things.” [Online]. Available: `http://www.riot-os.org/`, Date accessed January, 24 2017.

[127] freeRTOS, “FreeRTOS, a free open source RTOS for small embedded real time systems.” [Online]. Available: `http://www.freertos.org/`, Date accessed January, 24 2017.

[128] OpenWSN, “Berkeley’s OpenWSN project.” [Online]. Available: `http://openwsn.berkeley.edu/`, Date accessed January, 24 2017.

[129] NuttX, “NuttX Real-Time Operating System.” [Online]. Available: `http://nuttx.org/`, Date accessed January, 24 2017.

[130] eCos, “The Embedded Configurable Operating System.” [Online]. Available: `http://ecos.sourceware.org/`, Date accessed January, 24 2017.

[131] uClinux, “Embedded Linux/microcontoller project.” [Online]. Available: `http://www.uclinux.org`, Date accessed January, 24 2017.

[132] ChibiOS/RT, “ChibiOS/RT.” [Online]. Available: `http://www.chibios.org`, Date accessed January, 24 2017.

[133] CooCox, "CoOS." [Online]. Available: `http://www.coocox.org/`, Date accessed January, 24 2017.

[134] Nut/OS, "Nut/OS." [Online]. Available: `http://www.ethernut.de/en/software/`, Date accessed January, 24 2017.

[135] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pp. 455–462, November 2004.

[136] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pp. 29–42, ACM, 2006.

[137] Dunkels, Adam, "The ContikiMAC Radio Duty Cycling Protocol," tech. rep., SICS, Sweden, 2011.

[138] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, (New York, NY, USA), pp. 307–320, ACM, 2006.

[139] Contiki-OS, "Contiki Hardware." [Online]. Available: `http://www.contiki-os.org/hardware.html`, Date accessed January, 24 2017.

[140] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "Riot os: Towards an os for the internet of things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 79–80, April 2013.

[141] E. Baccelli, O. Hahm, H. Petersen, and K. Schleiser, "RIOT and the Evolution of IoT Operating Systems and Applications," *ERCIM News*, April 2015.

[142] H. Will, K. Schleiser, and J. Schiller, "A real-time kernel for wireless sensor networks employed in rescue scenarios," in *2009 IEEE 34th Conference on Local Computer Networks*, pp. 834–841, October 2009.

[143] RIOT-OS, "RIOT Hardware." [Online]. Available: `https://github.com/RIOT-OS/RIOT/wiki/RIOT-Platforms`, Date accessed January, 24 2017.

[144] Texas Instruments, Inc., "Wireless Connectivity Portfolio." [Online]. Available:`http://www.ti.com/lsds/ti/wireless-connectivity/simplelink-solutions/overview.page`, Date accessed January, 24 2017.

[145] Microsemi Corporation, "PolarFire FPGA Family." [Online]. Available: `https://www.microsemi.com/products/fpga-soc/fpga/polarfire-fpga`, Date accessed June, 24 2017.

[146] Microsemi Corporation, "IGLOO2 FPGA Family." [Online]. Available: `https://www.microsemi.com/products/fpga-soc/fpga/igloo2-fpga`, Date accessed June, 24 2017.

[147] Xilinx Inc., "Zynq-7000 All Programmable SoC." [Online]. Available: `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`, Date accessed June, 24 2017.

[148] Intel Corporation, "Cyclone V SoCs." [Online]. Available: `https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html`, Date accessed June, 24 2017.

[149] Lattice Semiconducto, "LatticeXP2." [Online]. Available: `http://www.latticesemi.com/Products/FPGAandCPLD/LatticeXP2.aspx`, Date accessed June, 24 2017.

[150] Cypress Semiconductor Corporation, "PSoC 5LP family." [Online]. Available: `http://www.cypress.com/products/psoc-5`, Date accessed June, 24 2017.

[151] Microsemi Corporation, "SmartFusion2 Security Evaluation Kit." [Online]. Available: `http://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/smartfusion2/sf2-evaluation-kit`, Date accessed January, 24 2017.

[152] W. Lamie and J. Carbone, "Measure your RTOS's real-time performance," *Embedded Systems Design*, vol. 20, no. 5, p. 44, 2007.

[153] Mentor Graphics, "ModelSim ME." [Online]. Available: `https://www.mentor.com/products/fv/modelsim/`, Date accessed January, 24 2017.

[154] N. A. Pantazis and D. D. Vergados, "A survey on power control issues in wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 9, pp. 86–107, Fourth 2007.

[155] Dunkels, Adam, “uIP - A Free Small TCP/IP Stack,” January 2012.

[156] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, “OpenWSN: a standards-based low-power wireless development environment,” *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[157] J. Granjal, R. Silva, E. Monteiro, J. S. Silva, and F. Boavida, “Why is IPSec a viable option for wireless sensor networks,” in *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 802–807, Sept 2008.

[158] S. Raza, T. Chung, S. Duquennoy, D. Yazar, and U. Roedig, “Securing Internet of Things with Lightweight IPsec,” Tech. Rep. 3, SICS, 2010.

[159] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, “Securing communication in 6LoWPAN with compressed IPsec,” in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pp. 1–8, June 2011.

[160] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis, “Koala: Ultra-low power data retrieval in wireless sensor networks,” in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, IPSN ’08, (Washington, DC, USA), pp. 421–432, IEEE Computer Society, 2008.

[161] 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*, SenSys ’04, (New York, NY, USA), pp. 95–107, ACM, 2004.

[162] IEEE, “IEEE 802.15 Working Group for Wireless Specialty Networks (WSN),” 2003. `http://www.ieee802.org/15/`.

[163] ZigBee Alliance, “ZigBee Document 053474r06 Version 1.0,” standard specification, ZigBee Alliance, 2004.

[164] ZigBee Alliance, “ZigBee-2006 specification, ZigBee Document 064112,” standard specification, ZigBee Alliance, 2006.

[165] "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE Std 802.15.4-2003*, pp. 0_1–670, 2003.

[166] A. Rajandekar and B. Sikdar, "A Survey of MAC Layer Issues and Protocols for Machine-to-Machine Communications," *IEEE Internet of Things Journal*, vol. 2, pp. 175–186, April 2015.

[167] "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, April 2016.

[168] Linear Technology, "Dust Networks," 2007. [Online]. Available: `http://www.linear.com/dust_networks/`, Date accessed June, 8 2017.

[169] K. S. J. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," in *In Proceedings of the IASTED International Symposium on Distributed Sensor Networks (DSN08)*, 2008.

[170] T. Watteyne, M. Palattella, and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement," RFC 7554, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7554.txt`.

[171] Texas Instruments, Inc., "CC2520 - 2.4 GHZ IEEE 802.15.4/ZIGBEE RF TRANSCEIVER." [Online]. Available: `http://www.ti.com/product/CC2520`, Date accessed June, 13 2016.

[172] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 1–320, September 2006.

[173] Atmel Corporation, "Low Power 2.4 GHz Transceiver for IEEE 802.15.4, ZigBee, 6LoWPAN, RF4CE, SP100, WirelessHART and ISM Applications.."

[174] Silicon Laboratories, "EM35x System-on-Chip (SoC) / Network Co-Processor (NCP) for ZigBee." [Online]. Available: `https://www.silabs.com/products/wireless/mesh-networking/em3xx/Pages/em3xx-zigbee.aspx`, Date accessed June, 13 2016.

[175] Texas Instruments, Inc., "CC2538 - System-On-Chip for 2.4-GHz IEEE 802.15.4-2006 6LoWPAN and ZigBee Applications." [Online]. Available: `http://www.ti.com/product/CC2538`, Date accessed January, 18 2017.

[176] O. Song and J. Kim, "An Efficient Design of Security Accelerator for IEEE 802.15.4 Wireless Sensor Networks," in *2010 7th IEEE Consumer Communications and Networking Conference*, pp. 1–5, January 2010.

[177] P. Hamalainen, M. Hannikainen, and T. D. Hamalainen, "Efficient hardware implementation of security processing for IEEE 802.15.4 wireless networks," in *48th Midwest Symposium on Circuits and Systems, 2005.*, pp. 484–487 Vol. 1, August 2005.

[178] A. D. Stefano, G. Fiscelli, and C. G. Giaconia, "An FPGA-Based Software Defined Radio Platform for the 2.4GHz ISM Band," in *2006 Ph.D. Research in Microelectronics and Electronics*, pp. 73–76, 2006.

[179] A. Massouri and T. Risset, "FPGA-based Implementation of Multiple PHY Layers of IEEE 802.15.4 Targeting SDR Platform," in *SDR-WInnComm*, (Schaumburg, Illinois, United States), Wireless Innovation Forum, April 2014.

[180] N. Kimura and S. Latifi, "A survey on data compression in wireless sensor networks," in *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, vol. 2, pp. 8–13 Vol. 2, April 2005.

[181] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, RFC Editor, September 2011. `http://www.rfc-editor.org/rfc/rfc6282.txt`.

[182] I. Hosni and N. Hamdi, "Neighbor Discovery Schedule for 6LoWPAN Smart Grid Applications," in *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 388–393, March 2016.

[183] M. Ha, S. H. Kim, and D. Kim, "Intra-MARIO: A Fast Mobility Management Protocol for 6LoWPAN," *IEEE Transactions on Mobile Computing*, vol. 16, pp. 172–184, January 2017.

[184] H. Shah, R. Shrimali, and V. Parikh, "Header Compression and Neighbor Discovery in 6LoWPAN based IoT - a survey," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 306–311, March 2016.

[185] L. Cui, G. Hua, and N. Lu, "A Dynamic 6LoWPAN Context Table Maintaining algorithm," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 1458–1463, July 2013.

[186] N. Li and X. Huang, "A context system for 6LoWPAN network," in *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology*, pp. 522–525, October 2011.

[187] M. A. M. Seliem, K. M. F. Elsayed, and A. Khattab, "Performance evaluation and optimization of neighbor discovery implementation over Contiki OS," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pp. 119–123, March 2014.

[188] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, "Denial-of-Service detection in 6LoWPAN based Internet of Things," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 600–607, October 2013.

[189] Czarnecki, Krzysztof, "Overview of Generative Software Development," in *Proceedings of the 2004 International Conference on Unconventional Programming Paradigms*, UPP'04, pp. 326–341, 2005.

[190] Uwe Zdun, "Concepts for Model-Driven Design and Evolution of Domain-Specific Languages," 2005.

[191] J.-P. Tolvanen and M. Rossi, "MetaEdit+: Defining and Using Domain-specific Modeling Languages and Code Generators," in *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, pp. 92–93, ACM, 2003.

[192] J. Marino and M. Rowley, *Understanding SCA (Service Component Architecture).* Addison-Wesley Professional, 1st ed., 2009.