

2017 • 2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Towards Software-Defined Radio on Configurable Hardware

PROMOTOR :

Prof. dr. ir. Nele MENTENS

PROMOTOR :

dr. ir. Wim AERTS

BEGELEIDER :

De heer Jori WINDERICKX

Karel Bertrands

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.



Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt



2017 • 2018

Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Towards Software-Defined Radio on Configurable Hardware

PROMOTOR :

Prof. dr. ir. Nele MENTENS

PROMOTOR :

dr. ir. Wim AERTS

BEGELEIDER :

De heer Jori WINDERICKX

Karel Bertrands

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT



KU LEUVEN

Acknowledgements

When I reached my Bachelors degree and had to choose a subject for my Masters thesis last year, I for sure knew that I wanted to do something which was entirely new to me. This way I would be able to expand my knowledge as an industrial engineer electronics-ICT. That is why I searched for something which did not have any strong connections with the elective courses I followed this year. When reading through all the available projects, only few really sparked my interest. It took me a lot of effort to get familiarized with the provided hardware and all the used protocols. But in the end, despite the thesis not being what I expected from it, I still learned a lot of new things which will certainly help me further on in my career.

I would like to thank some people for supporting me during my thesis and helping me reach my goals. First, I would like to thank prof. dr. ing. Nele Mentens for giving me the opportunity to work on my thesis at the Embedded Systems and Security research group. Without her experience and knowledge I would not have been able to accomplish this project. Then, I would like to thank dr. ing. Wim Aerts for all the information and insight he has provided during this year. Without his experience, it would definitely have been a lot harder to understand the concepts used in this thesis. I also would like to thank ing. Jori Winderickx for helping me get worked in and providing valuable insights.

Finally I would like to thank my parents, family and friends for their support and interest in my research during my thesis. Special thanks go out to Michiel Darcis and Sander Denorme for their encouragements and friendship we had during the 4 years we studied together.

Contents

List of Tables	v
List of Figures	viii
Acronyms	ix
Abstract	xi
Abstract in Dutch	xiii
1 Introduction	1
1.1 Context and Problem Statement	1
1.2 Objectives	2
1.3 Method	3
2 Literature Study	5
2.1 Software Defined Radio	5
2.1.1 Receiver	5
2.1.2 Transmitter	6
2.1.3 Digital Signal Processing	6
2.1.4 Ideal Software Defined Radio	7
2.2 Internet of Things Protocols	8
2.2.1 ZigBee	8
2.2.2 6LoWPAN	9
3 Materials and Method	11
3.1 BladeRF x115	11
3.2 Message Repeater	12
3.2.1 Defaul Architecture	12
3.2.2 Repeater Architecture	13
3.3 ZigBee Transceiver Matlab/Simulink Simulation	14
3.3.1 Transmitter	14
3.3.2 Receiver	15
3.3.3 Simulink Implementation	16
3.4 Xilinx ISE Simulation	17
3.4.1 Bit to Symbol Conversion	17
3.4.2 Direct-Sequence Spread Spectrum (DSSS) Lookup Table	18
3.4.3 De-interlace	18
3.4.4 Serializer	19
3.4.5 Pulse Shaping	19
4 Results	21

4.1	Message Repeater	21
4.2	Simulink Simulation Results	23
4.3	HDL Simulation Results	25
4.3.1	Design Summary	25
4.3.2	Simulation	25
5	Conclusion and Future Work	29
5.1	Message Repeater and Simulations	29
5.2	Future Work	30
	Bibliography	31
	Appendices	33
	A: Diagrams of VHDL modules	35

List of Tables

Table 1	Internet of Things (IoT) nodes installed by category (Millions of Units) . . .	1
Table 2	DSSS mapping table	15
Table 3	End-to-end delays created by (de)modulation configurations	17
Table 4	Results of Matlab functions and LUT values	20
Table 5	Bit Error Rate (BER) results for Simulink configurations with DSSS and Maximum Likelihood Estimator (MLE)	23
Table 6	BER results for Simulink transceiver configurations without DSSS and MLE	23

List of Figures

Figure 1	Channels of IEEE 802.15.4 standard	2
Figure 2	Topology of basic software defined radio receiver	5
Figure 3	Topology of basic software defined radio transmitter	6
Figure 4	Topology of ideal software defined radio	7
Figure 5	Possible ZigBee network topologies	8
Figure 6	ZigBee protocol stack	9
Figure 7	6LoWPAN protocol stack	9
Figure 8	6LoWPAN mesh topology	10
Figure 9	BladeRF x115 device	11
Figure 10	Diagram of default FPGA image	12
Figure 11	Connection between <i>tx sample fifo</i> and <i>rx sample fifo</i>	13
Figure 12	Zoomed-in view of architecture message repeater	13
Figure 13	BladeRF x115 device settings for message repeater architecture	14
Figure 14	Simplified diagram of ZigBee transmitter	14
Figure 15	Simplified diagram of ZigBee receiver	15
Figure 16	Simulink system ZigBee transmitter	16
Figure 17	Simulink single rate ZigBee receiver system with integer values	16
Figure 18	Simplified Very High Speed Integrated Circuit Hardware Description Language (VHDL) modulator system	17
Figure 19	Results of testbench for DSSS module	18
Figure 20	Results of testbench for De-interlace module	18
Figure 21	Results of testbench for serializer module	19
Figure 22	Half sine pulses generated by the Offset-Quadrature Phase Shift Keying (OQPSK) modulator	20
Figure 23	6LoWPAN messages sent by the Zolertia Z1 node	21
Figure 24	Retransmitted I- and Q-samples when Zolertia is deactivated	22
Figure 25	Retransmitted I- and Q-samples when Zolertia is activated	22
Figure 26	Retransmitted I- and Q-samples when Zolertia is activated, but Bluetooth emitter interferes	22
Figure 27	BER values of Simulink transceiver systems	24
Figure 28	Screenshot of design summary of ZigBee physical layer modulation simulation	25
Figure 29	First part of HDL simulation	25
Figure 30	Part of HDL simulation that illustrates clock cycle delay	26
Figure 31	Entire HDL simulation of ZigBee physical layer modulator	26
Figure 32	Zoom of HDL simulation	27
Figure 33	Part of simulation that displays two clock cycle delay introduced by <i>bit to symbol</i> block	27

Figure 34	Possible solution to solve delays in modulator system	30
Figure 35	Diagram of <i>bit to symbol</i> block and its neighbors	35
Figure 36	Diagram of <i>DSSS LUT</i> block and its neighbors	35
Figure 37	Diagram of <i>De-interlace</i> block and its neighbors	35
Figure 38	Diagram of <i>serializer</i> block and its neighbors	36

Acronyms

ADC Analog to Digital Converter. 5–7, 15

AP Access Point. 9

ASIC Application Specific Integrated Circuit. 6, 7

AWGN Additive White Gaussian Noise. 23, 29

BER Bit Error Rate. v, vii, xi, 14, 23, 24, 29

BLE Bluetooth Low Energy. 2, 9

BRAM Block Random Access Memory. 11, 25

CLI Command Line Interface. 13, 14, 29

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. 9, 30

CSV Comma Separated Value. 21

DAC Digital to Analog Converter. 6, 7, 12, 15, 16, 19, 20, 26, 29

DDC Digital Down Converter. 5, 6

DSP Digital Signal Processor. 6

DSSS Direct-Sequence Spread Spectrum. iii, v, vii, viii, xi, xiii, 14–18, 23, 25, 29, 30, 35

DUC Digital Up Converter. 6

FFD Full Function Device. 8

FFT Fast Fourier Transform. 6

FIFO First In, First Out. 12, 13, 17–19, 25, 26, 30

FIR Finite Impulse Response. 5

FPGA Field Programmable Gate Array. 3, 7, 11, 12, 14, 17, 24, 29

GPP General Purpose Processor. 6, 7

HDL Hardware Description Language. xi, xiii, 3, 11, 17, 29, 30

IETF Internet Engineering Task Force. 9

IF Intermediate Frequency. 5, 6

IoT Internet of Things. v, xi, xiii, 1–3, 8, 9, 11, 13, 30

IP Intellectual Property. 17

IPv6 Internet Protocol version 6. 2, 9, 10

LE Logic Elements. 11

LoRa Long Range Radio. 2

LSB Least Significant Bits. 14

MAC Media Access Control. 8, 9, 13, 14

MLE Maximum Likelihood Estimator. v, xi, xiii, 16, 23, 29

MQTT Message Queue Telemetry Transport. 2

MSB Most Significant Bits. 14

NFC Near Field Communication. 2

OQPSK Offset-Quadrature Phase Shift Keying. vii, 15–20, 23, 26

OSI Open System Interconnection. 2, 3, 15, 17

PHY Physical. 8, 9, 13

PSDU Physical layer Service Data Units. 30

RAM Random Access Memory. 2

RF Radio Frequency. 5, 7, 12

RFD Reduced Function Device. 8

SDR Software Defined Radio. xi, 2, 3, 5–7, 11, 21, 29

SNR Signal to Noise Ratio. 23, 29

SPI Serial Peripheral Interface. 12

UART Universal Asynchronous Receiver-Transmitter. 12

VHDL Very High Speed Integrated Circuit Hardware Description Language. vii, 3, 17, 24, 29, 30

WSN Wireless Sensor Network. 9

XMPP Extensible Messaging and Presence Protocol. 2

Abstract

The evergrowing amount of IoT devices induce the need for multi-protocol gateways in order to create heterogeneous networks. Modern-day IoT gateways are able to handle vast amounts of data, but are difficult to reconfigure when adding nodes which use a new communication standard. The goal of this thesis is to take a look at the role of configurable hardware within software-defined radio gateways.

The approach to this thesis is divided into three parts. First, the architecture of the provided bladeRF x115 Software Defined Radio (SDR) is configured to receive and retransmit 6LoWPAN messages. This way, its capabilities as an IoT gateway were explored. Then, four different Simulink/Matlab simulations are constructed to mimic a ZigBee transceiver system. All their BER values are determined when passing a distorted channel and compared to implementations without DSSS and MLE. Finally, based on these results, a ZigBee physical layer modulator Hardware Description Language (HDL) simulation is constructed and tested.

The HDL simulation is able to convert 32-bit input sequences into 12-bit signed half sine samples within 14 system clock cycles. Although the design induces some delays, it meets the required functionality and forms a foundation for further research. Based on all the test results, it can be concluded that acceleration through configurable hardware will play a significant role within IoT multi-protocol gateways.

Abstract in Nederlands

De groeiende hoeveelheid IoT-apparaten leidt tot de behoefte aan multi-protocol gateways zodat heterogene netwerken gecreëerd kunnen worden. Huidige gateways kunnen grote hoeveelheden data verwerken, maar zijn moeilijker te herconfigureren wanneer een node met een ongekend protocol geïntroduceerd wordt. Het doel van deze thesis is de rol van configureerbare hardware binnen Software Defined Radio (SDR) gateways te verkennen.

De aanpak tot deze thesis is verdeeld in drie delen. Eerst wordt de architectuur van de voorziene bladeRF x115 aangepast zodat hij een 6LoWPAN-bericht ontvangt en heruitzend. Zo worden zijn mogelijkheden als IoT gateway verkend. Vervolgens worden er vier verschillende Simulink/Matlab-simulaties samengesteld die een ZigBee transceiver systeem imiteren. Al hun BER waarden worden bepaald nadat een boodschap verzonden wordt over een verstoord kanaal. Deze waarden worden vergeleken met implementaties zonder DSSS en MLE. Ten slotte, wordt er op basis van deze resultaten een ZigBee fysische laag modulator simulatie gebouwd.

De HDL simulatie kan 32-bit ingangssequenties converteren naar 12-bit halve sinus samples binnen 14 systeemklokcycli. Ondanks de geïntroduceerde vertraging zal het modulatorsysteem een uitgangspunt vormen voor verder onderzoek. Gebaseerd op alle testresultaten, kan er geconcludeerd worden dat versnelling via configureerbare hardware een belangrijke rol zal spelen binnen IoT-multi-protocol gateways.

Chapter 1

Introduction

1.1 Context and Problem Statement

The worldwide digitalization has a big impact on the expansion of the internet. Thanks to the ever growing amount of electronic smart devices, a new collection of technologies, called the Internet of Things (IoT), has emerged. It enables everyday objects like washing machines, lamps, door locks, etc. to communicate with each other over the internet without user interaction [14]. A multitude of applications have emerged. Farmers, for example, are now able to equip their livestock with sensors which monitor fertility, movement, behavior and even lactation. All this information will then be automatically transferred to the farmer's mobile device when he is within a 1 km range [19]. In the medical world the IoT is also gaining interest. Hospitals are equipping their beds with IoT nodes so that they can determine how many of them are occupied and where they are located at the moment. Another example to decrease downtime of critical medical devices integrates sensors, which constantly provide information about their status; allowing mechanics to anticipate upcoming failures. To prevent staff from searching for certain supplies; all machines, drugs, etc. will be located and listed on a personal mobile device. All these applications will lead to an increase in the hospital's efficiency and so limiting the wait time for the patients during their stay [12].

Besides these useful and possibly life-safing IoT projects, there are a lot of applications which are created for entertainment value. Griffin, for example, manufactures a Bluetooth connected toaster which is able to send a notification when the toast has reached its desired level of crispness [25]. Other collectors items are the connected comb, which improves your hair combing procedure or the WiFi-connected wine bottle sleeve. It is inevitable that all these new inventions, whether they are useful or not, will establish a substantial amount of new machine-to-machine like connections. These will lead to a tripling of the internet's size in the coming 5 years [7]. According to Gartner [11], there will be a total of 20.4 billion connected 'things' by 2020 as depicted in Table 1, while Statista [13] predicts an amount of 30.7 billion.

Table 1: IoT nodes installed by category (Millions of Units) [11]

Category	2016	2017	2018	2020
Consumer	3,963.0	5,244.3	7,036.3	12,863.0
Business: Cross-Industry	1,102.1	1,501.0	2,132.6	4,381.4
Business: Vertical-Specific	1,316.6	1,635.4	2,027.7	3,171.0
Grand Total	6,381.8	8,380.6	11,196.6	20,451.4

All these smart devices will not rely on one IoT standard to establish communication. To this day there exist a substantial amount of communication standards, which all enable machine-to-machine data exchange. Protocols that take care of the network access and physical layer (Open System Interconnection (OSI) model) are for example: ZigBee, Bluetooth Low Energy (BLE), Long Range Radio (LoRa), WiFi, Near Field Communication (NFC) and many more. Examples for the internet layer are: 6LoWPAN and Internet Protocol version 6 (IPv6). At the application level Message Queue Telemetry Transport (MQTT) or Extensible Messaging and Presence Protocol (XMPP) can be used. Of course, companies, hospitals and households which aim to introduce Internet of Things to their surroundings would like to use nodes from various manufacturers. These might all use a different combination of IoT standards, causing interoperability problems and preventing the formation of a heterogeneous IoT ecosystem [20]. The need for a central gateway which acts as a translator between the different protocols is undeniable. This hub will need to be able to explore different regions of the radio spectrum as not all the physical level IoT protocols work in the same frequency band. ZigBee, for example, uses 16 channels between 2.4 GHz and 2.4835 GHz worldwide and 10 channels between 902 MHz and 928 MHz in America as depicted in Figure 1. WiFi on the other hand, which relies on IEEE 802.11, occupies 5 distinct frequency ranges: 2.4 GHz, 3.6 GHz, 4.9 GHz, 5.0 GHz and 5.9 GHz.

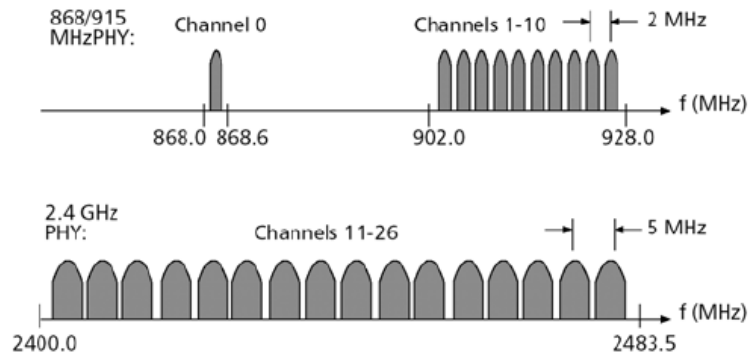


Figure 1: Channels of IEEE 802.15.4 standard [16]

Companies like DELL, IBM, Intel and many more provide configurable IoT gateways which enable heterogeneous IoT networking by scanning a wide range of the radio spectrum. All of them are powerful computers equipped with quad-core processors and a vast amount of Random Access Memory (RAM) to ensure reliable and fast data exchange. But as the size of data-packets continues to grow, due to the increasing complexity or the type of information that is being sent (e.g. video), gateways will need to be able to handle this growing amount of information without loss of speed. Manufacturers are of course able to upgrade their processors and so increase the performance of their product. But this requires an entirely new design of the gateway architecture. It is clear that other solutions to create IoT gateways have to be explored.

1.2 Objectives

The focus of this thesis lies on further exploring the role of Field Programmable Gate Arrays (FPGA) within Software Defined Radios (SDR) and how they can be used as a multi-protocol IoT gateway to connect nodes, which employ different protocols, to each other. To achieve this goal it is necessary to accomplish the following objectives:

- execute a literature study to gain further insight in the functioning of a SDR but also examine several IoT protocols which can be implemented;

- get better acquainted with the provided hardware by examining the device architecture and completing several Field Programmable Gate Array (FPGA) tutorials provided by the manufacturer;
- create a simple message repeater, which forms the foundation for a more complex multi-protocol SDR gateway;
- construct a HDL simulation, which mimics the functionality of a physical level ZigBee modulation system, based on a Matlab/Simulink model.
- determine which steps need to be taken to implement a fully functioning multi-protocol gateway on the provided hardware. This is based on the results of both the Matlab/Simulink and HDL simulations.

1.3 Method

This thesis will consist of two phases. First, there will be an extensive literature study to gain insight in the structure and working of a SDR and what role a FPGA can play within this concept. Furthermore, there will be a closer look at the existing IoT protocols and how their nodes are able to exchange information with each other and their supervising edge routers. As the IoT offers such a wide range of usable standards on all levels off the OSI model, only two of the most implemented standards, ZigBee and 6LoWPAN, were studied.

In the second part of the thesis, the development stage, there will be a closer look at the provided hardware and how it can be used to implement existing IoT protocols, such as ZigBee, on its FPGA. HDL files, more specifically VHDL files, will be written to create a simple physical layer message repeater, which is locked on a certain frequency. This is done to further understand the device structure. Next, based on the literature study, a low level ZigBee modulation and demodulation system is created in Matlab/Simulink. This makes the transition towards implementation of IoT protocols on a FPGA enabled SDR easier. Finally, using HDL simulation tools (Xilinx ISE Project Navigator), a Physical level modulation system is created in VHDL. Based on evaluations of this design, there will be decided which further steps will need to be taken to create a multi-protocol SDR gateway.

Chapter 2

Literature Study

2.1 Software Defined Radio

2.1.1 Receiver

A basic SDR receiver typically consist of five big building blocks: an antenna, a radio frequency tuner, an Analog to Digital Converter (ADC), a Digital Down Converter (DDC) and a digital signal processing block as depicted in Figure 2. The Radio Frequency (RF) tuner converts RF signals, received from the antenna, to Intermediate Frequency (IF) signals. These are then fed into the ADC, which changes the signal's domain and provides digital samples at its output. Next, the samples enter the DDC-block which has three smaller components: a digital mixer, a digital oscillator and a Finite Impulse Response (FIR) Low-Pass filter. The digital mixer and oscillator multiply the digitized cosine with the phase channel and the digitized sine with the quadrature channel, this respectively results in the sum and difference frequency. Then, the FIR Low-Pass filter is applied to both the phase and quadrature channels. This results in a frequency band which is substantially reduced to its desired baseband, hereby retaining all its information [24]. Finally, the digital signal processing block demodulates and decodes the baseband signals so that the embedded information is interpretable and ready to be handled.

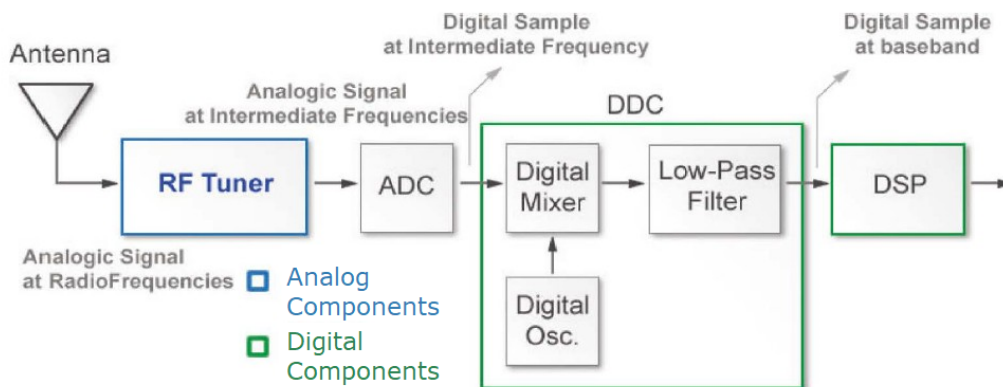


Figure 2: Topology of basic software defined radio receiver [21]

2.1.2 Transmitter

A SDR transmitter basically consists of the same elements as the receiver. The ADC is replaced by a Digital to Analog Converter (DAC) and the Digital Down Conversion block is replaced by a Digital Up Converter (DUC). Moreover, the DUC consists of three blocks: an interpolation filter, digital mixer and digital oscillator. Its purpose is to upscale the baseband samples coming from the digital signal processing block to intermediate frequency (IF) signals, which are then fed into the DAC, amplified and send out by the antenna. All this can be seen in Figure 3.

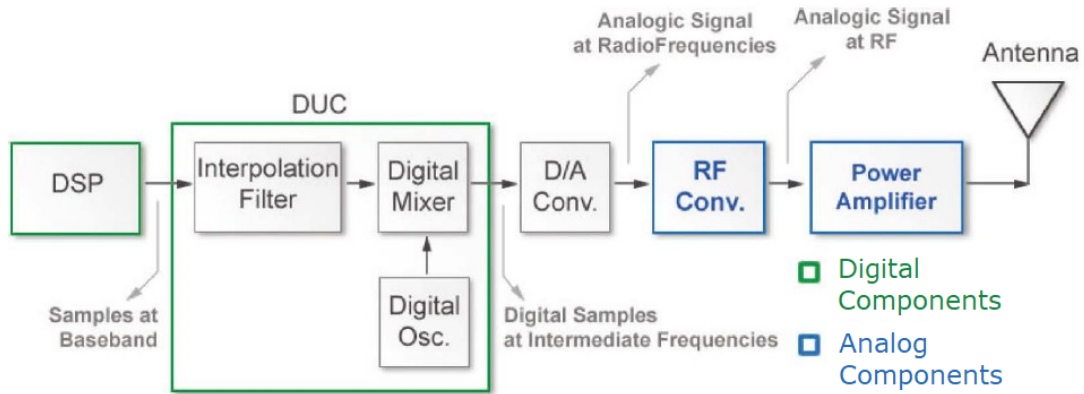


Figure 3: Topology of basic software defined radio transmitter [21]

2.1.3 Digital Signal Processing

As discussed earlier, both the transmitter and receiver respectively collect from or feed samples to the digital signal processing block, which represents the main functionality of the SDR. It will implement functions like (de)modulation, (de)coding, filtering and more. As its name suggests, a Software Defined Radio needs to be reconfigurable. Therefore, when implementing a SDR, there are multiple alternatives to choose from:

- an **Application Specific Integrated Circuit (ASIC)** is specifically developed for a certain application, which means it is not (re-)programmable and contradicts with the basic principle of a SDR. However, their economical advantage when produced in large numbers and low power consumption has made them worth considering [23]. They are generally used for analog to digital conversion, filtering and digital down conversion (DDC) as these modules require a specialized hardware structure to operate in real-time;
- a **General Purpose Processor (GPP)** is mostly found in personal computers as it possesses the ability to process digital signals and so edit text, display multimedia, etc. Because General Purpose Processors are not integrated with a specific programming languages nor software, they are very flexible and easy to configure. In SDRs they are used to decode received messages, perform Fast Fourier Transform (FFT), make decisions or analyze data. Compared to the ASICs, what GPPs lack in process intensity and energy efficiency, they make up for in flexibility [23];
- a **Digital Signal Processor (DSP)** is a specialized microprocessor that is optimized for the operational needs of digital signal processing. Although its flexibility is significantly lower than the GPP's, the DSP has a better performance when used in a SDR [23];

- a **FPGA** is mostly used when an application requires high computing power, which it acquires through hardware parallelism, at a relatively low power consumption. Bhandari et al. [3] held a case study comparing the size, power and cost of a Xilinx Virtex4-SX25 FPGA, a Power6 GPP and a Freescale SC8144 ASIC. The FPGA operated at a frequency of 500 MHz, has a computation rate of 256 GMAC/s and power consumption of 4 W. Its total size of 726 mm² is less compared to the 841 mm² of the Freescale SC8144, which operates at 1 GHz, has a computation rate 16 GMAC/s and a power consumption of 4.5 W. The Power6 operates at a frequency of 4.7 GHz and has a computation rate of 120 GFlop/s. Its power consumption was not measured during the tests. Overall, the FPGA has a superior flexibility and power consumption to computation rate. But, depending on the implementation a GPP or ASIC may still be more suited [3].

2.1.4 Ideal Software Defined Radio

Most of the current radio communicating systems are hardware based. This implicates that their radio functionalities are not reconfigurable and that they are limited to the wireless protocols that were coded onto the device. In addition, one single failure in the software, hardware or firmware would make the entire system unusable [21]. A software defined radio (SDR) is able to circumvent these two complications because its physical layer functions are software determined. An ideal SDR, depicted in Figure 4, would at the receiver side consist of an antenna and an Analog Digital Converter (ADC). Likewise, the transmitter would contain a Digital to Analog converter (DAC) and an antenna. A reprogrammable processor would handle all the signal processing functionalities [9],[21]. Until today, however, no manufacturer has succeeded to create a functioning SDR that meets these requirements because of the following reasons [9]:

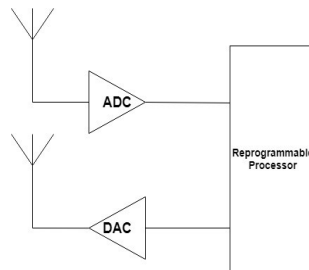


Figure 4: Topology of ideal software defined radio

- the antenna of an ideal SDR should be able to explore the entire frequency spectrum (from < 1 MHz to > 300 GHz). Designing such wideband-antenna is to this day still impossible;
- a key feature of the RF front end is to select the relevant signals and reject the interferers. The electromechanical structures, used by the antennas and filters for channel selection, are hard to tune dynamically. This is why the entire spectrum will be digitized. Nyquist's criterion, which states that the sample frequency must be twice the maximum signal frequency, implicates that the performance of currently available analog to digital converters is nowhere near;
- because the signal strength of the region of interest can be much lower than the strength of interfering signals, power differences can be substantial (up to 120 dB) [9]. This makes it nearly impossible to isolate the desired signal.

2.2 Internet of Things Protocols

As IoT nodes are constrained towards power consumption and processing power, the need for dedicated communication protocols emerged. Nowadays, there are a multitude of Internet of Things standards which are easily adopted by cloud service providers, software developers and device manufacturers. All of the protocols differ in factors such as security, range, power demands, etc. The following section will give an overview of two major communication technologies that are being used to this day.

2.2.1 ZigBee

ZigBee is a short-range Internet of Things communication standard, developed by the ZigBee Alliance. It is specially designed for low-power, low-cost and low-data-rate wireless networks which occupy a wide range of battery-powered, low capability, low-cost sensors, actuators and controllers [17]. These devices operate in 3 different frequency bands: 868 MHz, 915 MHz or 2.4 GHz [5], [22]. Applications that might use this standard are home automation/monitoring and industry control systems. ZigBee supports three types of network topologies: mesh, tree and star which are depicted in Figure 5. Using the mesh topology results in higher reliability and a wider range [5]. Every topology consists of a coordinator and a Full Function Device (FFD), which is the node that creates the network, collects all the transmitted data and assigns addresses to newly added devices. A router (FFD) acts as an intermediate device. It joins a network that already exists and relays data from its ‘children’ to the coordinator and the other way around. The third and final component of a ZigBee topology is an end device, Reduced Function Device (RFD) [22]. This simple node is not able to have children nor able to forward packets.

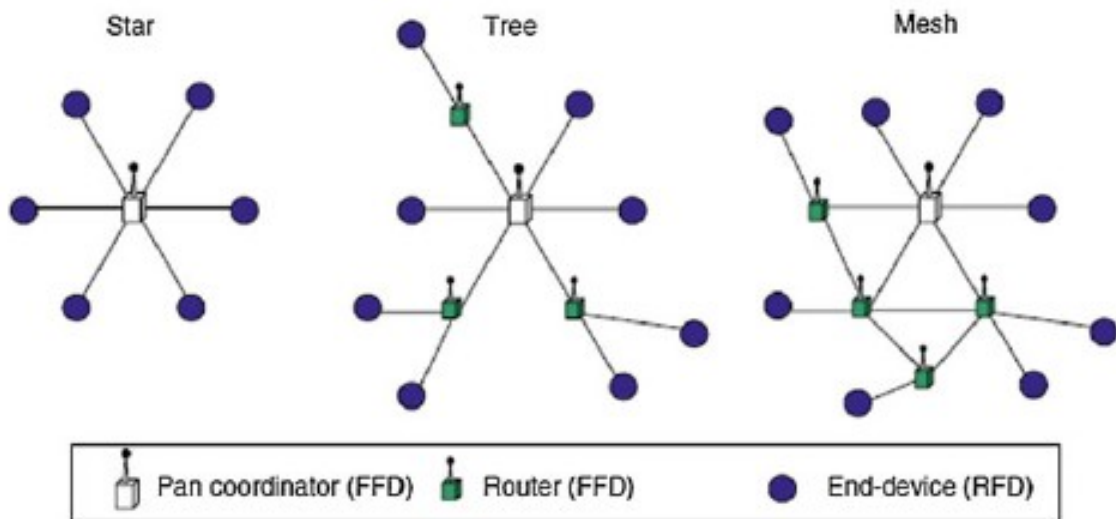


Figure 5: Possible ZigBee network topologies [22]

Figure 6 depicts the ZigBee protocol stack, which consists of 4 layers: application, network, Media Access Control (MAC) and Physical (PHY). IEEE 802.15.4 defines the specifications for the physical and medium access control layer, but it is not involved in higher networking layers. Application, network and security layer related specifications are controlled by the ZigBee standard [17]. Therefore, a ZigBee communication network conforms the IEEE 802.15.4 standard in order to function efficiently.

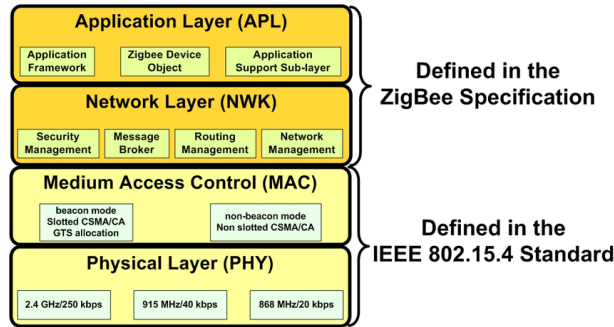


Figure 6: ZigBee protocol stack [1]

2.2.2 6LoWPAN

6LoWPAN is, just like ZigBee, a low-cost, low-power, short range and low-data-rate Internet of Things open standard. This protocol, created by the Internet Engineering Task Force (IETF), provides IPv6 routing capabilities over a IEEE 802.15.4 Wireless Sensor Network (WSN) [15]. Figure 8 depicts an IPv6 enabled 6LoWPAN mesh network. An Access Point (AP) acting as a IPv6 router provides an uplink to the internet. Typically it is connected to several devices such as PC's and servers. Communication between the IPv6 network and the 6LoWPAN network is established by a 6LoWPAN edge router. Its main functions are: data exchange between locally connected devices, data exchange between 6LoWPAN devices and the internet and maintenance of the radio subnet [15]. Connections to other networks, such as ZigBee and BLE, are possible through either IP routers or 6LoWPAN edge routers that forward IP datagrams to an IoT gateway, which translates the messages. Within a 6LoWPAN network there are typically 2 devices included: routers and hosts. Routers are able to collect and forward data to other nodes in the same network. While hosts, also known as end devices, are not able to direct traffic. They are usually 'sleeping' and so often check in with their parents for data [8]. Figure 7 depicts the structure of the 6LoWPAN protocol stack.

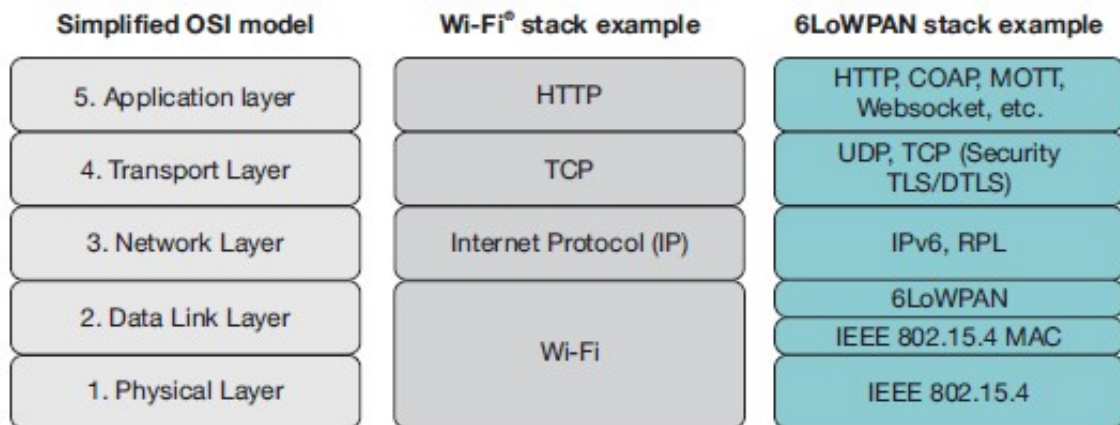


Figure 7: 6LoWPAN protocol stack [15]

Just like ZigBee, 6LoWPAN adopts the IEEE 802.15.4 standard to take care of the Physical (PHY) layer and Media Access Control (MAC) layer. The PHY layer defines a total of 27 available channels which are allocated into 3 different frequency bands: 868 MHz, 915 MHz or 2.4 GHz. The tasks of the MAC layer consist of Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), beacon generation and synchronization, etc. [8]. The 6LoWPAN standard is part of the data link layer (OSI model). It takes care of the adaptation from IEEE 802.15.4 to IPv6 and the other way

around. During transmission it detects and corrects errors that may occur, this way it ensures a reliable data exchange between two directly connected nodes. The upper tiers of the OSI model such as the network, transport and application layer are respectively taken care of by the IPv6 standard, UDP/TCP and HTTP/COAP/MOTT/etc.

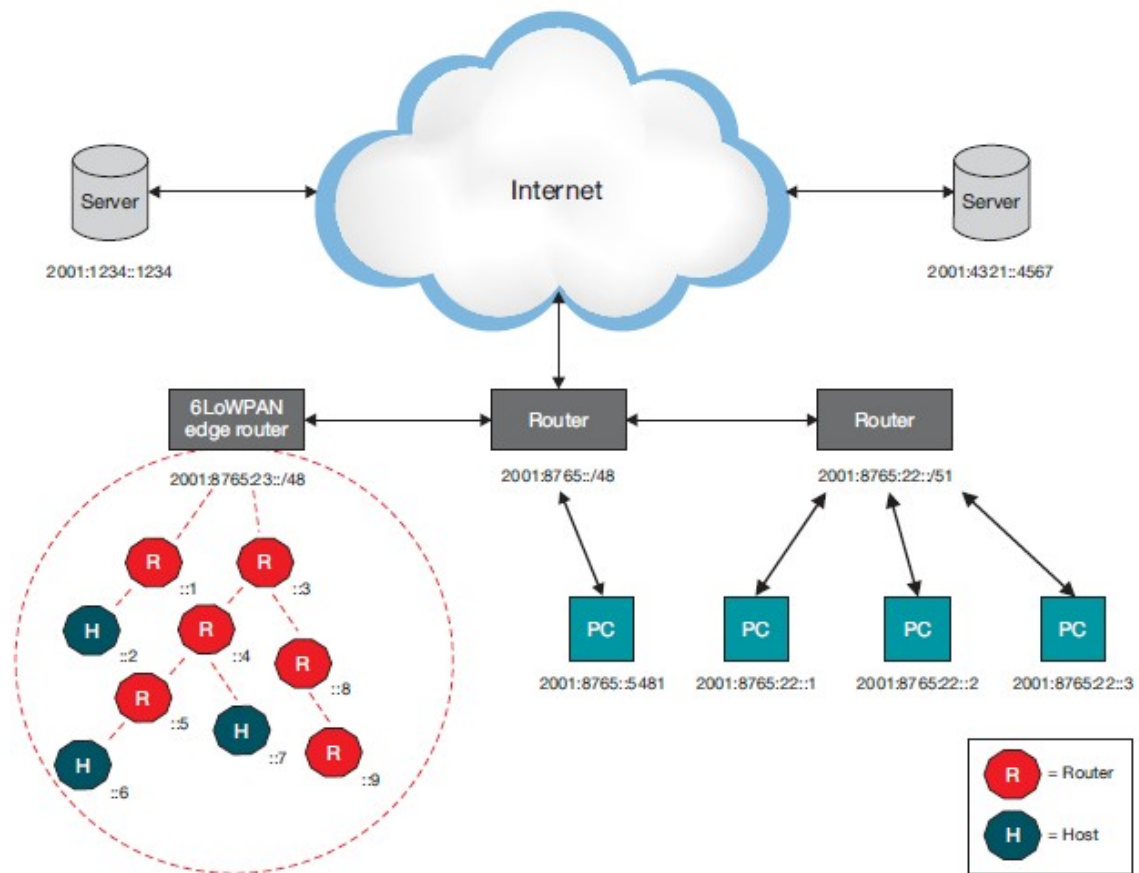


Figure 8: 6LoWPAN mesh topology [15]

Chapter 3

Materials and Method

After explaining the functioning of a SDR and how a FPGA can enhance its performance, the first part of this section takes a look at the hardware that will be able to support a reconfigurable IoT gateway. Next, section 3.2 describes the implementation of a simple message repeater on the used hardware, shaping a clear image of its possibilities. Finally, section 3.3 and 3.4 explain how a physical layer (de)modulation system can be constructed through use of Matlab/Simulink and Xilinx ISE simulation tools. This design can form a base for a reconfigurable multi-protocol SDR gateway.

3.1 BladeRF x115

The bladeRF x115, depicted in Figure 9, is a powerful, but yet affordable Software Defined Radio manufactured by Nuand. It is able to detect frequencies ranging from 300 MHz up to 3.8 GHz without the need for an external expansion board. The bladeRF is equipped with a LimeMicro LMS6002D fully integrated RF transceiver, which is able to detect anything from simple FM audio to 4G LTE. The fact that its transmitter and receiver sides are both separately configurable, allows users to receive messages from one protocol node, process the information and retransmit the data to a node using another protocol. At peak performance the bladeRf is capable of occupying 28 MHz of bandwidth, making it an ideal solution for reconfigurable, multi-protocol gateways. The FPGA-chip equipped on the bladeRF x115 is an Altera Cyclone IV, which possesses 114,480 Logic Elements (LE) and a total amount of 3,888 kbits Block Random Access Memory (BRAM) [6]. It acts as an interface between the LMS front end and FX3 microcontroller and is capable of extensive digital signal processing. All of the bladeRF libraries, utilities, firmware, schematics and platform HDL are available online under an open source license. This makes it a suited tool for exploring the implementation of IoT protocols on reconfigurable hardware.



Figure 9: BladeRF x115 device [6]

3.2 Message Repeater

3.2.1 Default Architecture

The FPGA includes an Altera NIOS II soft core processor which is clocked at 80 MHz for command and control. The main purpose of the default FPGA image is to shuffle captured I/Q samples between the LMS6002D transceiver and FX3 microcontroller. Other occupations are:

- interacting with expansion boards,
- controlling SPDT RF switches lying between the LMS6002D transceiver and the SMA RX/TX ports,
- configuring the LMS6002D via Serial Peripheral Interface (SPI) based on commands it received from the FX3 microcontroller over Universal Asynchronous Receiver-Transmitter (UART),
- controlling the Si5338 clock generator chip,
- controlling the VCTCXO DAC [4].

Figure 10 depicts a simplified diagram of the modules that the I/Q samples flow through in the default image. At the LMS6002D block, I/Q samples consist of a 12 bit signed I-sample and a 12 bit signed Q-sample. When entering the FPGA, I/Q samples are sign extended to 16 bit, for ease of use in the FX3 firmware. At the transmit side, samples are sent from the FX3 to the FPGA over a bidirectional bus to the **fx3 gpif** module. This block handles the driving and releasing of the received samples which are then fed into an asynchronous First In, First Out (FIFO) to realize a clock domain crossing. Samples then pass the **fifo reader** block where they are split into the I- and Q-samples. Corrections regarding DC offset or other imperfections are amended in the **tx iq correction** module. Finally, to create a time-multiplexed output signal, the I- and Q-samples are firstly sign contracted to 12 bits and then at alternating clock cycles fed into the LMS6002D chip.

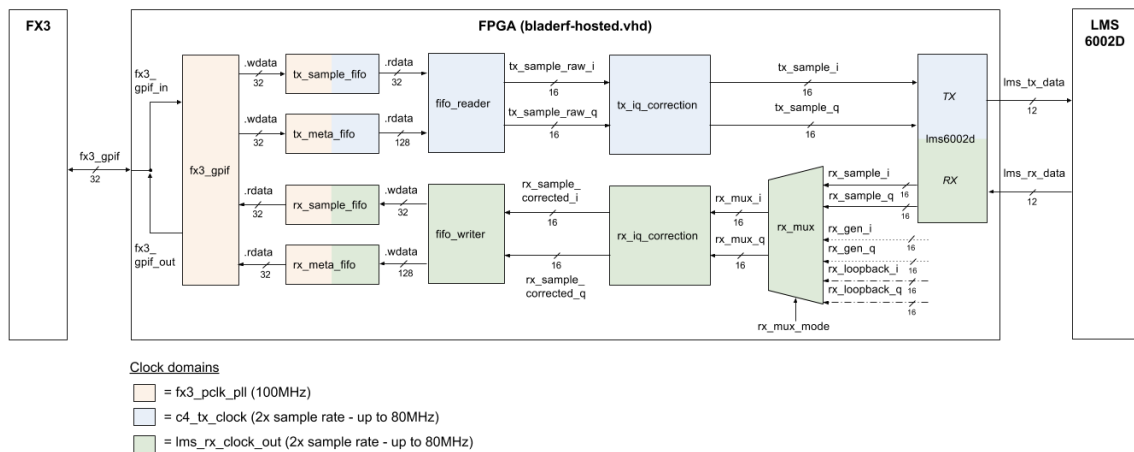


Figure 10: Diagram of default FPGA image [4]

When samples are being received by the default image, they enter the FPGA as 16 bit sign extended signals through the **rx mux** process. Based on the mode the FPGA is in, the multiplexer will pass different signals. In RX MUX NORMAL mode, raw I- and Q-samples are passed. The RX MUX 12BIT COUNTER and RX MUX 32BIT COUNTER mode will pass 12-bit/32-bit counter signals which are used to detect if samples are being lost in the USB connection. The blocks that follow after the multiplexer are basically the reverse of the transmit side.

3.2.2 Repeater Architecture

The reason why a simple message repeater was created is to form a base for later implementation of IoT protocols. In the default image, samples are being collected, shuffled and passed on to the host computer which is controlling the bladeRF x115. When transmitting, samples are loaded from a file onto the board via the FX3 microcontroller, shuffled and send out by the radio front end. In a stand-alone, multi-protocol gateway, the bladeRF has to operate without the need for a host computer which controls the center frequency, bandwidth, sample rate, etc. To come as close as possible to this functionality some minor changes were made to the default architecture. First, as depicted in Figure 10 the samples always pass through the **fx3 gpif** block and so communicate with the host PC. Not all control signals are displayed on the diagram, but to limit computer interaction both the sample data and meta data FIFOs are connected to each other. As depicted

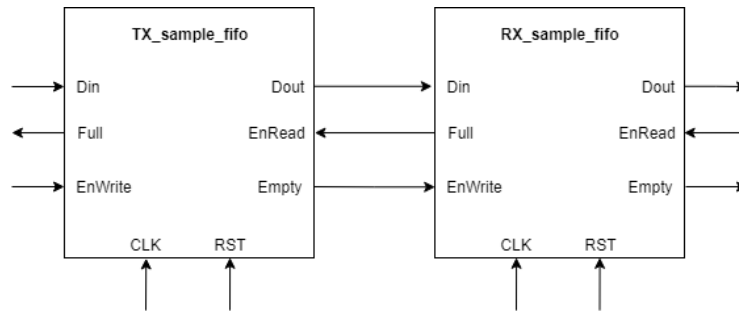


Figure 11: Connection between *tx sample fifo* and *rx sample fifo*

in Figure 11, the data coming from the receiver side will be passed on to the transmit side when both the **rx sample fifo** is not full and the **tx sample fifo** is not empty. Other signals, coming from the **FX3** module, are left untouched as they are needed to configure the bladeRF at startup via the Command Line Interface (CLI). Blocks such as the **fifo reader**, **tx iq correction**, **fifo writer**, **rx iq correction** and **rx mux** are still in use, as they are only responsible for I/Q sample shuffling and splitting. Figure 12 depicts the changed subsection of the message repeater architecture.

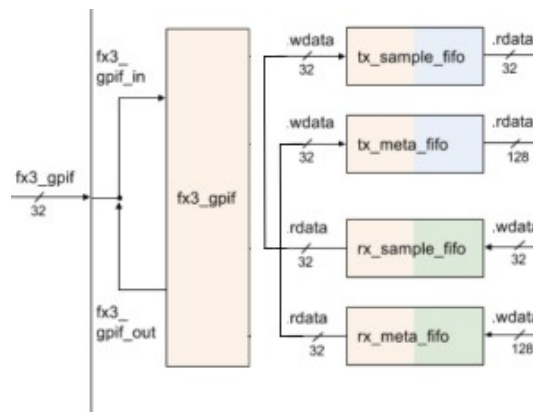


Figure 12: Zoomed-in view of architecture message repeater [4]

To test the functionality of the architecture, a programmable Zolertia Z1 node is configured to emit a repetitive 6LoWPAN message. As mentioned in 2.2.2, 6LoWPAN relies on IEEE 802.15.4 for the PHY and MAC layer. Worldwide, the available frequency spectrum for this standard ranges from 2.4 GHz to 2.4835 GHz with 16 channels which are 2 MHz wide. For this test, channel 26 was selected. A simple script was written to tune the bladeRF x115 so that it is able to detect

and retransmit the messages. The center frequency and sample rate are respectively set to be 2480 MHz and 1 MHz. While trying to set the bandwidth to 2 MHz, as is required for an IEEE 802.15.4 channel, the CLI is not able to and clamps it to 2.5 MHz. Figure 13 summarizes the device settings for the repeater architecture.

```

RX Bandwidth: 2500000 Hz
TX Bandwidth: 2500000 Hz

RX Frequency: 2479999999 Hz
TX Frequency: 2479999999 Hz

GPIO: 0x0000002f
LMS Enable: Enabled
LMS RX Enable: Enabled
LMS TX Enable: Enabled
TX Band: High Band (1.5GHz - 3.8GHz)
RX Band: High Band (1.5GHz - 3.8GHz)
RX Source: LMS6002D

Loopback mode: none
RX mux: BASEBAND_LMS - Baseband samples from LMS6002D

RXLNA Gain: 6 dB
RXVGA1 Gain: 30 dB
RXVGA2 Gain: 3 dB
TXVGA1 Gain: -14 dB
TXVGA2 Gain: 0 dB

Sampling: Internal
RX sample rate: 1000000 0/1
TX sample rate: 1000000 0/1

Current VCTCX0 trim: 0x8e56
Stored VCTCX0 trim: 0x8e56

VCTCX0 tamer mode: Disabled

Expansion GPIO register: 0xffffffff
Expansion GPIO direction register: 0x00000000

```

Figure 13: BladeRF x115 device settings for message repeater architecture

3.3 ZigBee Transceiver Matlab/Simulink Simulation

The transceiver is designed in Matlab/Simulink using multiple fundamental building blocks provided by all the available toolboxes (e.g. Communications system toolbox). This shows how a cost effective and yet efficient radio transceiver can be constructed using complex modulation schemes. All the used building blocks are provided with an extensive amount of documentation and so give more insight in how a FPGA based solution can be created. This section will first explain what the requirements for a physical layer ZigBee transmitter are and how they are realized in Simulink. Then will be explained how a compatible physical layer ZigBee receiver is constructed. Finally, the performance of the transceiver system is tested by calculating the BER.

3.3.1 Transmitter

Figure 14 depicts a simplified flow diagram of the ZigBee transmitter. The input bitstream coming from the MAC-layer enters the transmitter at a rate of 250 kbps. It will first pass a bit to symbol mapping block. Here, bytes are split into two 4-bit symbols, the 4 Most Significant Bits (MSB) and 4 Least Significant Bits (LSB). Each 4-bit symbol then enters the symbol to chip block where they will be spread to a certain 32-bit value. This technique is called DSSS. IEEE 802.15.4 uses a predefined mapping table of sixteen pseudorandom noise 32-bit values to realize DSSS, as illustrated in Table 2. Spreading is applied to improve the resistance to (un)intended jamming of the signal, increase the frequency of the information signal to 2 kchips/s and improve the signal to noise ratio.

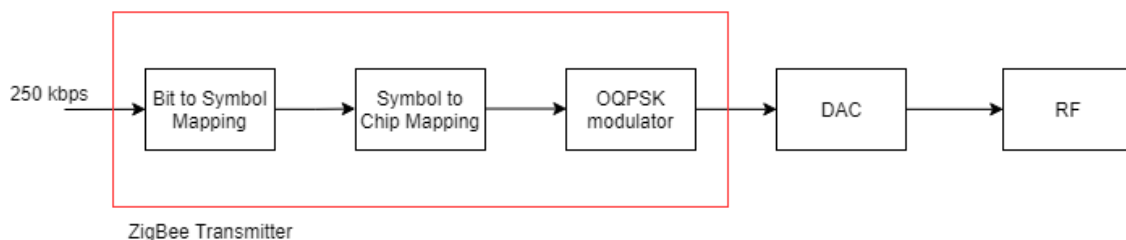


Figure 14: Simplified diagram of ZigBee transmitter

Table 2: DSSS mapping table [22]

Data Symbol ($b_3 b_2 b_1 b_0$)	Chip Value ($c_{31} c_{30} c_{29} \dots c_1 c_0$)
0000	01110100010010101100001110011011
0001	01000100101011000011100110110111
0010	01001010110000111001101101110100
0011	101011000011100110110111101000100
0100	11000011100110110111010001001010
0101	00111001101101110100010010101100
0110	10011011011101000100101011000011
0111	10110111010001001010110000111001
1000	11011110111000000110100100110001
1001	11101110000001101001001100011101
1010	11100000011010010011000111011110
1011	00000110100100110001110111101110
1100	01101001001100011101111011100000
1101	10010011000111011110111000000110
1110	00110001110111101110000001101001
1111	00011101111011100000011010010011

The 32-bit pseudorandom noise sequences then enter the OQPSK block where they are demultiplexed into an Inphase (I) signal and a Quadrature (Q) signal [10], [22]. These signals are then used for half-sine pulse shaping. A chip ‘1’ is shaped into a positive half sine, while a chip ‘0’ is shaped into a negative half sine according to the following formula [2]:

$$p(t) = \begin{cases} \sin(\pi \frac{t}{2T_c}) & 0 \leq t \leq 2T_c \\ 0 & otherwise \end{cases}$$

To ensure a continuous phase change in the output signal the Q-samples are given an offset of half a chip interval. The purpose of pulse shaping is to match the signal’s frequency spectrum to the available channel bandwidth [10]. Finally, the OQPSK modulated I- and Q-samples are passed on to the DAC where they are converted to an intermediate frequency and eventually send out by the radio front end.

3.3.2 Receiver

Figure 15 illustrates a simplified diagram of a ZigBee receiver. Analog samples are captured by the radio front end, digitized by the ADC and then fed into the OQPSK demodulator block to convert half sine waves into chips. These chips are then grouped and used as input for the de-spreading block. DSSS-despreading is realized by correlating the input values to 32-bit pseudorandom noise sequences in a lookup table. Its output are 4-bit data samples, which are send to the higher layers of the OSI model for further processing.

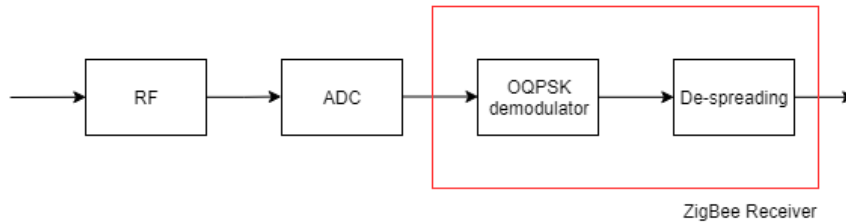


Figure 15: Simplified diagram of ZigBee receiver

3.3.3 Simulink Implementation

The input for the Simulink model is generated by a random integer generator with a sample time of $1.25e-7$ seconds (250 kbps). Its output type is set to binary because the ease of use in further calculations. A simple buffer is used for bit to symbol mapping. It collects 4 samples and outputs them as a frame to the bit to integer converter. This block creates an integer number between 0 and 15, which serves as input for the DSSS lookup table (Table 2). Finally, the 32-bit pseudorandom noise sequences are passed on to the OQPSK block, which is provided by the Matlab communications system toolbox. Because the bladeRF has an on-board DAC, the transmitter system stops here. The modulation delay of the entire system is directly correlated to the setting of the OQPSK block. That is why four distinct transmitters, which all use the same building blocks as depicted in Figure 16 but different OQPSK configurations, were constructed. Their settings and respective delays are summarized in Table 3.

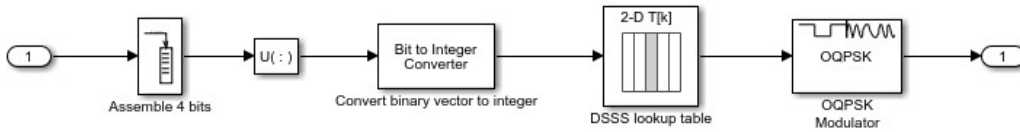


Figure 16: Simulink system ZigBee transmitter

The input for the ZigBee receiver model is the output of the transmitter system after it has passed through a configurable AWGN channel. This, as the name implies, adds white Gaussian noise to the data that is being transmitted. Upon entering the receiver system, data will first pass the OQPSK demodulator block, which at its output provides a 32-bit pseudorandom noise sequence. This value then enters one of four different modules to compensate the delay induced by the modulation settings depicted in Table 3. As the AWGN channel adds noise to the received signal, the OQPSK demodulator may provide a wrong ‘translation’ at its output. A MLE is implemented to prevent the receiver system from using this wrong 32-bit pseudorandom noise sequence that does not match one of the sixteen values in Table 2. In turn to find out the corresponding symbol, the MLE block calculates the minimum *hamming distance* between each incoming 32-bit chip sequence P and all the pseudorandom noise sequences PN . *Hamming distance* is the number of different positions between two bit strings of equal length [22]. Finally, the output of the MLE is passed on to the DSSS lookup table for chip to symbol conversion as illustrated on Figure 17.

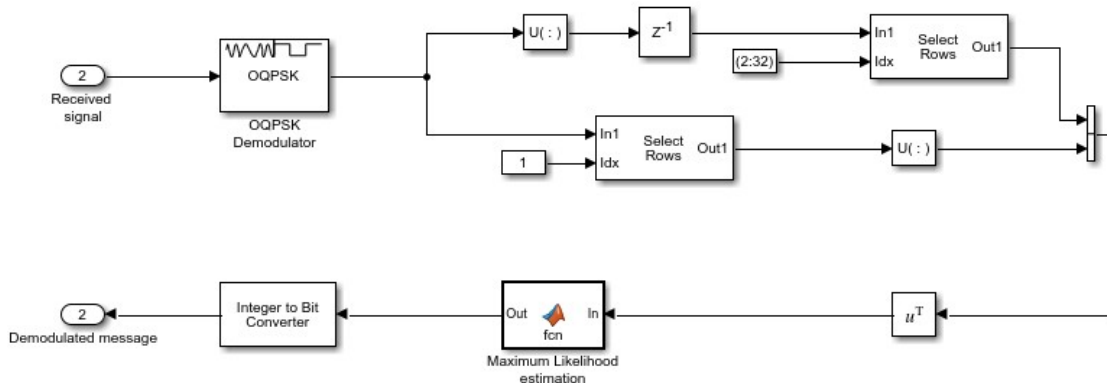


Figure 17: Simulink single rate ZigBee receiver system with integer values

Table 3: End-to-end delays created by (de)modulation configurations

Rate Option	Input/Ouput data type	End-to-end delay (<i>in samples</i>)
Enforce single-rate	Integer	1
	Bit	2
Allow multi-rate	Integer	length data + 1 + 1
	Bit	length data + 2 + 2

3.4 Xilinx ISE Simulation

This section describes the transition from a simplified physical layer ZigBee transceiver to a functioning VHDL simulation. By studying the Matlab/Simulink implementation and its used building blocks, a better understanding of what an IEEE 802.15.4 OQPSK (de)modulator does and how it functions was acquired. Although, Matlab provides conversion tools from Simulink to HDL code (HDL-coder), none of the modules used were translatable. That is why every component used in the simulation is either written from scratch or Intellectual Property (IP) provided by Xilinx. The software tool used to generate the VHDL-files is the free Xilinx ISE WebPACK, which provides full HDL synthesis, simulation, device fitting, implementation and JTAG programming. As the embedded Cyclone IV FPGA, installed on the bladeRF, is manufactured by Intel (Altera), only the synthesis and simulation tools can be explored. Intellectual property modules generated in the project are of course not usable for implementation on the bladeRF x115, but to solve this problem Intel provides similar IP blocks. Before constructing the entire modulator, the functionality of all VHDL components was tested and validated by several testbenches which described different kind of situations. To understand the flow of the created architecture this section first takes a look at the entire modulator and then explains the role of each component individually. Figure 18 illustrates a simplified diagram of the architecture. Not all control signals are included because they would affect the readability of the chart.

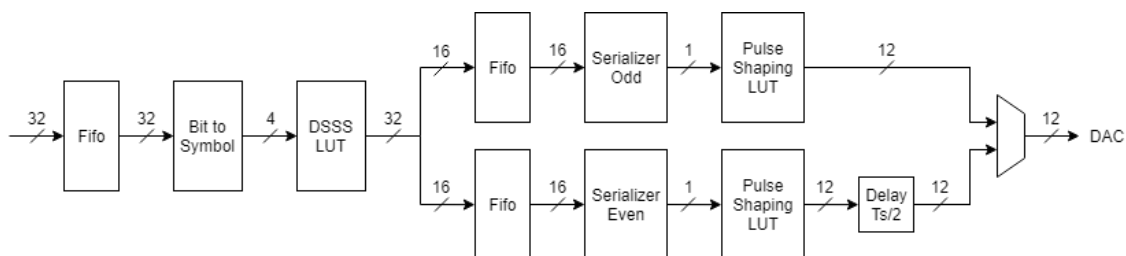


Figure 18: Simplified VHDL modulator system

3.4.1 Bit to Symbol Conversion

Like in 3.3.1, the input signal needs to be converted to 4-bit sequences in order to enter the DSSS block. The **bit to symbol** module takes a 32-bit signal coming from either the receiver side or a higher layer of the OSI model. Because it takes multiple clock cycles to process one 32-bit signal, a FIFO is needed to store all the input sequences. At every rising edge of the clock the **bit to symbol** block checks whether the FIFO is empty or not. In case that the FIFO is not empty, the block reads a 32-bit sequence and at every clock cycle it outputs a 4-bit symbols until the entire word is processed. In the other case were the FIFO is empty, undefined values are passed through. The entire duration of processing one 32-bit sequence is ten clock cycles. The first two are lost due to the reason that the **empty** signal coming from the FIFO has to be read and the **read** signal going to the FIFO has to be written. The other eight ($32/4 = 8$) cycles are dedicated to sending

out the 4-bit values. In order to prevent following blocks from using the undefined or zero values, that are being passed through when the FIFO is empty, a **valid** signal is set to high when valid 4-bit symbols are vented. A more detailed schematic on how the FIFO is connected to the bit to symbol block is depicted in figure 35 of Appendix A.

3.4.2 DSSS Lookup Table

Just like in the Matlab/Simulink simulation the DSSS lookup table is based upon Table 2. At every rising clock edge this block will assign the output of the **bit to symbol** module to an internal signal. This enters a case statement to output the right DSSS sequence. There is no noticeable delay between input and output as is illustrated in Figure 19. The following 4-bit symbols are loaded at successive rising edges of the clock: *1000*, *1010*, *0000*, *0110*. Their corresponding 32-bit pseudorandom noise sequences are provided at the output within the same clock pulse. A more detailed schematic on how the DSSS lookup table is connected to its neighboring modules is depicted in Figure 36 of Appendix A.

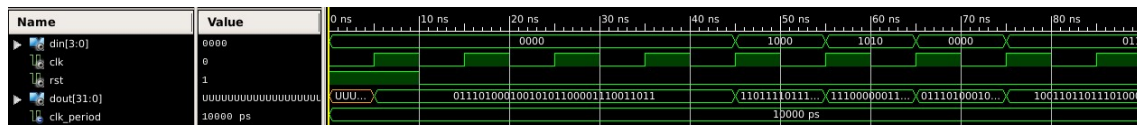


Figure 19: Results of testbench for DSSS module

3.4.3 De-interlace

Although this block is not depicted in figure 18, it plays a significant role in the OQPSK modulation. At every rising clock edge it takes the output coming from the DSSS block and separates the even bits from the odd bits. This results in a 16-bit I-sample and a 16-bit Q-sample. These are send to the pulse shaping module. The following example further explains the functionality:

- **32-bit input sequence:** 10101010111111110011010100101000,
- **16-bit even (I-sample):** 0000111101110000,
- **16-bit odd (Q-sample):** 1111111101000110.

Figure 20 is the simulation result corresponding to the previous example. It is clearly visible that there is no delay introduced from input to output. The odd and even values are calculated within the same clock cycle as the input is set. Further details on how this block is connected to its neighbors are depicted in Figure 37 of Appendix A.

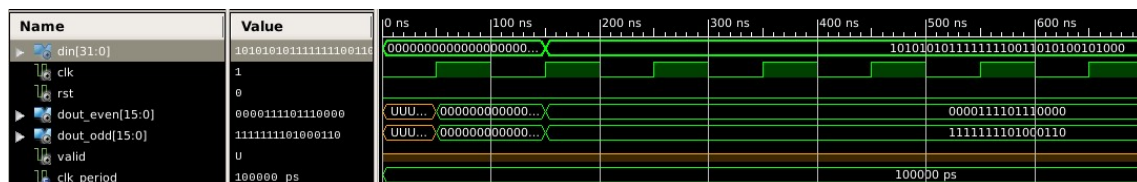


Figure 20: Results of testbench for De-interlace module

3.4.4 Serializer

For the pulse shaping block to do its work, the incoming 16-bit I- and Q-samples need to be serialized. Because it takes multiple clock cycles to process one 16-bit sequence, a FIFO is introduced to store the incoming signals. At every rising clock edge the **serialize** block checks whether the FIFO is empty or not. In case the FIFO is filled, the module reads a 16-bit value and releases one bit every clock cycle until the entire sequence is processed. This outputting happens at a another clock rate than the reading of the values in order for the OQPSK modulator to work, but this is explained in 3.4.5. The different clocks are realized by implementing a clock divider which uses the fastest clock as its input. At every rising edge it increases a counter until a certain value is reached. It then sets its output to high and resets the counter. By receiving and sending out a **valid** signal, pulse shaping of undefined or zero values is prevented. The duration of processing an entire 16-bit sequence is eighteen clock cycles, as can be seen on figure 21, which depicts the results of a testbench where *1010110011000111* is loaded in at the first clock cycle. For this test the output clock period is set to 20 ns, while the input clock period was 10 ns. These values are chosen just for test purposes and are not related to an OQPSK modulator in any way. The first two cycles are lost due to the reason that the **empty** signal coming from the FIFO has to be read and the **read** signal has to be written. Remaining cycles are dedicated to sending out the bits. As indicated by the yellow markers on figure 21, the total time the **validbit** signal is high is 320 ns, corresponding to $320 \text{ ns} / 20 \text{ ns} = 16$ cycles. More information about how the **serialize** block is connected to its neighbors is depicted in Figure 38 of Appendix A.

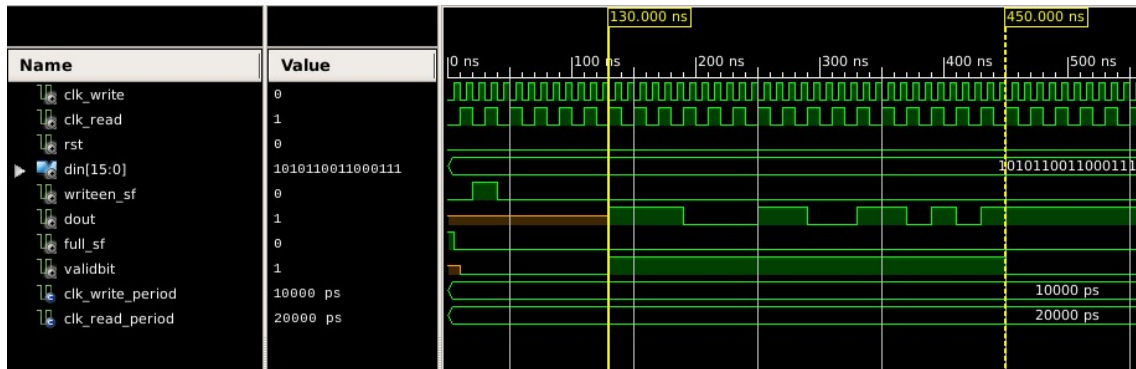


Figure 21: Results of testbench for serializer module

3.4.5 Pulse Shaping

This module is the last block of the OQPSK modulation system. A bit stream is received and converted to half sine samples, which serve as input for the DAC. All this is done to limit the occupied bandwidth of the effective transmission and keep intersymbol interference under control. The **pulse shaping** module receives the serialized bits, coming from the previous module, at a different clock rate than the entire architecture. Every incoming bit has to correspond with either a negative half sine pulse when zero or a positive half sine pulse when one, as depicted in Figure 22. Because the bladeRF allows the sample rate to be a rounded plural of the symbol rate (62.5 ksymbols/s); instead of implementing the formula explained in 3.3.1, a simple lookup table can be used.

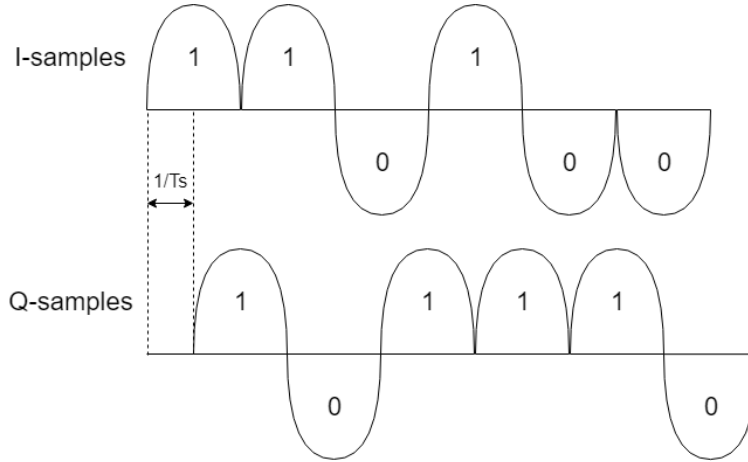


Figure 22: Half sine pulses generated by the OQPSK modulator

The LUT contains eight different sine samples which all need to be outputted within a single clock pulse. To alleviate this problem, the bits are supplied at a clock rate which is eight times the system clock (clock period: $\frac{1}{62.5k\text{symbols/s}} = 16ns$). Following formulas determine the LUT values:

$$\begin{cases} x = [0 : \frac{6\pi}{49} : \frac{6\pi}{7}] \\ y = \sin(x) \end{cases}$$

Variable \mathbf{x} forms an array of eight equidistant values between 0 and $\frac{6\pi}{7}$. By using these as input for a sine, \mathbf{y} is filled with eight equidistant values from the first half. The results of these simple math equations are depicted in Table 4. Of course, these values need to be converted to usable bit sequences. By looking at the provided documentation is deduced that the VCTCXO DAC needs a 12-bit signed value as input [18]. The maximum value of a 12-bit signed sequence (011111111111) corresponds to 2047. Row two in Table 4 is transformed by multiplying every value with this maximum and then converting it to a signed sequence. This is illustrated in row three and four of Table 4. The penultimate step of the OQPSK modulation is to delay the odd half sine pulses by half a symbol interval. By using a shift register which can store up to 4 ($\frac{4\text{samples}}{8\text{samples/symbol}} = 0.5$ symbol) sine samples this was realized. Finally, the I and Q sine samples enter a multiplexer to connect them to the DAC. As mentioned in [4], the selection signal of this mux is clocked at two times the sample rate in order to prevent the loss of I and Q samples. More information about how the **pulse shaping** block is connected to its neighbors is depicted in figure 38 of Appendix A.

Table 4: Results of Matlab functions and LUT values

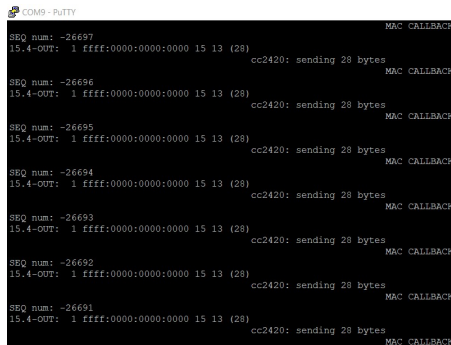
\mathbf{x}	\mathbf{y}	$\mathbf{y * 2047}$	signed LUT values
0	0	0	000000000000
0.3847	0.3753	768	001100000000
0.7694	0.6957	1424	010110010000
1.1541	0.9144	1872	011101010000
1.5387	0.9995	2046	011111111110
1.9234	0.9385	1921	011110000001
2.3081	0.7403	1515	010111101011
2.6928	0.4339	888	001101111000

Chapter 4

Results

4.1 Message Repeater

As stated in 3.2, the architecture is tested by configuring the SDR to receive a 6LoWPAN message emitted by a Zolertia node. Using Putty, a software solution used for determining serial communication over a computer's COM-ports, there can be checked what message was send. Based on Figure 23, there can be concluded that the Zolertia Z1 node sends a 28-bit repetitive message. The bladeRF x115 provides the possibility to cache the retransmitted analog I- and Q-samples by storing them in Comma Separated Value (CSV) files. This is used to determine the quality of the message repeater. Following three situations are simulated: the Zolertia node is deactivated, the Zolertia node is activated and the Zolertia node is activated but a Bluetooth emitter disrupts the send messages. Figures 24 depict the first situation where the node is inactive. Very noisy samples, which have no structure whatsoever, are received and retransmitted. Figure 25 depicts the situation where the node is activated and sending a repetitive message. Both the I- and Q-samples are clearly less random than the previous situation. Of course, some minor interference is inevitable, but the repetitive structure is clearly visible. Finally, Figure 26 depicts the received and retransmitted I- and Q-samples when the Zolertia node is activated but a Bluetooth emitter disturbs the received messages. The samples' amplitude is significantly higher than in previous situations. As a result of this, the repetitive messages are less distinguishable from the rest of the signal. Based on all these results there can be concluded that the message repeater meets its required functionality. It has potential to form a base for implementation of a multi-protocol gateway on the bladeRF x115, but in order to achieve this more signal processing blocks need to be added. These will have to filter unwanted signals send by nodes which are not connected to the heterogeneous network.



```
COM9 - PUTTY
MAC CALLBACK
SEQ num: ~26697
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
SEQ num: ~26698
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
SEQ num: ~26695
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
SEQ num: ~26694
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
SEQ num: ~26693
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
SEQ num: ~26692
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
SEQ num: ~26691
15.4-OUT: 1 ffff:0000:0000:0000 15 13 (28)
cc2420: sending 28 bytes
MAC CALLBACK
```

Figure 23: 6LoWPAN messages sent by the Zolertia Z1 node

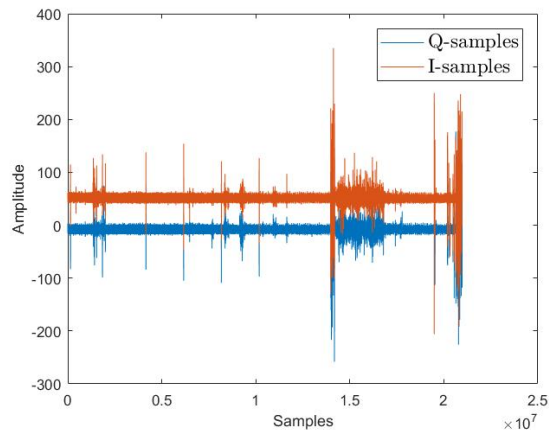


Figure 24: Retransmitted I- and Q-samples when Zolertia is deactivated

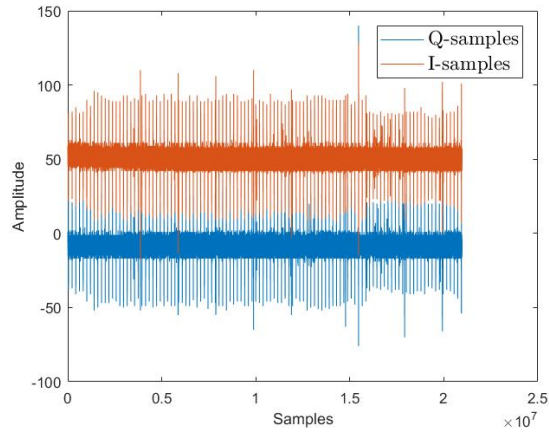


Figure 25: Retransmitted I- and Q-samples when Zolertia is activated

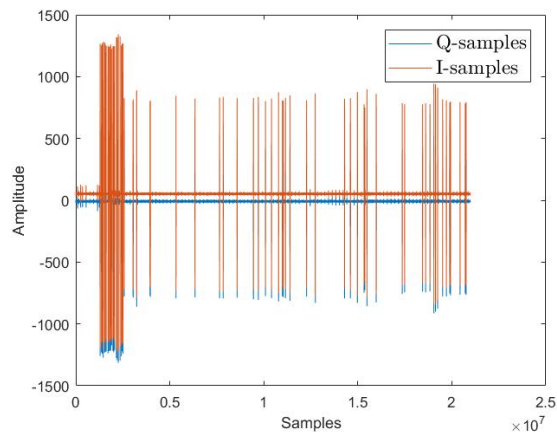


Figure 26: Retransmitted I- and Q-samples when Zolertia is activated, but Bluetooth emitter interferes

4.2 Simulink Simulation Results

To test the performance of the four ZigBee transceiver systems, seven different Signal to Noise Ratio (SNR) values for the Additive White Gaussian Noise (AWGN) signal are applied. This way gradually more noise is introduced to the transmitted information. At the receiving end of the systems, the BER is calculated to determine which design is most suited. BER of a modulation technique means the number of bits corrupted to the total amount of transmitted bits. Each test evaluates 1 million samples send over both an implementation with DSSS and maximum likelihood detection and an implementation without these functionalities. This way the results, summarized in Table 5 and Table 6 are comparable. The names of the ZigBee transceivers are based on the fact if they employ single- or multi-rate and the input/output type (integer or bit). Single-rate ensures that the model's input and output employ the the same port sample time. In this mode the output size of the OQPSK block is an integer multiple of the **samples per symbol** parameter, in this case $4 * 32\text{-bits} = 128\text{-bits}$. Multi-rate processing allows the input and output port to have disparate sampling periods. Its output sample time is equal to the symbol period divided by the **samples per symbol** parameter.

Table 5: BER results for Simulink configurations with DSSS and MLE

SNR (dB)	Single-rate (Integer)	Single-rate (Bit)	Multi-rate (Integer)	Multi-rate (Bit)
0	0	0	0	2.5e-5
-2	4.4e-5	1.8e-5	0.0002	0.0004
-4	0.0014	0.0007	0.0038	0.0047
-8	0.0655	0.0368	0.0902	0.0755
-16	0.3816	0.3321	0.3956	0.3643
-32	0.4913	0.4865	0.4922	0.4887
-64	0.5004	0.5002	0.5009	0.4998

Table 6: BER results for Simulink transceiver configurations without DSSS and MLE

SNR (dB)	Single-rate (Integer)	Single-rate (Bit)	Multi-rate (Integer)	Multi-rate (Bit)
0	0.0336	0.0225	0.0627	0.0823
-2	0.0810	0.0556	0.1073	0.1111
-4	0.1447	0.1029	0.1669	0.1524
-8	0.2744	0.2122	0.2886	0.2479
-16	0.4224	0.3755	0.4273	0.3911
-32	0.4900	0.4797	0.4907	0.4823
-64	0.5001	0.4992	0.5002	0.4993

The results in Table 5 and Table 6 indicate that all four implementations, which use DSSS and MLE, are able to circumvent the added white Gaussian noise fairly well until the SNR values come close to -16 dB. From here on, the systems are less able to suppress the noise and chances they 'translate' received data sequences to wrong symbols increase significantly. The implementations without DSSS and MLE, which only use the OQPSK (de)modulator blocks, have considerably more problems at low SNR values, as illustrated by both Table 6 and Figure 27. From -32 dB to -64 dB all BER values start to stabilize towards 0.5. Although, the MLE block should prevent wrong chip to symbol conversions from happening, there may be multiple PN sequences with the same *hamming distance* to the corrupted 32-bit input signal. The system delays (modulation delay and sample correction delay) of all four configurations with DSSS and MLE are:

- **Single-rate (integer)**: 4 samples,
- **Single-rate (bit)**: 4 samples,
- **Multi-rate (integer)**: 12 samples,
- **Multi-rate (bit)**: 12 samples.

Based on these results and the BER values in Table 5, there is no clear answer on which implementation is most suited. All of the BER values are approximately the same and the difference in system delay is negligible. But knowing that the transition to VHDL requires the system to employ different clock speeds in certain parts of the receiver and transmitter architecture, makes the **Multi-rate (Bit)** configuration stand out. Starting from this implementation, a FPGA-based solution is constructed as described in 3.4.

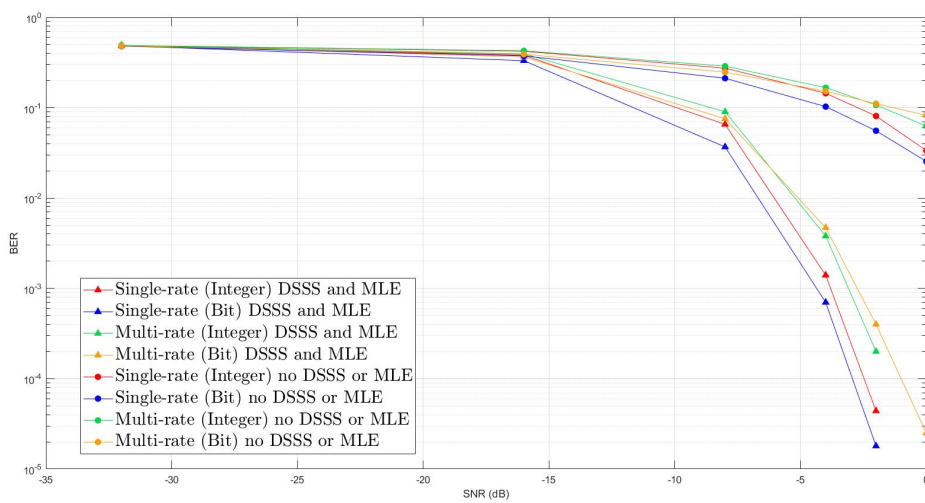


Figure 27: BER values of Simulink transceiver systems

4.3 HDL Simulation Results

4.3.1 Design Summary

Figure 28 depicts the design summary of the entire ZigBee physical layer modulation simulation. The targeted device was a xa6slx4-3csg225 Spartan-6 FPGA, which has a significantly less available resources than the Cyclon IV equipped on the bladeRF x115. This means that the architecture can be further expanded without falling short on available BRAM, registers, phase locked loops, etc.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	385	4800	8%
Number of Slice LUTs	392	2400	16%
Number of fully used LUT-FF pairs	219	558	39%
Number of bonded IOBs	75	132	56%
Number of Block RAM/FIFO	2	12	16%
Number of BUFG/BUFGCTRLs	3	16	18%

Figure 28: Screenshot of design summary of ZigBee physical layer modulation simulation

4.3.2 Simulation

To test the performance of the created ZigBee physical level modulator system, the entire design is simulated using the Xilinx ISE WebPACK. To shorten the simulation time all used clocks are scaled by a factor of 0.01. The situation that is analyzed goes as follows. First, to synchronize all modules in the system, the reset signal is set to high after waiting for sixteen output clock (1 MHz) periods. This causes the system clock to disappear for 1 cycle as it is invoked by a clock divider which uses the output clock and the reset signal as inputs. After waiting for another sixteen output clock cycles, reset is set to low and the system clock will ‘recover’ . Next, a 32-bit sequence (00000001001000110100010101100111) is loaded into the input FIFO by setting the **write input fifo** signal to high for one system clock cycle. Then, another 32-bit sequence (100010011010101110011011101111) is stored by following the same procedure. Figure 29 depicts this first part of the simulation. Both the system clock and output clock are colored red. The input data is colored white.

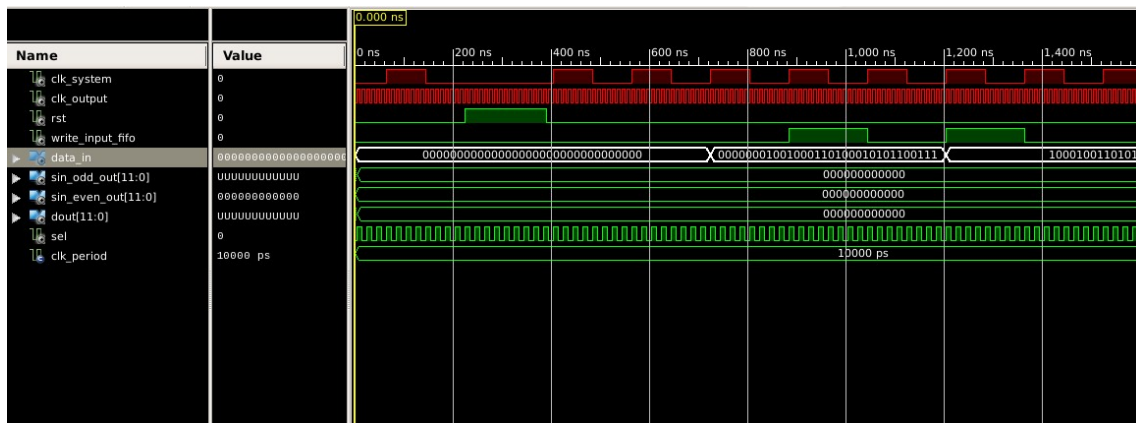


Figure 29: First part of HDL simulation

Once the 32-bit sequences are loaded into the modulator system, the FIFO pulls its **empty** signal down to start the data processing. As described in 3.4, the data first passes the **bit to symbol** module where it is converted to 4-bit symbols. Then, it enters the DSSS block. Here 4-bit symbols

are converted to 32-bit pseudorandom noise chips. Next, the **de-interlace** module separates the even and odd bits into two 16-bit sequences (I- and Q-values), which thereafter enter a **serialize** block to convert the 16-bit parallel data into bitstreams. Finally, these streams are used as input for the **pulse shaping LUT** to output eight half sine samples in each symbol interval. The odd sine samples are delayed by half a symbol interval as the OQPSK standard requires. By multiplexing both the I- and Q-values, they are sent to the DAC at alternating output clock cycles. As illustrated on figure 30, it takes the architecture around 2.24 ns or 14 system clock cycles to produce an output starting from when the first sequence is loaded into the input FIFO. Some of this delay is due to imperfections in the designed modulator, another reason is that all the signal assignments are clocked.

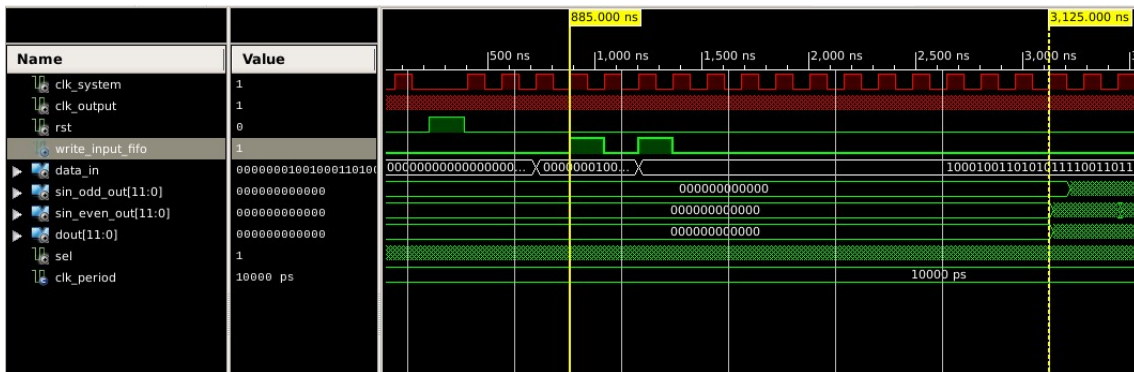


Figure 30: Part of HDL simulation that illustrates clock cycle delay

Figure 31 depicts the entire simulation. It takes the modulator architecture 48.9650 μ s to handle the reset procedure and the two 32-bit sequences. After the data is processed, the architecture emits zero values preventing the radio front end from sending out wrong messages. Of course, as the control signals are all set manually and sometimes skip a clock period, it is possible to reduce the overall simulation time by optimizing the control signal allocation. Figure 32 is zoomed in on a part of the simulation to demonstrate that the modulator architecture actually outputs half sine values. It is clearly visible that the output (illustrated in blue) is either the I-sample when the **sel** signal is '1' or the Q-sample when the **sel** signal equals '0'. This happens at a rate of $62.5 \text{ kHz} * 8 * 2 = 1 \text{ MHz}$ when upscaled by 10. The multiplication by eight is bound to the number of half sine samples in the OQPSK lookup table, while the multiplication by two is to prevent the loss of I- and Q-samples. Figure 33 illustrates the greatest shortcoming of the created ZigBee modulator system. Every time a 4bit-symbol is processed, the system is paused for 0.32 ns or 2 clock cycles. This is due to the **bit to symbol** block which has to respectively read and write the **empty** and **read** signals. Solutions to this problem are provided in the section 5.2.

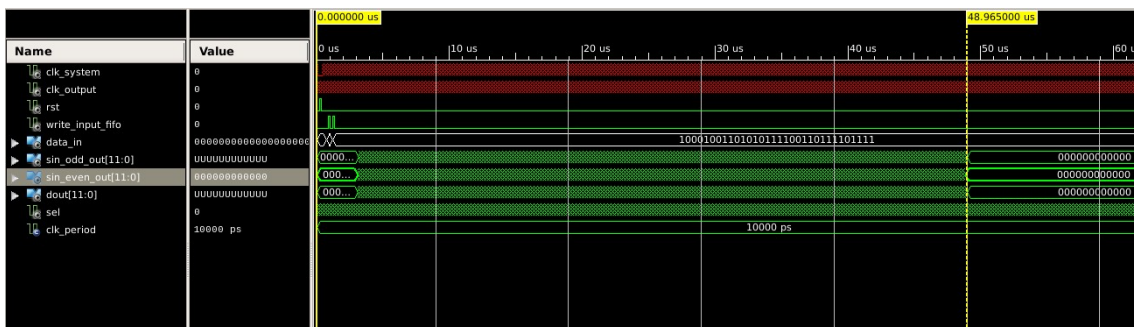


Figure 31: Entire HDL simulation of ZigBee physical layer modulator

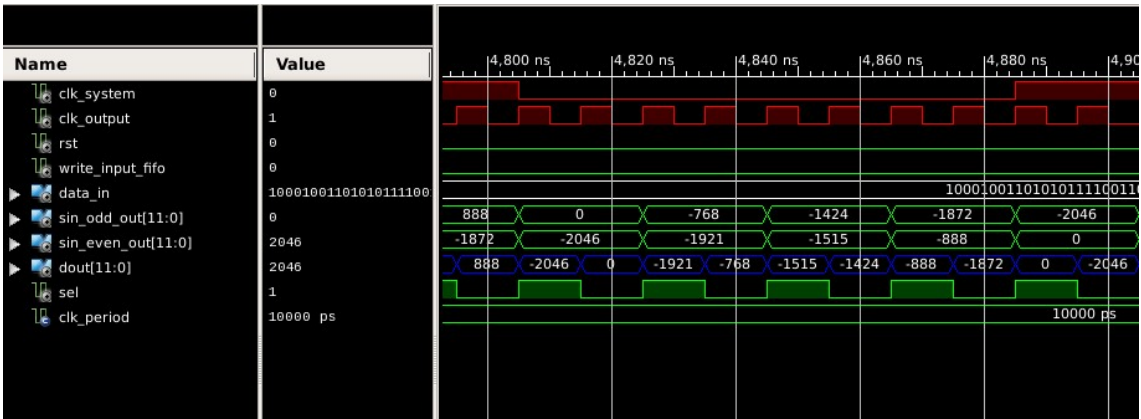


Figure 32: Zoom of HDL simulation

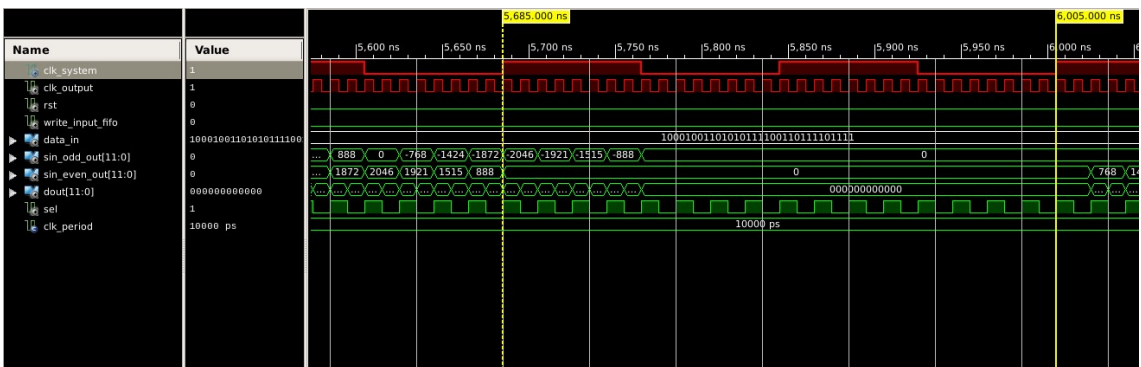


Figure 33: Part of simulation that displays two clock cycle delay introduced by *bit to symbol* block

Chapter 5

Conclusion and Future Work

5.1 Message Repeater and Simulations

The message repeater created in 3.2 is able to tune to a certain frequency, bandwidth, sample rate, etc. by loading a script onto the bladeRF x115 through the Nuand CLI. The SDR then starts receiving I- and Q-samples. These are recombined to 32-bit data sequences and sent to the transmitter side of the architecture. Here, the 32-bit samples pass all the standard modules which will split them back to I- and Q-samples. These are retransmitted by the radio front-end. Results indicate that the message repeater is able to receive and retransmit the messages sent by the 6LoWPAN node despite interference of a Bluetooth transmitter. The architecture will form a good base for a multi-protocol gateway which receives messages from one protocol, applies signal processing to the data and retransmits to a node functioning on another standard. Multiple signal processing blocks will have to be added in order to achieve this goal.

Example systems which could potentially be based on the message repeater are both the Simulink and Xilinx ISE WeBPACK simulations. All four implementations built in Simulink were able to mimic a physical layer ZigBee transceiver. They were tested by comparing the BER when messages were sent over an AWGN channel with or without DSSS and MLE. The configurations with DSSS and MLE functioned significantly better at low SNR values than the standard implementations, but had similar performance at higher values. Based on these BER results, there was no clear answer on which transceiver should form a foundations for the VHDL ZigBee modulator system. Knowing that a HDL implementation would require multiple different clock rates within the same architecture, the **Multi-rate (Bit)** transceiver was chosen.

The results from the Xilinx ISE WeBPACK simulation indicate that a fairly robust ZigBee physical layer transmitter system was created. It is able to transform 32-bit input sequences into 12-bit signed half sine samples. These eventually serve as input for the VCTCXO DAC embedded on the bladeRF x115. Provided some changes to the created architecture, which are recommended in the section 5.2, this simulation can form a base for a fully functioning ZigBee and possibly multi-protocol gateway. All the tests and the literature study executed in this thesis imply that FPGAs will have an important role within the future of both Software Defined Radio and the Internet of Things. They allow to process larger amounts of data within fewer clock cycles, also known as hardware acceleration. The bladeRF will form the ideal tool to explore this, as its cost is reasonable yet its performance is powerful.

5.2 Future Work

As stated in the conclusion, the ZigBee physical layer modulation is far from optimized. The system is on hold for one system clock cycle while the **bit to symbol** block checks whether its accompanying FIFO is empty or not. Another clock cycle is ‘wasted’ on sending a read signal to the FIFO. Now, the **bit to symbol** block tries to serially divide each 32-bit input sequence into eight 4-bit symbols sequentially. A possible solution could be to temporally parallelize the system from the **bit to symbol** to the **De-interlace** block, as depicted in figure 34. This reduces the number of clock cycles to process an input sequence from ten to one. Of course, a block will have to be implemented which serializes the eight DSSS chips, which are fed into the **de-interlace** module. Another possible solution, which maintains the serialized nature of the architecture, is to start looking whether the FIFO is empty when sending out the penultimate 4-bit symbol. This way those two cycles will not matter as still two symbols have to be sent out. Modules

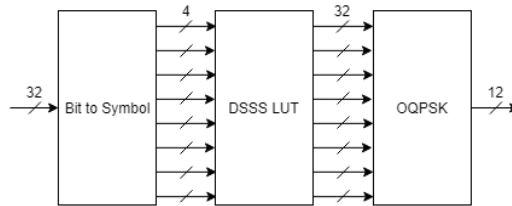


Figure 34: Possible solution to solve delays in modulator system

that can possibly be added to improve the system’s performance are: cyclic redundancy check, encryption blocks, etc. To create a fully functioning multi-protocol gateway, a ZigBee physical layer demodulation system will need to be constructed. Originally some VHDL files have been designed to accomplish this, such as a maximum likelihood estimation and DSSS despreading. But they were never implemented in a simulation due to lack of time. The promising thing about the ZigBee modulator and demodulator architecture is that it, just like many other IoT protocols (6LoWPAN, MiWi and Bluetooth Low Energy), is based on IEEE 802.15.4. So these blocks can be reused when introducing other standards to the bladeRF x115. But for other IoT communication protocols such as WiFi, which relies on IEEE 802.11, or Z-Wave, which uses an entirely different modulation technique, new HDL files will need to be constructed in order to implement them. Concrete, the next steps to achieve a multi-protocol gateway on the bladeRF x115 are:

- finishing the ZigBee physical layer system by modeling a CSMA/CA module to optimize data exchange and implement several block to (de)construct Physical layer Service Data Units (PSDU);
- implementing another IoT protocol, preferably one which also employs the IEEE 802.15.4 standard, otherwise the written HDL files can not be reused. In case another protocol was chosen, an entirely new physical layer system has to be created;
- configuring the bladeRF x115 to scan the radio spectrum for multiple standards at the same time and hereby deciding how to process the received data units;
- building a test network to prove the functionality of the heterogeneous IoT network.

Bibliography

- [1] Koubaa A., Alves M., and Tovar E. *IEEE 802.15.4: a Federating Communication Protocol for TimeSensitive Wireless Sensor Networks*. Tech. rep. Rua Dr. António Bernardino de Almeida, 431 4200-072 Porto, Portugal: Polytechnic institute of Porto (ISEP-IPP), 2006.
- [2] Rafidah A. et al. “Implementation of IEEE 802.15.4-Based OQPSK-Pulse-Shaping Block on FPGA”. In: *IEEE* (2011), pp. 459–464.
- [3] Lalge A.M., Karpe M.S., and Bhandari S.U. “Software Defined Radio Principles and Platforms”. In: *International Journal of Advanced Computer Research* 3.3 (Sept. 2013), pp. 133–138.
- [4] Glod B. *FPGA Development*. <https://github.com/Nuand/bladeRF/wiki/FPGA-Development>. 2018.
- [5] Wang X. Baek I. H. *EE209 AS Project: Investigation on ”Design Transceiver for IEEE 802.15.4 using ZigBee Technology and Matlab/Simulink*. Web page. 2017. URL: <https://www.semanticscholar.org/paper/EE209AS-Project%3A-Investigation-on-%E2%80%99Design-for-IEEE-Baek-Samueli/27b6e635cbdb678f9e50e4f142b3f4516eacfc3d?tab=abstract>.
- [6] *bladeRF*. Nuand. 720 East Ave Suite 201 Rochester, NY 14607.
- [7] *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. Sept. 2017. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html?CAMPAIGN=VNI+2016&COUNTRY_SITE=us&POSITION=Press+Release&REFERRING_SITE=Cisco+page&CREATIVE=PR+to+VNI+White+Paper&_ga=1.14.
- [8] Gee K. E. et al. “A review of 6LoWPAN routing protocols”. In: *Proceeding of the Asia-Pacific Advanced Network*. Vol. 30. Apan. 2010, pp. 71–81.
- [9] Grayver E. “Implementing Software Defined Radio”. In: Springer, 2013. Chap. 2, pp. 5–9.
- [10] Kavya G. and Mani V. V. “Simulink Model for Zigbee Transceiver Using OQPSK Modulation under Fading Channels”. In: *IEEE* (2015), pp. 220–224.
- [11] *Gartner Says 8.4 Billion Connected ”Things” Will Be in Use in 2017, Up 31 Percent From 2016*. Feb. 2017. URL: <https://www.gartner.com/newsroom/id/3598917> (visited on 11/29/2017).
- [12] *How IoT is transforming healthcare – 5 new technologies that help make hospitals more human*. Apr. 2017. URL: <https://blog.dimensiondata.com/2017/04/iot-transforming-healthcare-5-new-technologies-help-make-hospitals-human/> (visited on 11/30/2017).
- [13] *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)*. Nov. 2016. URL: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>.

- [14] Morgan J. *A simple explanation of 'The Internet Of Things'*. May 2014. URL: <https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#49fee85c1d09>.
- [15] Olsson J. *6LoWPAN demystified*. Tech. rep. Texas Instruments, Oct. 2014.
- [16] Gutierrez J.A. et al. "Applying wireless sensor networks in industrial plant energy evaluation and planning systems". In: *IEEE* (2006).
- [17] Ravikanth K. "Design of ZigBee transceiver for ieee 802.15.4 using Matlab/Simulink". MA thesis. National Institute of technology Rourkela, Odisha, 2011.
- [18] *LMS6002D*. 1.1.0. Lime microsystems. Dec. 2012.
- [19] Fildes N. *Meet the "connected cow"*. Oct. 2017. URL: <https://www.ft.com/content/2db7e742-7204-11e7-93ff-99f383b09ff9> (visited on 11/30/2017).
- [20] Gravina R. et al. "Integration, Interconnection, and Interoperability of IoT Systems". In: Springer, 2018. Chap. 3, pp. 211–213.
- [21] Machado-Fernández J. R. "Software Defined Radio: Basic Principles and Applications". In: *Revista Facultad de Ingeniería (Fac. Ing.)*, Vol. 24 (2015), pp. 79–96.
- [22] Li K. Shi G. "Signal Interference in WiFi and ZigBee Networks". In: vol. 7. Springer, 2017. Chap. 2, pp. 9–27.
- [23] "Software defined radios - overview and hardware (1)". In: *Software defined radios*. Vol. Vol. 182. Nov. 2004, pp. 58–61.
- [24] Hollis T. and Weir R. "The Theory of Digital Down Conversion". In: *Hunt Engineering* Vol 1.2 (June 2003), pp. 1–6.
- [25] Akkerman W. *De broodrooster van Griffin Technology werkt met bluetooth*. Jan. 2017. URL: <https://www.smarthomemagazine.nl/2017/01/broodrooster-griffin-technology-werkt-bluetooth/> (visited on 11/30/2017).

Appendices

Appendix A: Diagrams of VHDL modules	35
--	----

Appendix A: Diagrams of VHDL modules

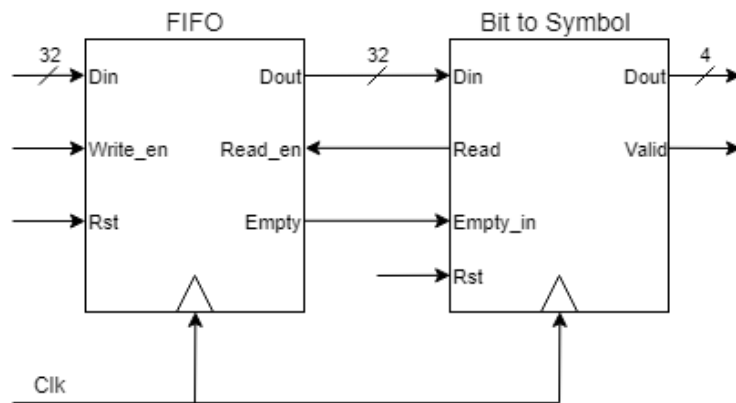


Figure 35: Diagram of *bit to symbol* block and its neighbors

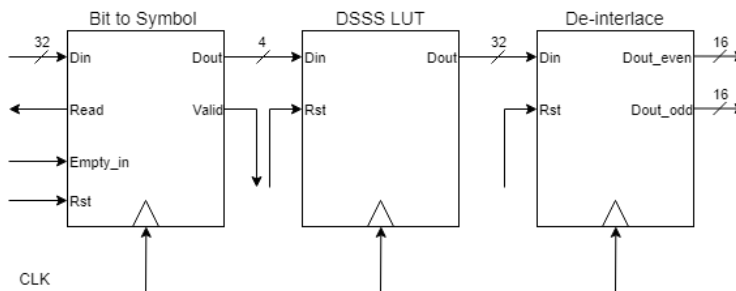


Figure 36: Diagram of *DSSS LUT* block and its neighbors

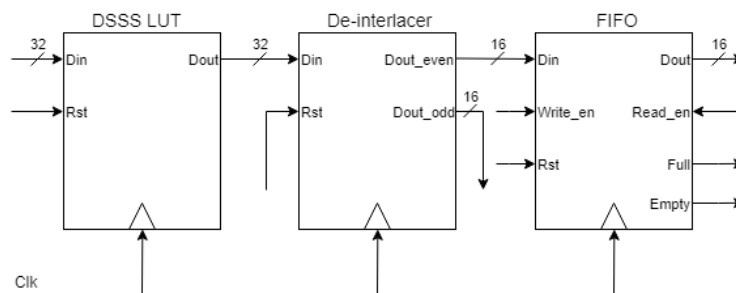


Figure 37: Diagram of *De-interlace* block and its neighbors

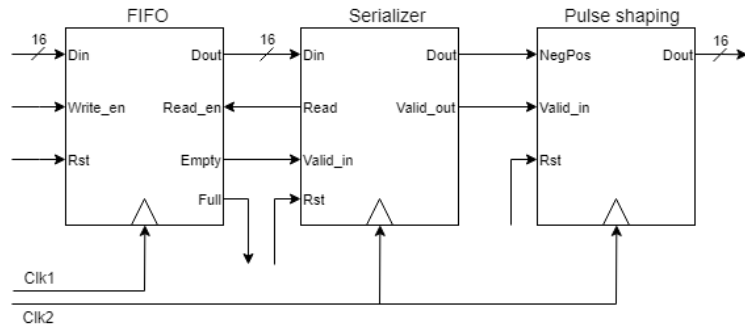


Figure 38: Diagram of *serializer* block and its neighbors

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Towards Software-Defined Radio on Configurable Hardware

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2018**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Bertrands, Karel

Datum: **4/06/2018**