Wireless System on Virtual Platform Evaluation

Joel Gustafsson
elt12jg1@student.lu.se

Tobias Mähl
har11tma@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Stefan Höst (EIT), Magnus Midholt (ARM)

Examiner: Maria Kihl

September 15, 2017

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Wireless System on Virtual Platform Evaluation

The most common way to develop System-on-Chip's (SoC) today is to utilize hardware design on a Field programmable gate array (FPGA). By utilizing the hardware on an FPGA, the opportunity to verify the hardware and start software design before implementing the final solution on silicon is possible. However, using an FPGA as a reference model for a final design gives a slightly imprecise estimation regarding required hardware such as memory and bus sizes. Trying to counteract this weakness, specific Electronic System-Level (ESL) Design tools has been developed in order to simplify production and increase capabilities to analyze and optimize during the developing phase. ARM SoC Designer is such a tool which provides a virtual environment for simulation of integrated SoC's to simulate whole systems at a cycle accurate level.

This Master's Thesis intend to evaluate ARM SoC Designer aspects and validity as an alternative to an FPGA when implementing and verifying a larger SoC system. To provide a thorough assessment of ARM SoC Designer, a cycle accurate model of the digital signal processing system of a general NB-IoT system was to be built and verified. Results shown, reveal that an implementation in ARM SoC Designer can be a valid representation of a preexisting wireless NB-IoT system currently developed on an FPGA board. Also that SoC Designer adds some very useful functionalities to traditional SoC development techniques but not without some significant flaws.

Acknowledgements

Our thesis work were performed at ARM Sweden AB - Wireless business unit, and we want to extend our gratitude and appreciation to their employees for providing help and feedback throughout our thesis. Especially we would like to thank our supervisor Magnus Midholt for valuable insight into pursuing and carrying out projects of this kind. We particularly also would like to thank Patrik Elfborg and Marcel Tovar for their valuable help and advice throughout the entire process. In addition we would like to thank our academic supervisor Stefan Höst for guiding us through this project.

Popular Science Summary

With rising complexity and highly competitive market of integrated circuits, new techniques to develop embedded systems are always needed in order to secure the quality and to streamline the production of circuits. The tool ARM SoC Designer tries to accomplish this through adjusting traditional System-on-Chip development by adding debugging and troubleshooting capabilities not present in current design techniques. The question is, will the use of SoC Designer help?

A System-on-Chip (SoC) is an especially designed chip that combines required electronic circuits of different components onto a single integrated chip, usually not bigger than a thumbnail. By directly building a chip containing all desired components instead of assemble multiple chips together makes it possible to manufacture the chips as small as possible and at the same time make them faster and more energy efficient than before. But the tininess of a SoC and the numerous amount components it is going to contain makes it also to a very complex system with a long and expensive developing process, where faults and errors are very hard to find and eliminate.

The procedure of designing and developing SoC have been almost the same for a very long time. But with the rising demands on chips with more capacity in less area puts a lot of pressure on the manufacturers in the industry. Therefore manufacturers and developers have been starting to look for better design tools to increase the possibility to analyze and optimize during development. One of these tools are ARM SoC Designer which provides a virtual environment for simulation of integrated SoC's with the promise of great debugging and verifying capabilities. ARM SoC Designer are able to do this by allowing the user to simulate the process step by step and to easily modify system parameters.

In order to evaluate ARM SoC Designer, a virtual model of an existing wireless system was built, tested and verified in the virtual environment. Different tests to analyze performance, accuracy, and debug- and troubleshooting capabilities were constructed and performed. With results showing great promise in categories accuracy and debugging, but left some things to be desired with performance and usability. This thesis can confirm that ARM SoC Designer does deliver an accurate representation of a SoC and that the verification capabilities are extremely helpful during implementation and testing. But, at the current performance in

combination with a less good user experience, a steep learning curve and scarce documentation makes it difficult to recommend during a daily development basis. However, ARM SoC Designer shows a lot of promise if the usability can be enhanced slightly and performance be improved upon to such an extent that downtime of simulations won't be the majority of time consumed during usage.

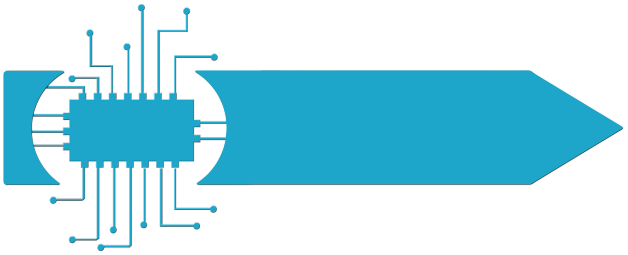


Table of Contents

List of Figures

List of Tables

Abbreviations

3GPP 3rd Generation Partnership Project

CPU Central Processing Unit

DMA Direct Memory Access

ESL Electronic System Level

FIFO First In, First Out

FPGA Field Programmable Gate Array

I/O Input/Output

IoT Internet of Things

IP Intellectual Property

LTE Long Term Evolution

MTC Machine Type Communication

NB-IoT Narrowband Internet of Things

NPSS Narrowband Primary Synchronization Signal

OSI Open Systems Interconnection

RAM Random Access Memory

RF Radio Frequency

RTOS Real-Time Operating System

RX Receiver

SoC System-on-Chip

TX Transmitter

UART Universal Asynchronous Receiver/Transmitter

Chapter 1
Introduction

Today the most common way to develop System-on-Chip's (SoC) is to utilize hardware design on a Field programmable gate array (FPGA) in order to verify and to start software design before implementing the final solution on silicon. However, using an FPGA as a reference model for the final design gives a slightly imprecise estimation regarding required hardware such as memory and bus sizes. By integrating a virtual platform as a tool to develop SoC's, an entire system will be able to run at a cycle accurate level. This will make it possible to evaluate a system and its behavior as if it already was integrated on silicon. In the end this will lead to more precise hardware design where the knowledge of the system will guarantee the designers not to under- or over design the system. A more precise SoC design will shorten the time to market and reduce the cost of producing.

When this project was first initialized, it was the second of its kind at ARM Sweden. A couple of months earlier a similar Master's Thesis project had been addressed at ARM's wireless business unit, with the goal to investigate and evaluate the possibilities to use a virtual platform as a developing and verification tool. The purpose of the thesis was to build up a small system which could prove the possibility of changing the way to develop System-on-Chip's. The outcome of the project gave the managers at ARM's wireless business unit a lot of confidence in the use of a virtual platform as a development tool. As a result of the the previous findings a new Master's Thesis project was proposed in order to investigate the tool even further. This time with the ambition and expectation of building up an entire design of a wireless system in the virtual environment to evaluate the tool.

1.1 Purpose and Goals

The aim of this Master's Thesis project is to build up a cycle accurate model of the digital signal processing system in ARM SoC Designer for simulating ARM's current NB-IoT system. The purpose is to evaluate system aspects such as performance, module interfacing and time saving capabilities compared to using an FPGA.

The Master's Thesis will investigate the following questions

- Investigate the possibility of implementing a complete system into SoC Designer and if it can be used to speed up the development process.
- Evaluate platform aspects such as turnaround time, flexibility time and debug capabilities.
- Evaluate the possibility of streaming test data instead of using test vectors.

To more easily answer the investigatory questions and to gather the needed data. Subquestions were introduced and formulated to what were required investigating to conclude this Master's Thesis. These were summarized to:

- How close to the preexisting system is the representation in SoC Designer?
- Is it possible to perform a band scan?
- How is performance and how much is it affected by the size of the system?
- What is the time consumption for different parts of the system?
- What specific debugging capabilities does SoC Designer provide?
- Is SoC Designer valid to use in real development scenarios?

To investigate how close of an representation the implemented model in SoC Designer is to the actual system and if it is possible to perform a band scan would give an indication of how valid the results given by the simulation in SoC Designer actually is. To validate, radio test vectors will be sent through the wrapped hardware and the result will be compared to the result of a Matlab representation of the hardware. The same approach will be performed on the software to find if it is possible to perform a band scan in the simulation. The energy peaks found by the simulation will be compared to the energy peaks found when studying graphs of the energy level sent into the Matlab model.

Also data regarding the performance of the implemented system would need to be gathered. Is it fast enough and how much of an impact does the major hardware blocks have on the performance? This also relates to time consumption, turnaround time and how much time is needed to execute certain parts? Also how much time is consumed during a scenario where a change is made, system recompiled, simulation restarted and executed to the same point as before are of interest. All major factors to conclude if SoC Designer is viable during a real development scenario.

SoC Designers debugging capabilities will be judged primarily subjectively during implementation, debugging and troubleshooting of how much of a benefit it provides during those stages of the project. Lastly, arguments regarding why or why not SoC Designer could be used within a real development scenario will be presented.

1.2 Limitations

To limit this thesis, the focus of the project has been to build up only the required system of a general NB-IoT design in order to perform the band scan procedure of the connection phase. All functionality, excluding the required parts to perform a band scan has therefore been left as unimplemented components in hardware and as stubbed methods in software.

Since the goal was to investigate if a system such as the wireless NB-IoT system could be developed in the virtual environment and performing a virtual band scan as a proof of concept, only the correctness were of importance. To save time when evaluating the virtual system, a Matlab reference model has therefore been used instead of building up a similar system on a FPGA board.

As mentioned earlier in this chapter, one part of this thesis was to evaluate the possibilities to stream data into the system instead of using test vectors. But to limit the thesis, this was left outside the scope of this project, since it involved a bigger workload than expected.

1.3 Contributions

The thesis was evenly distributed during the course of the project and both Authors have been equally involved in most parts. Thanks to the diverseness of knowledge gained through different Master specializations, the Authors have learned from each other in situations where one had better knowledge than the other.

In the beginning of the project, both Authors worked together to learn how the virtual platform ARM SoC Designer worked and how the system implementation should be done in order to be compatible with the platform. Later, implementation into SoC Designer were carried out in parallel to streamline development. Some parts of the implementations were naturally divided among the Authors, e.g. Joel worked more with the hardware solutions and Tobias with the systems software.

When writing this thesis, the Authors worked on different sections in parallel but both were involved on every part.

1.4 Thesis Outline

The remainder of this thesis will be structured as follows:

Chapter 2: Background Theory - Gives a brief introduction to theory that is needed to understand the thesis. Procedures such as Band Scan will be presented and expressions such as NB-IoT and NPSS will be explained.

Chapter 3: System - Gives an overall presentation of how the system design of both hardware and software could look like.

Chapter 4: Method - Describes the methodology of the the project and how it was carried out.

Chapter 5: Results - Presents the result of the thesis.

Chapter 6: Discussion - Covers a discussion with respect to research questions and general thoughts.

Chapter 7: Conclusion - Presents the conclusion of the outcome of this Master's Thesis.

Chapter 8: Future Works - Explores possibilities and future work with ARM SoC Designer.

Chapter 2
Background Theory

This chapter will present the relevant background theory and key concepts needed to understand this Master's Thesis.

2.1 Internet of Things

Internet of Things (IoT) is nowdays a trendy expression to describe devices equipped with processors and sensors which in some way is connected to a network in order to collect and exchange data to surrounding devices. The purpose of IoT is to simplify peoples lives by providing data from devices at users disposal. In order to do this, a smart and autonomous system with connected "*things*" need to be developed. One of the main challenges with this is to build devices with very scarce resources in order to make them cheap to produce but also reliable at a minimal footprint.

Cellular IoT is an exciting technology that will have a huge impact on the world in many aspects. The technology has been available for many years using GSM communication. But this is about to change with the new upcoming cellular standards such as Narrowband Internet of Things (NB-IoT), which is a new radio interface optimized for machine type communication (MTC). From the recent forecast there will be 1.5 mobile devices per capita by 2021, which equals around 11.6 billion devices connected through cellular protocols all over the world [3]. With the increasing number of connected devices one of the technologies biggest challenges at the moment is the demand on making the communication cheaper and more power efficient [4].

2.2 Narrowband Internet of Things

With the recent standardization of the new Narrowband radio technology for Internet of Things (NB-IoT), new possibilities to integrate and connect new devices with wide-area coverage can be discovered. The new standardization for NB-IoT is a subpart belonging to the Long Term Evolution (LTE) and was completed with release 13, June 2016, by the 3rd Generation Partnership Project (3GPP) [5]. 3GPP is a collaboration between telecommunication associations in order to make an international standard within the field of Radio, Core Network and Service

Architecture [6], which involves all specifications for GMS, LTE and the emerging 5G technology [7]. The standardization contained the following goals for NB-IoT:

- Long battery life - Over 10 years battery life.
- Low device cost - Under 5 USD per module
- Low deployment cost - Under 1 USD per module
- Extended coverage - 20dB better than GPRS
- Support for an immense number of devices - 40 devices per household

In LTE the frequency units can be described in number of physical resource blocks allocated to a user. For instance, the smallest frequency unit in LTE, 1.4MHz, can be described as 6 resource blocks wide and the highest, 20MHz, can be described as 100 blocks wide. One resource block is 180kHz wide and is the smallest unit allowed to be allocated. In NB-IoT the key feature is to use a very narrow bandwidth to transfer data within. The bandwidth for a NB-IoT device is only 180kHz wide, which implies that communication over NB-IoT only allocates one resource block [8]. As a result, these small bands can be deployed either in the LTE guard band, or as standalone band, or even within an existing LTE carrier as seen in Figure 2.1. NB-IoT also allows devices to send small amount of data in parallel [9].

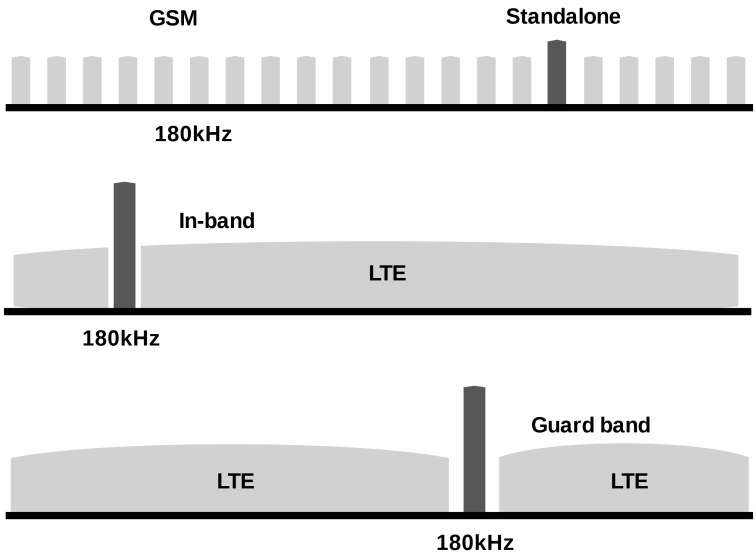


Figure 2.1: Spectrum usage deployment options for NB-IoT.

To achieve all requirements for NB-IoT, a finite and scarce spectrum of resources has to be applied as efficient as possible. Focus on optimization and innovation needs to be considered in each step of the development to achieve this. In contrast to LTE, the goal of NB-IoT is not to transfer data with a high data

rate, instead an NB-IoT device will most of the time be in idle and only for a couple of times a day, week or even year wake up, connect to the network and exchange a small amount of data before it returns to idle mode. By taking this into account, to save battery usage, the initialization phase is of great importance and an essential part to meet the 3GPP requirements of NB-IoT.

2.2.1 Where to use NB-IoT

For a NB-IoT device the rate of data transfer will be very low, making this kind of communication perfect for data exchange where transfer rate, latency and reliability is not the most important features. The main target for this technology is simple sensors and actuators that require low mobility and low levels of data transfer in areas with limited coverage or power supply. A good example of where to use this technology is in the garbage industry. Instead of collecting and emptying the garbage cans once a week, a simple sensor deployed on every can with the NB-IoT technology could be used to indicate to the garbage company when the best time to collect is. The mobility, extended coverage and long battery life together with a low device cost makes this kind of technology perfect for this kind of industry. Since the technology can be deployed with almost anything that have some kind of sensor, hence it can be deployed almost everywhere. The possibilities for this technology is almost infinite if all requirements for IoT communication can be fulfilled.

2.3 Data communication

In order for two persons to communicate and understand each other they need to speak the same language. In data communication it is almost the same principle. Two devices that want to communicate need to follow specific rules and protocols in order to understand each other. In telecommunication and computer systems these rules and protocols are categorized by the open systems interconnection model, also called the OSI-model which defines how data communications should take place between connected systems. Similar functions and processes are grouped and standardized by categorizing them into abstract layers. The layers then provide services to the ones under and above their existing layer [10]. The protocols in the OSI layered model determines:

- How devices should communicate.
- When devices should send and not send data.
- The data flow rate between devices.
- How data should be transmitted and received so that only the intended recipient gets the transmission.
- How the physical communication is organized and connected.

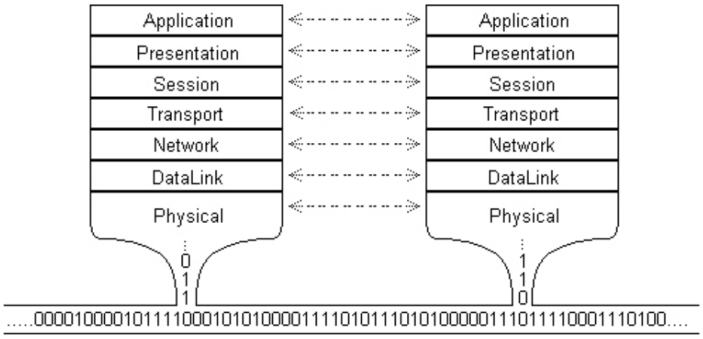


Figure 2.2: OSI seven layer protocol model

Layer 7 - Application Layer provides an interface between the software running on the device and the network protocol.

Layer 6 - Presentation Layer responsibility is to define the structure in which the network devices handle compression and encryption.

Layer 5 - Session Layer administer how communication sessions is established, maintained and terminated.

Layer 4 - Transport Layer ensures messages are being delivered and handles the fragmentation/reassembly of messages.

Layer 3 - Network Layer is responsible for the transfer of data packages between devices on the network.

Layer 2 - Data-Link Layer defines the interface between network and application. Also responsible for the communication between network nodes.

Layer 1 - Physical Layer handles the transmission of bits between network nodes.

With every layer having its own unique features, the process of sending data usually starts at the application layer of the sending host. The data is sent through the host stack to the physical layer where it is transmitted via the network to the target host. At the recipient, the data is received at the physical layer and passed up through the stack to the application layer [10]. In this Master's Thesis the main focus has been on the physical layer, hence, further discussions in this thesis will mainly refer to the design in the physical layer of the OSI model.

2.4 NB-IoT Connection Process

As stated earlier in this chapter the connection process of NB-IoT devices are of great importance since it is one of the most critical stages regarding a units power consumption. For a device to be considered connected it has to go through several initialization steps which can be more or less energy demanding. Once the device has established a connection to a radio access point, the radio access point will take over and connect the device to the telecommunication network.

2.4.1 Connecting Procedure

When an NB-IoT device is first switched on it needs to know which frequency the radio has to be set at, in order to synchronize with the base station. To accomplish this the device needs to perform a band scan, a procedure which scans the frequency bands in order to search for the most suitable network to communicate over. A band scan is completed once all frequency bands have been scanned. Once the band scan and the synchronization towards the base station is done, the device will call for a Cell Search in order to find and access to the most suitable telecommunications cell as seen in Figure 2.3. As the Cell Search procedure is considered to be outside the scope of this Master's Thesis, only the band scan procedure will be explained further.

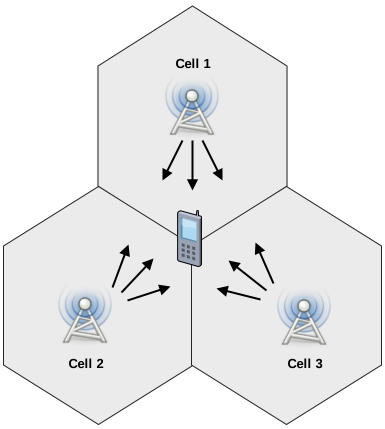


Figure 2.3: Illustration of telecommunication cells

2.4.2 Band Scan

A band scan is the procedure where a device scans the entire frequency band in order to find the most suitable frequency to communicate over. As an initial part of the band scan, the device scans and collects data within a specific band in order to find the primary synchronization signal (NPSS). The NPSS is a known signal which always should appear in subframe number 5 and repeatedly transmitted every 10 ms [11].

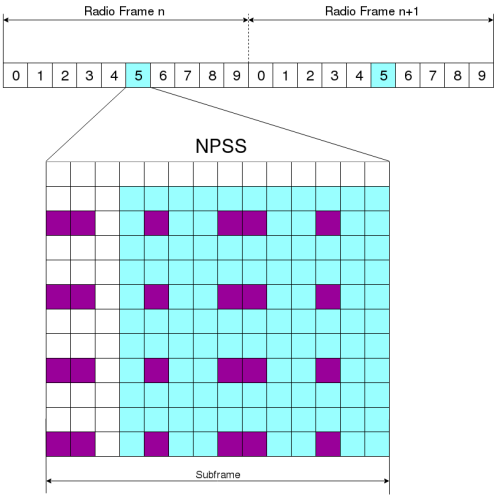


Figure 2.4: Illustration over 20ms of NB-IoT data where the NPSS is repeated every 10ms and located in subframe number 5.

As shown in Figure 2.4, subframe number 5 consists of 14 symbols and 12 subcarriers and in this subframe the NPSS is transmitted on subcarrier 0 to 10 in the symbols 3 to 13, which generates a 11x11 matrix representing the NPSS. Before the signal is transmitted from the base station it is transformed into the time domain and then transfered. Since the NPSS is repeated every 10ms, thus only 10ms of data needs to be collected in order to find if the specific frequency containing the signal.

The procedure aims to sort out at what frequencies the NPSS is transmitted on and making estimations to find out which of these frequencies to be the strongest. When the band with the most suitable signal is found, the device is synchronized against that signal in time and the band scan is considered to be done.

To do this, the radio chooses a frequency and starts to sample data. The data is then processed through the digital signal processing system and analyzed. If adequate information are found within the signal, the frequency and data containing the peak will be placed in memory and new data will be collected for processing. As seen in Figure 2.5, the process start by scanning the selected frequency and will then continue until the entire frequency band has been scanned. Once the entire frequency band has been scanned and the outcome is satisfactory, the analyzed result will be returned and the band scan will be considered complete. In Figure 2.6 two different frequencies are scanned and processed and as can be seen only Figure 2.6b contains a distinct peak somewhere within the scanned time interval, which implies that the NPSS is transmitted over this frequency. From the information of where in time the peak is detected and the knowledge of that the NPSS is transmitted in subframe number 5, the device can make estimation of how it should be calibrated in order to synchronize to the base station, as the next step of the connection procedure.

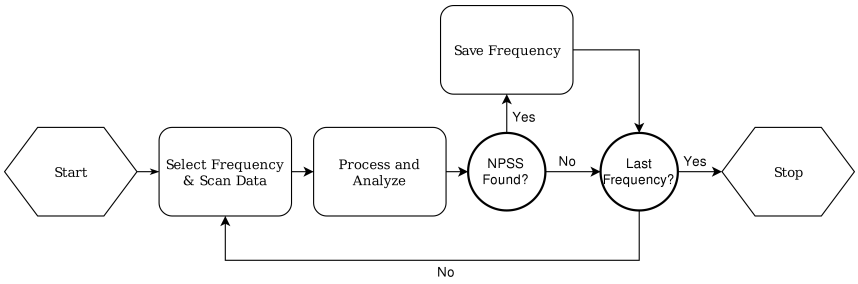
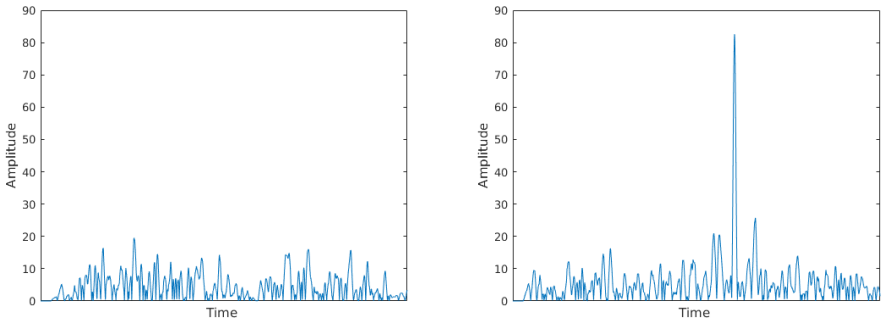


Figure 2.5: Band scan procedure

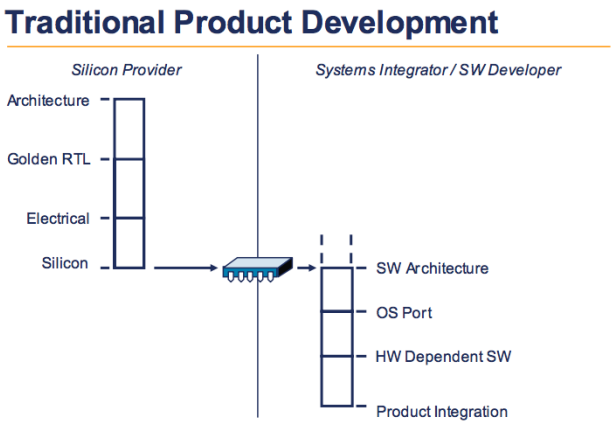


(a) NPSS not found **(b)** NPSS found

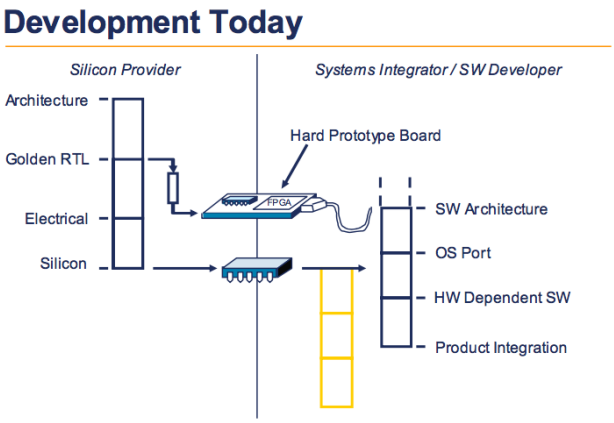
Figure 2.6: Two plots of processed data

2.5 Direct Memory Access

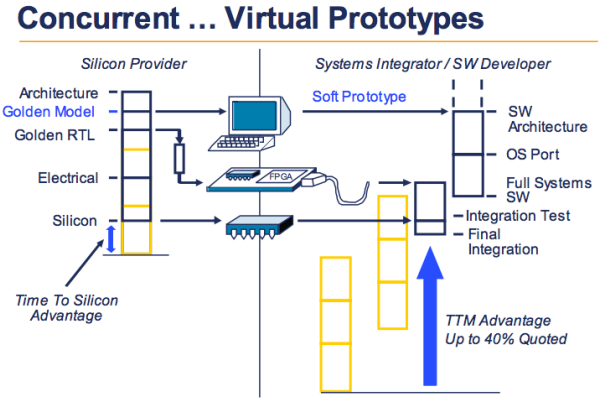
In a system such as the one in a NB-IoT device, the CPU must be able to handle large quantities of in- and outgoing data during a short amount of time. For any CPU to handle this kind of data on their own is a heavily power demanding process which also will slow down the computational speed of the CPU. One way to get around this power consuming process is to use a Direct Memory Access (DMA) controller to control the data flow in the system. A DMA controller is a device programmed to transfer data on behalf of the CPU. A DMA have direct access to the memory and can therefore transfer data from one place in memory to another, or from a I/O device to memory and vice versa. Usually a DMA controller have several programmable channels, which makes it possible to transfer sequences of data in parallel. When a CPU needs to perform a data transfer it sends a request signal to the DMA controller with information about the source- and destination address and the amount of data that is going to be sent. After the DMA have granted the signal, the DMA transfers the requested data while the CPU either goes back to idle mode to save power, or continue to perform other tasks [12].



(a) Traditional development



(b) Development with FPGA board



(c) Development on virtual platform

Figure 2.7: Different SoC development techniques [1]

2.6 ARM SoC Designer

Systems-on-Chip's are complex systems which requires particular design methodologies compared to traditional hardware architectures. This complexity increase the need for procedures to test and verify the hardware and software concurrently during the development process. [13]. Traditionally SoC development has been carried out in a two step process, where first hardware is designed and developed before the software design can take place as seen in Figure 2.7a. Different strides to get around this bottleneck has been made and today the technology most widely used by the industry is to utilize a hardware design on an FPGA as seen in Figure 2.7b in order to verify the hardware and to be able to start software design before implementing the final solution on silicon. However, using an FPGA as a reference model for the final design often leads to a larger implemented silicon area and power consumption not equal to the one estimated on the FPGA. This due to the imprecise estimation on the FPGA always leads to developers adding a margin of error onto the design. To take SoC development one step further, specific Electronic System-Level (ESL) Design tools has been developed in order to simplify the production and to increase the possibilities to analyze and optimize during the developing phase. The use of ESL makes it possible to utilize the SoC virtually in order to start implementing software in parallel. From an industry perspective the new methodology will make it possible to design hardware and software fully in parallel as seen in Figure 2.7c and at the same time give the opportunity to debug and verify whole systems at a cycle accurate level.

One of the tools aiming to solve this bottleneck is ARM SoC Designer, which provides a virtual environment for simulation of integrated SoC's with multiple cores, peripherals and memories using C++. From the SoC Designer User Guide manual it can be read that with a cycle-based scheduler and a transaction-based component interface SoC Designer are able to simulate whole systems at a cycle accurate level at very high speeds. The program can be used for standalone model simulations, integrated hardware simulations or as a hardware/software co-simulation tool. As stated in the User Guide, some of the key features are [2]:

- Controlling the simulation at a cycle accurate level
- Viewing the system structure and hierarchy
- Viewing and modifying the parameters of a system
- Viewing registers and memories through the Cycle Accurate Debug Interface
- Setting breakpoints on registers, memories, disassembly, transactions, and signals
- Monitoring transactions and signals

These features gives the ability to speed up the system integration time, reduce the risks by validating hardware with actual system software, dismiss physical hardware prototypes availability as a bottleneck and accelerate the overall process of testing and debugging a system.

ARM SoC Designer is divided into two separate parts, the SoC Designer Canvas and the SoC Designer Simulator. The different layouts from the two parts can be seen in Figure 2.8 and 2.9, respectively.

The canvas is a drag and drop environment where the hardware system is built up with block components connected to each other. An entire hardware system can be built up using already defined components from the component library or by adding own designed components. If using components not included in the standard library the components must first be converted to the predefined SoC Designer type before it can be added to the canvas. The methodology of this procedure will be more extensively explained in section 4.2.1.

The SoC Designer Simulator is as the name may reveal, the simulator in SoC Designer in order to test and validate the performance of the HW design together with desired software. The simulator gives the opportunity to perform trade-off analysis of the architecture to find if any bottlenecks in the system requires a redesign. The simulator also offers a debugging stage, which adds the possibility to set breakpoints and watchpoints in the system, but also allows the user to follow the signals with cycle accurate precision. From a software point of view, the simulator gives the opportunity to debug the software at a cycle accurate level and by enabling processor profiling each step of the executed software can be followed as shown in Figure 2.10 [1].

2.7 Real-Time Operating Systems

In every operational system there is some system in the background controlling it. This is called the Operating system (OS) and is responsible for managing the processors hardware resources and hosting applications. A real-time operation system perform these things but is also designed to run applications with a very precise timing and at a high level of reliability. Real-time systems are common to find in measurement and automation systems where downtime is costly and where a delay could cause safety issues. There are some requirements the OS need to fulfill to be considered a real-time operating system. For instance, the system need to have a known maximum time for every critical operation and needs to be able to perform interrupt handling if necessary. The purpose of choosing a real-time operating system instead of a general operating system is to guarantee that the the program will run with a very consistent timing [14]. As mentioned several times before in this chapter a NB-IoT device has very limited supply of resources and at the same time very high timing constraints, which require a real-time operation system.

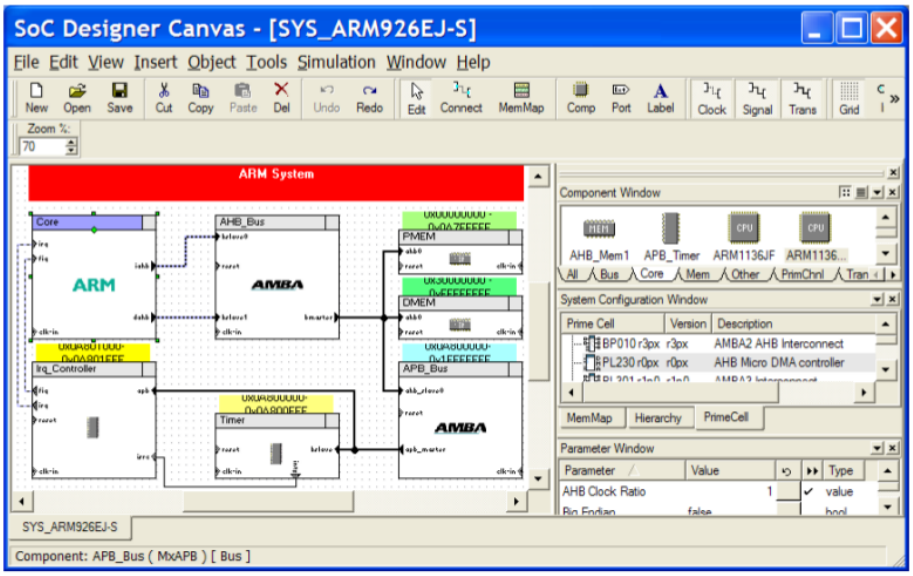


Figure 2.8: Combined Canvas Windows [2]

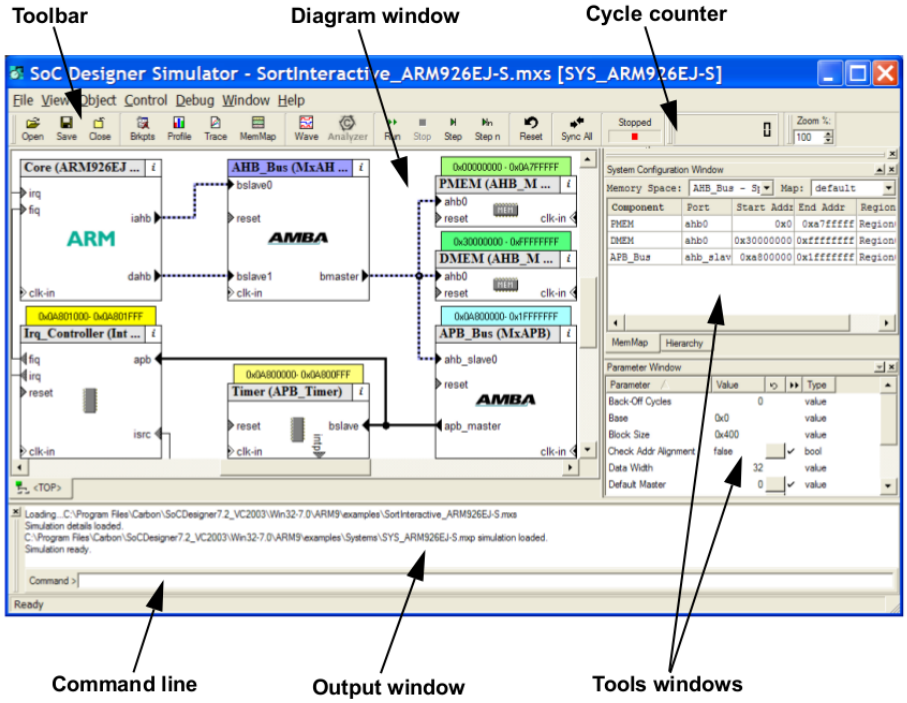


Figure 2.9: SoC Designer Simulator main window [2]

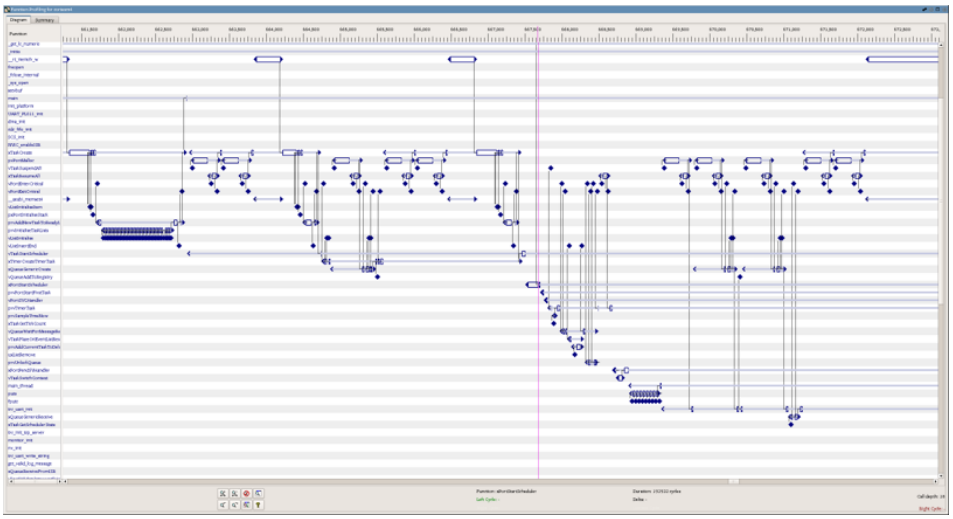


Figure 2.10: Function profiling window for CPU

Chapter 3
System

A description of the hardware and software system implemented in ARM SoC Designer. This includes specific hardware and software design choices when porting the system. Due to some confidential information belonging to ARM, some parts may not be explained or shown with the same level of accuracy as the rest of the system.

3.1 Hardware

The hardware system is built up in the ARM SoC Designer Canvas and is mainly covered by three separate parts connected to each other. The main parts are the processor system, the digital processing system and DMA controllers, seen in Figure 3.1.

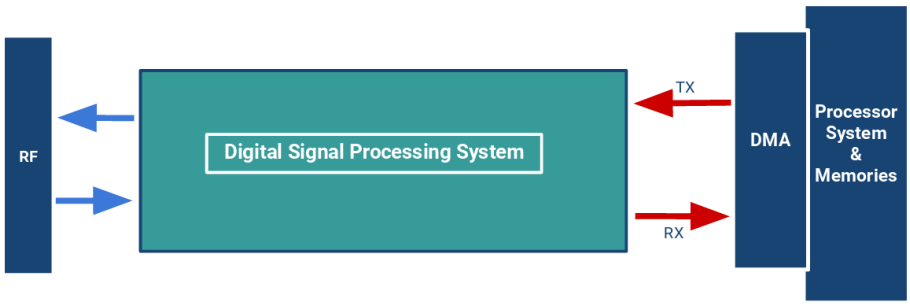


Figure 3.1: System overview of the hardware

The processor system is the primary part of the system and includes an ARM Cortex processor, a bus matrix, an Universal Asynchronous Receiver/Transmitter (UART) and memories. The bus matrix makes it possible to connect all hardware components to each other and to the same bus. By connecting all components to the same bus enables the possibility to use DMA controllers to transfer large amount of data throughout the system, when the processor demands it. The system's UART on the other hand does not add or change any of the system's

functionality and is only there as debugging support during the development phase. The UART enables the possibility to monitor the system during runtime which is done by adding messages in the software which during runtime is printed from the UART. The memories in the system is divided into two parts. The first part is used for execution and is where the instructions from the software is placed during runtime in order to make the processor do what it is programmed to do. The second part of the memory is used to store data. For instance, the test data that is passed into the system is stored at a specific place in this part of the memory and also the data after it has been processed is saved here. Also, while the data is analyzed in the software, the processor stores some parts of the data in this part of memory. One thing to note about the entire processor system is that all these blocks are all compatible with ARM SoC Designer. With a valid licence of the program all these blocks can be downloaded from ARM's IP exchange website and directly imported to SoC Designer Canvas without any modification.

The digital signal processing system contain the wrapped blocks required to perform the hardware side of a band scan. This part contain several subparts where each part within it processes the ingoing data in order to find and extract the NPSS before it is analyzed in the processor. An important thing regarding this part of the system is that it is operating in a different time domain compared to the rest of the system. Since it operates in another domain, this system also includes clock domain crossing FIFO's at every in and out port. Unfortunately the components inside the digital signal processing system part may not be described any further.

The third part of the hardware system are the DMA controllers. Even though they are compatible with ARM SoC Designer as the processor system described earlier in this section, they are here considered as standalone parts. The DMA controllers handle all big data transfers from the digital signal processing system to the memory and vice versa. The DMA transfer starts when the processor sends a call to the DMA with the size of the data and the source- and destination address. The controller starts to transfer data from the source to the destination address until all data has been sent. When the transfer is finished, the DMA sends an interrupt message to the processor in order to notify it of the completion of the transfer.

3.2 Software

The system software is a general implementation of an NB-IoT physical layer consisting three main parts. The layer one software services, the platform specific drivers and a real time operating system as seen in Figure 3.2 on top the hardware platform at the bottom.

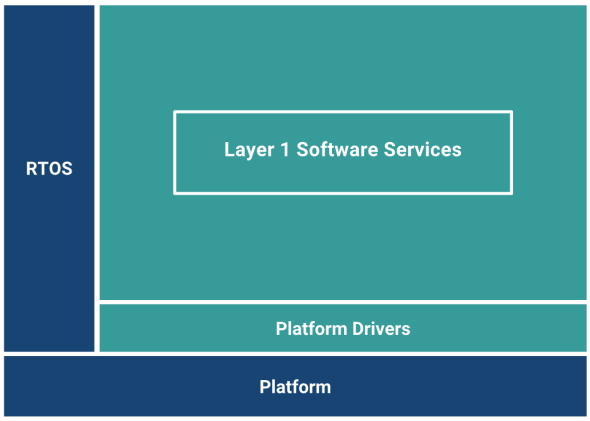


Figure 3.2: System software on top of the hardware platform

The layer 1 services include all functionalities from the software side to perform a band scan. In order to start any processes in the OSI-model layer 1, a call from a higher abstraction level to the physical layer is made. When a band scan call is made from layer 2, the processor initializes the hardware and sends a request to the DMA to send data from memory into the digital signal processing system. When the data has been processed in the digital signal processing system and again placed in memory by the DMA controller, the data is analyzed by the software in order to investigate if the data consist the desired NPSS or not. When this procedure is done the physical layer services returns the result to layer 2. Unfortunately the software procedures during a band scan may not be described any further.

The platform specific drivers are there to integrate the programmable hardware components, such as DMA controllers and UARTs, with SoC Designer and to make them work with the layer 1 services. For instance, when integrating a DMA controller with SoC Designer the block will have all functionalities to work properly, but without the information of how and when it should perform certain tasks or what different calls from the processor signifies. Seen in Figure 3.2, the platform drivers are placed in between the physical layer and the hardware platform and handle the communication between the two parts. The processor perform simple method calls to the platform drivers and the driver make sure the correct information is passed forward to the hardware block.

FreeRTOS is the systems real-time operating system (RTOS). An active open source project targeted at embedded systems which is designed to be robust, simple and easy to use [15]. FreeRTOS responsibilities involve prioritizing, scheduling and running user-defined tasks [16].

Chapter 4
Method

This chapter will describe the methods used to implement the wireless system and the choices made during the process. It will also cover the approaches made to verify the system.

4.1 Introduction

As an initial description, the hands-on part of this project is divided into two parts. One hardware and one software part. The hardware part includes the methodology and approach to integrate existing hardware design into the virtual platform ARM SoC Designer and the software part how to integrate ARM's software design and make it run in SoC Designer.

4.2 Hardware

From a hardware point of view the goal was to build up the required hardware blocks of the digital signal processing system in ARM SoC Designer in order to perform a band scan. This was done in several sprints where the initial phase entailed to first break down the current hardware design into separate blocks and then wrap each of them to make the blocks compatible in the virtual platform. Another sprint involved finding and integrate a solution to cross clock domains between different blocks in the design. In the last step an RF interface replacement solution was developed and integrated onto the system in order to feed data to the complete system.

4.2.1 Hardware Integration

In order to make the hardware design compatible with ARM SoC Designer, each and every HW-block in the design necessary to perform a band scan, had to be modified before it could be integrated into the virtual platform. To do this, a wrapper around the hardware components was created. The wrapper around the blocks made it possible to integrate the hardware into SoC Designer, but it also gave the possibility to add registers to the in- and out ports of the components to monitor every signal at each clock cycle.

The created wrapper can be described as a surrounding shell designed to be compatible to the platforms interface on the outside and to the developed hardware on the inside as seen in Figure 4.1. Since every wrapper is designed individually for each hardware component, different components can be connected to each other with help of the wrapper even if the protocols is made differently.

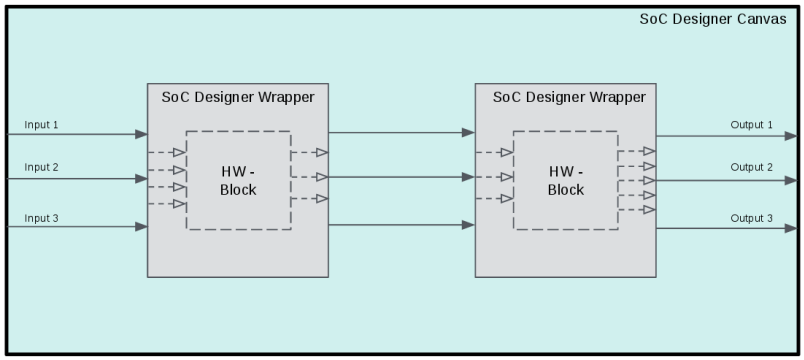


Figure 4.1: Systematic overview of how hardware blocks are wrapped to be compatible with ARM SoC Designer

A general description of how to wrap a hardware component

1. Create in- and out ports compatible to SoC Designer
2. Create interfaces to the ports in order to make them visible
3. Create signals compatible to the inner HW-block
4. Instantiate registers of the signals
5. Bind the clock signal from SoC Designer directly to the inner HW-block
6. Bind the signals to the ports of the inner HW-block
7. Implement a method to convert signals on the in port of the wrapper to the created input signals and from the output signals to the out ports.
8. Update sensitivity list to call the implemented method above when a triggering event happens on the in ports.

4.2.2 Clock Domain Crossings

As mentioned in the previous chapter the digital signal processing system operates on a different clock frequency compared to the rest of the system. In order to cross over from one clock domain to another, clock domain crossing FIFO's needed to be implemented between the two parts in the hardware. In this design the FIFO's worked as a buffer that received incoming data at one rate and transmitted on

another. They also had a built in memory to store values if the rate of the incoming data was higher than the outgoing data rate.

To implement the FIFO's into ARM SoC Designer the block needed to be wrapped with the same methodology as mention in the section above, but with ability to handle two clock frequencies. This was unfortunately a problem since SoC Designer did not have any standard solution for two different clock domains. To get around this problem with only one clock signal, the same frequency as the processor system was connected to the block and the frequency was then modified inside the wrapper. In the wrapper a parameter with the ratio between the input clock frequency and the desired clock frequency was set and with it creating a new clock signal with the input clock frequency multiplied with the ratio. Two clock signals with the correct ratio was created and could then be connected to the inner module.

4.2.3 RF Interface Replacement Solution

Once the wireless system containing CPU, memories, DMA controllers and the digital signal processing system was implemented into the virtual platform. A solution to stream data into the system needed to be developed in order to simulate a band scan in the hardware.

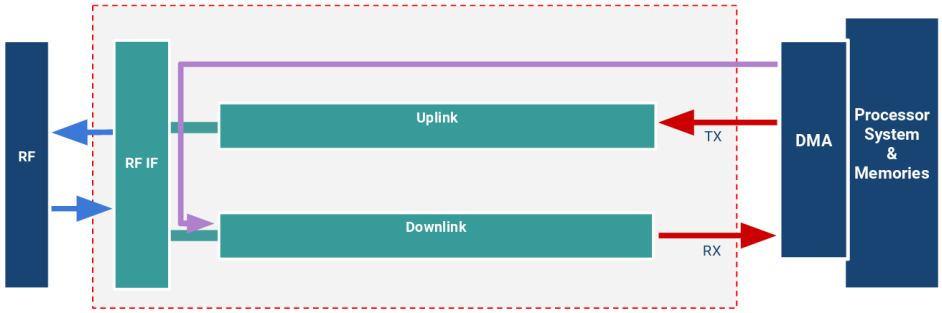


Figure 4.2: Schematic figure of the RF interface replacement solution

The solution to this problem was to use an additional DMA controller as a simulated RF interface as seen in Figure 4.2. By placing data in memory representing different scanned frequencies from LTE bands, a place in memory then represented scanned data at a specific radio frequency. By performing a call from the CPU to the DMA before every simulation, data would be streamed from the DMA into the digital signal processing system with the same behavior as if it came from a real RF interface.

4.3 Software

From a software point of view, almost all of the functionalities to perform a band scan was already developed at the time this project was initialized. However, all the functionalities was developed to run on an FPGA and not in the ARM SoC Designer environment. In order to make the software compatible to the virtual platform, all the platform specific implementations compatible with the FPGA had to be removed. The compiler had to be changed and new platform specific drivers for SoC Designer had to be developed.

4.3.1 Target Adaption

In order to make the already developed software project work in the ARM SoC Designer environment, the first step was to eliminate all target specific dependencies to the old target. This was done by updating all functions and defines connected to the platform into stubs. A simple example of how to stub a function can be seen in Listing 4.1 and 4.2.

Listing 4.1: Original method

```
int distance(int x, int y)
{
    if (x > y) {
        return x - y;
    } else {
        return y - x;
    }
}
```

Listing 4.2: Stubed method

```
int distance(int x, int y)
{
    (void) x;
    (void) y;

    return 0;
}
```

4.3.2 Compilation and Linking

Once all hardware specific dependencies was eliminated, the next step was to make the software compile with the compiler specified for ARM SoC Designer, ARMClang. Since the project towards the old target was built with GCC, some parts had to be added, removed, modified or rewritten in order to make the project build and link to an executable file.

4.3.3 Target Specific Implementation

With the compiler changed to ARMClang and all target specific methods stubbed. The software part was now ready to be integrated together with the hardware. The earlier stubbed methods required to perform a band scan were again implemented, but now with the functionality to work on the virtual platform instead. New addresses regarding in and out ports to the memories and peripherals was defined and new drivers was developed in order to receive and transmit data from the CPU to the rest of the system. Also a driver to the UART was implemented, by doing this the possibility to print debug messages during execution time was enabled.

4.3.4 Band Scan

Once all platform specific methods were implemented and the ports had been mapped with the correct addresses. The system now had functionality to transmit and receive data through the DMA controller and also the ability to navigate the hardware system by sending control signals directly from the processor to the hardware. The system was now finally ready to make use of the already existing methods to perform a band scan.

To be able to perform a band scan, a portrayed layer 2 call had to be implemented in the physical layer. This was done by creating a thread containing the required calls to set up the correct path in the hardware and to enabling the components needed. Once a band scan call had been made the command was placed in a queue until the system was ready to perform the commands. The state machine of the band scan procedure can be seen in Figure 4.3.

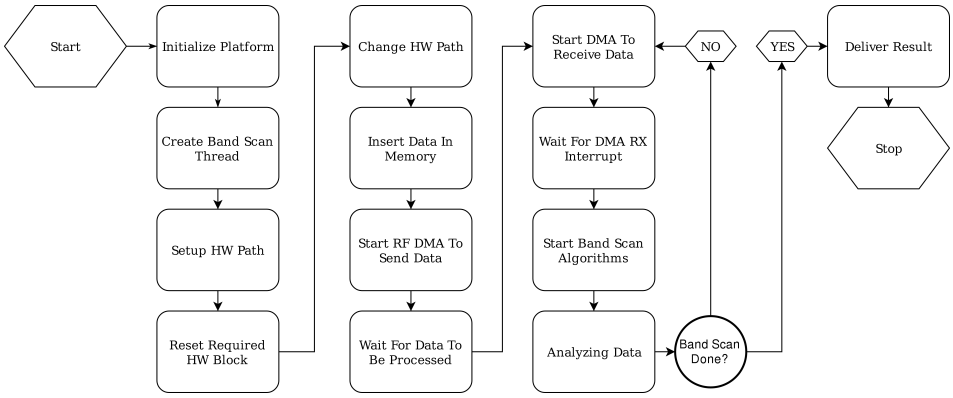


Figure 4.3: Software State Machine

4.4 Testing and Validation

When developing a complex system such as a SoC, every implemented part of the system needs to be tested and validated thoroughly. During implementation of the NB-IoT system in to ARM SoC Designer, the four main levels of software testing approach were adopted. This meant the testing process was split up into four lesser stages; unit, integration, system and acceptance testing [17]. By adopting this verification approach, both hardware and software could separately be proven to work properly before putting them together to perform the system and acceptance testing stages.

The hardware testing approach could be described as the following. Starting out by testing every hardware block wrapper implementation by itself in the SoC Designer environment, together with the same test data as the one used to verify the component at a register transfer level. A first step to assert the system is made by verifying the wrapped blocks returned the same values as the original implementation without the wrap. The blocks were then tested in a much-simplified version of the desired system with a so-called ramp, where the lowest possible to the highest possible value was sent through the system to see if the blocks worked and interacted together as intended. If the result was valid, another block was added and so forth until the entire system existed in SoC Designer.

Due to the existence of different time domains in the system, the crossing between two different clock frequencies also had to be tested. This was done in a similar way as before, with a ramp, which made it obvious if there were any loss or addition of data in the crossings between time domains. Once the unit and integration tests were working as intended, the hardware system was tested with a real scenario radio test vector sent with a DMA controller from the processor memory to the hardware chain before being fetched by another DMA controller and placed back in memory. The processed result in memory could then be compared to the results processed from a Matlab reference model representing the same system.

The software side of the system were mainly tested with the help of ARM SoC Designers great software debugging capabilities, which made it possible to display exactly what process were being executed at the current clock cycle. For instance, this made it possible to discover where timeout or execution errors occurred. The timing of the different software processes were then tested in the same manner with SoC Designer. This to verify the processes were performed in the correct order and that each process had enough time to execute. To conclude the software system testing, a radio test vector were ran through a Matlab reference model representing the hardware and later placed in the memory to be used to perform a band scan. By doing this, the software could be decoupled from the hardware and tested individually. By comparing peak(s) found in the Matlab model with the ones in the software, the model would be considered as working when the two models generated the same answer.

Once the respective hardware and software testing procedures were completed, the combined system of them put together were tested with the real scenario test vector. The execution scheme was verified to work as the intended state machine in Figure 4.3 and the number of peaks found was verified to be the same number

of peaks found in the Matlab reference model.

The implemented system in SoC Designer was lastly acceptance tested by running multiple variants of radio test vectors and comparing them to their equivalence in Matlab to see if different scenarios would have any impact on the intended results.

To summarize, the testing and validation process looked like the following:

Hardware

a. *Unit.* Test every block individually with previously generated test data.

b. *Integration.* Test blocks together with each other by adding additional blocks to functional system one at a time.

c. *System.* Test complete hardware implementation with radio test vector.

Software

a. *Unit.* Test the different parts of the band scan algorithm.

b. *Integration.* See if the different parts of the band scan algorithm run in the correct order.

c. *System.* Run a hardware processed radio test vector and compare to the results to a Matlab representation of the system.

Full System

a. *System.* Test the two implementations together with a radio test vector.

b. *Acceptance.* Test the system with multiple variants of radio test vector and compare the results to their equivalence in Matlab.

Once all these steps were working as intended, the system would be concluded as working.

Chapter 5
Results

This chapter will present the results carried out from this Master's Thesis and the data used to represent a normal use case.

5.1 Measurements

In order to verify the system, three NB-IoT test vectors where used. Each test vector contained sampled NB-IoT test data separated with 300kHz. Test vector 1 was sampled at frequency f, test vector 2 at frequency f + 300kHz and test vector 3 at frequency f + 600kHz. All test vectors contained the same number of sampled data points but only test vector 2 was sampled at a frequency containing the desired NPSS, hence only the output result from test vector 2 was supposed to indicate that a NPSS was found. The indication of a found NPSS can be distinguished by looking at the output stages of Figure 5.1 to 5.3. If a NPSS is found, a distinct peak with a much higher amplitude compared to the rest of the signal will be visible.

In order to visualize and validate the result, the vectors were processed through the system and extracted after specific stages in the system. Figure 5.1 to 5.3 represent the results of the three different datasets processed through the implemented hardware. Each figure contains the input signal and three different stages of processed data, band scan stage 1, band scan stage 2 and band scan stage 3, where the third stage also is the output signal. As expected, only Figure 5.2 contains a distinct peak at the output stage, which indicates that the test vector contains the desired NPSS. Worth noticing is the scale of the y-axis at the output stages. In Figures 5.1 and 5.3 where no desired signal is found, the amplitude is varying from -400 to 400 compared to when the NPSS is found in Figure 5.2, where the amplitude of the peak is over 20 000. When detecting a peak, only the relationship between the amplitudes are of importance and the reason why the amplitude is left without any specified unit.

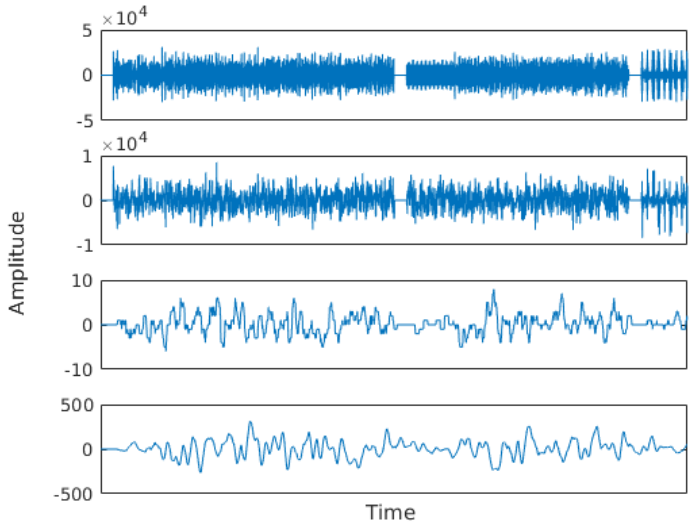


Figure 5.1: Test vector 1: a) Sampled input data b) Band scan stage 1 c) Band scan stage 2 d) Band scan stage 3

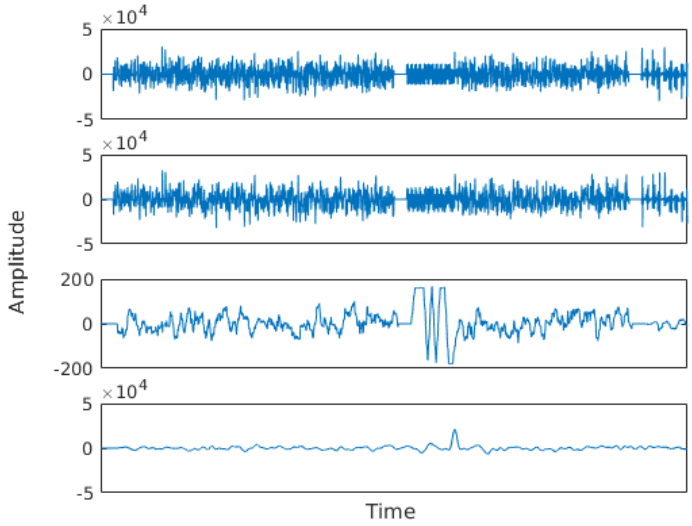


Figure 5.2: Test vector 2: a) Sampled input data b) Band scan stage 1 c) Band scan stage 2 d) Band scan stage 3

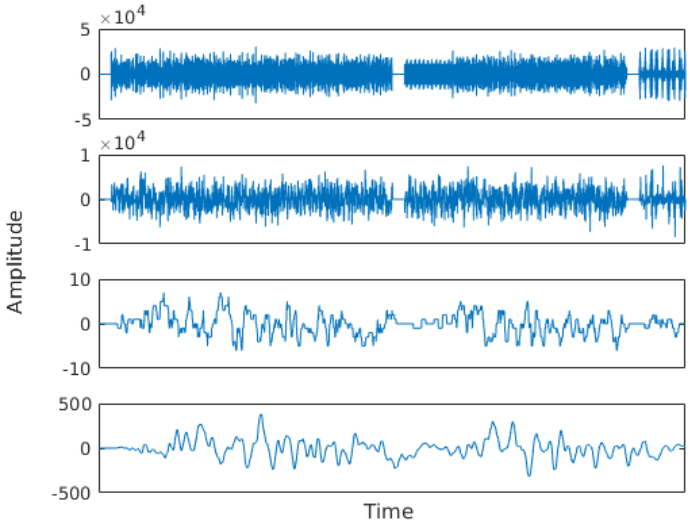


Figure 5.3: Test vector 3: a) Sampled input data b) Band scan stage 1 c) Band scan stage 2 d) Band scan stage 3

5.2 SoC Designer and Matlab Signal Comparison

As described in Section 4.4, the method to verify and evaluate the systems processed data was to compare the data to values extracted from a Matlab reference model after each modification stage.

To verify the system, the previously mentioned test vectors from section 5.1 were passed through the two models to find if they processed the signal equally and if the results after each processed stage were the same. Since the Matlab reference model were stated to process the data correctly, this model were used as the true value and the SoC Designer model could only be considered correctly implemented if the difference between the Matlab- and SoC Designer model were zero.

Figures 5.4 to 5.12 shows a comparison of the test vectors at different stages when processed through Matlab and ARM SoC Designer. It also shows the difference between the two signals. As seen in the figures the processed signals looks identical to each other, notably the difference between them are zero. This implies that the signals not only looks to be the same, they are in fact identical to each other and the result of the processed signals will be the same independently if signals are processed in Matlab or in the virtual platform. By using different test vectors and passing them through the two models, it is also shown that the signals are processed equally regardless if the signal contains the NPSS or not.

Figures 5.4 to 5.6 show a comparison of test vector 1 at three different stages of the process. By extracting the data at different stages makes it possible to follow how the data is processed and changed along the way in the model. In Figure 5.4 the data has been processed through the first band scan stage. By looking at the

figure it can be seen that the two signals are equal, but it is hard to extract any other information from the graph. In Figure 5.5 and 5.6 the data from test vector 1 has been processed through the second and the third band scan stage. As can be seen in the figures, no distinct peak is found which means the signal most likely do not contain any NPSS.

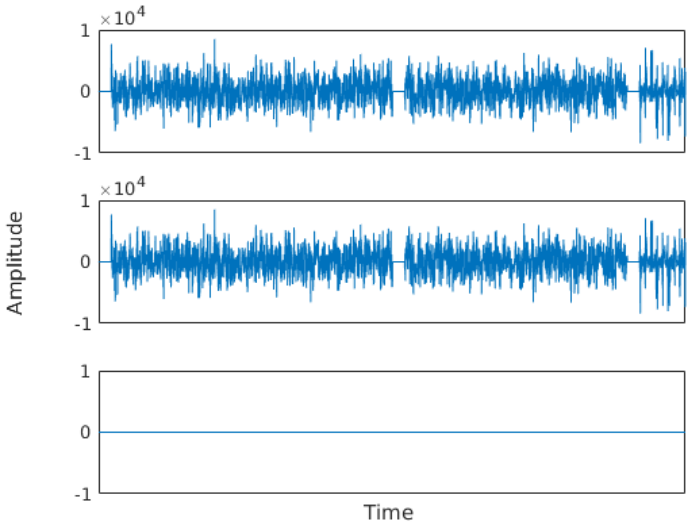


Figure 5.4: Test vector 1: Comparison of band scan stage 1 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

Figure 5.7 to 5.9 shows a comparison of test vector 2 at the same stages of the band scan process as the first vector. At band scan stage 1 the signal looks very similar to the signal in Figure 5.4, but after the signal has been processed though band scan stage 2, something starts to happen around 6/10 of the time (Figure 5.8) and by looking at Figure 5.9, when the signal is passed trough the third stage, a distinct peak is found which indicates the data processed through the system most likely carries the desired NPSS.

Figure 5.10 to 5.12 shows a comparison of test vector 3 processed through the two models at the same three stages as before. In this case, the signal after the first band scan stage (Figure 5.10) looks very similar to the other two signals from test vector 1 and 2 (Figure 5.4 and 5.7). In Figure 5.11 the signal is processed through the second stage and by looking at the amplitude and then comparing it to the amplitude of the signal in Figure 5.8 the content of the signal can be considered, from a band scan perspective, to contain nothing of value. This also reflects the outcome after band scan stage 3, were no peak is found (Figure 5.12).

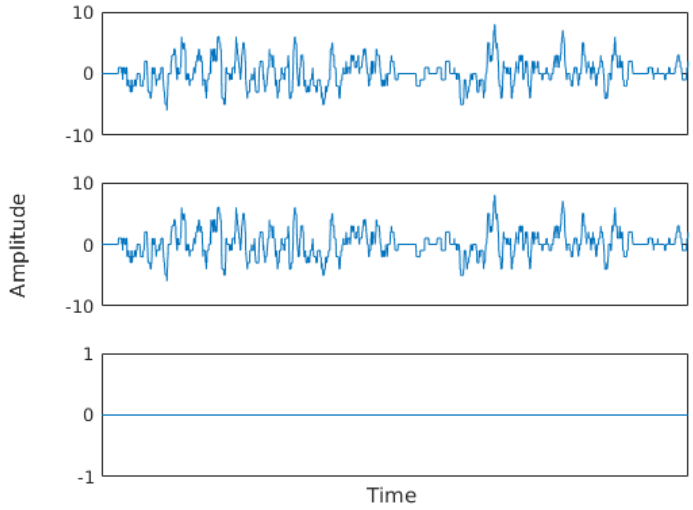


Figure 5.5: Test vector 1: Comparison of band scan stage 2 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

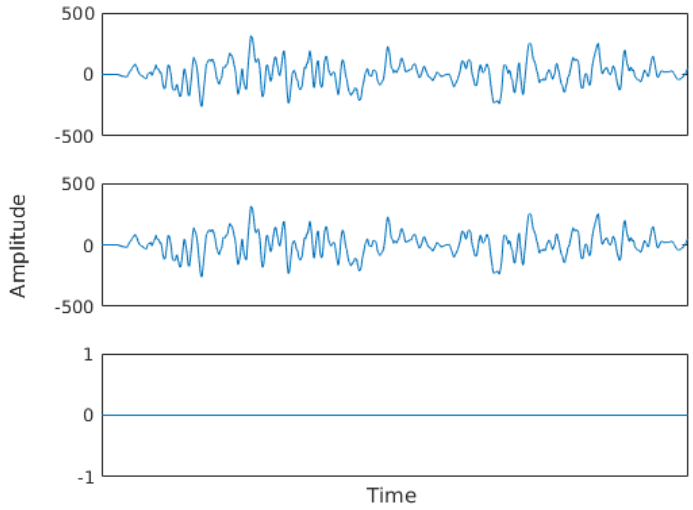


Figure 5.6: Test vector 1: Comparison of band scan stage 3 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

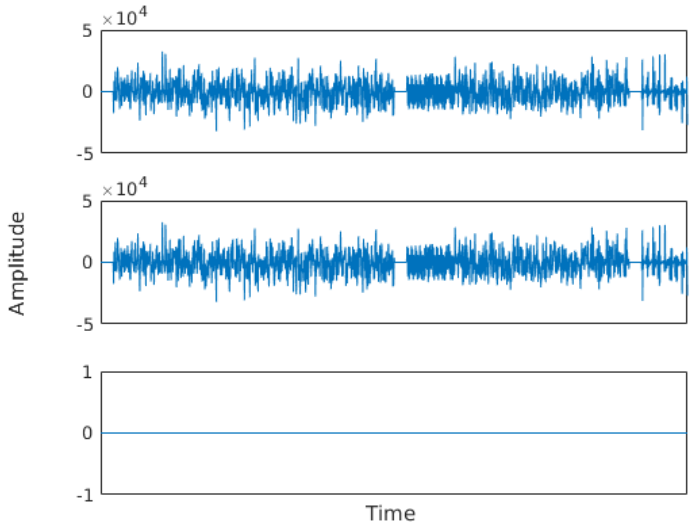


Figure 5.7: Test vector 2: Comparison of band scan stage 1 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

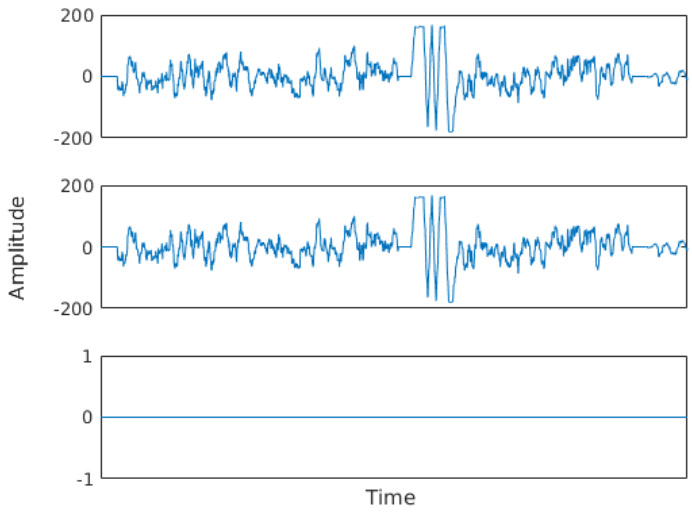


Figure 5.8: Test vector 2: Comparison of band scan stage 2 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

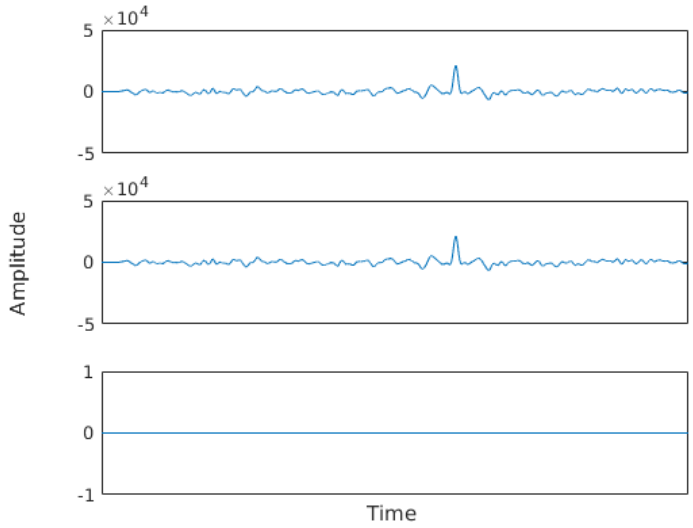


Figure 5.9: Test vector 2: Comparison of band scan stage 3 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

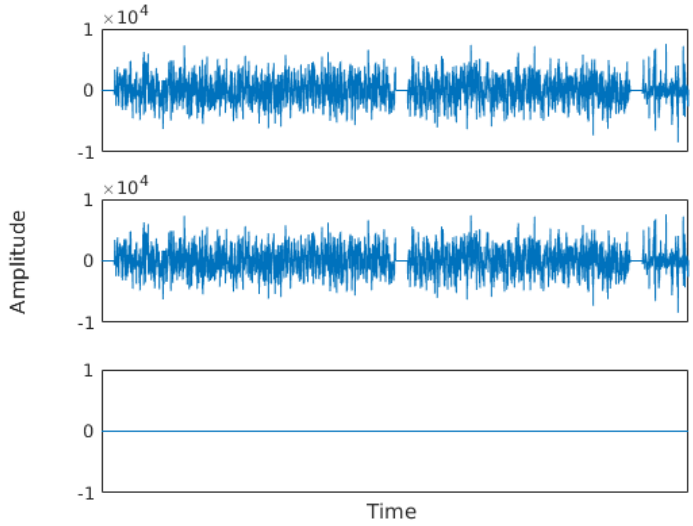


Figure 5.10: Test vector 3: Comparison of band scan stage 1 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

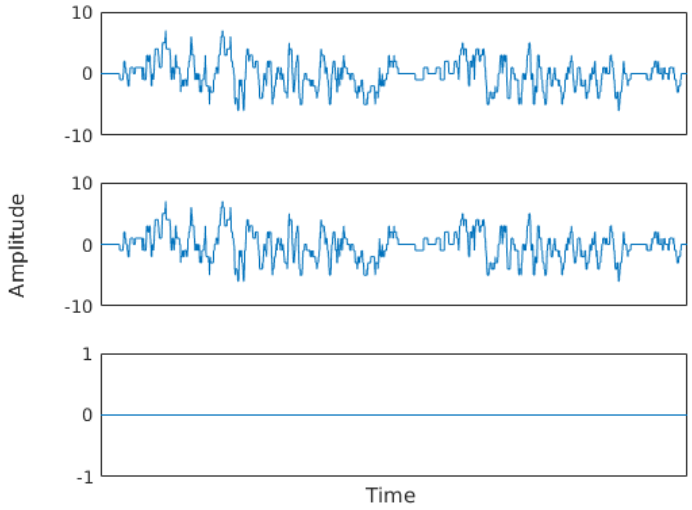


Figure 5.11: Test vector 3: Comparison of band scan stage 2 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

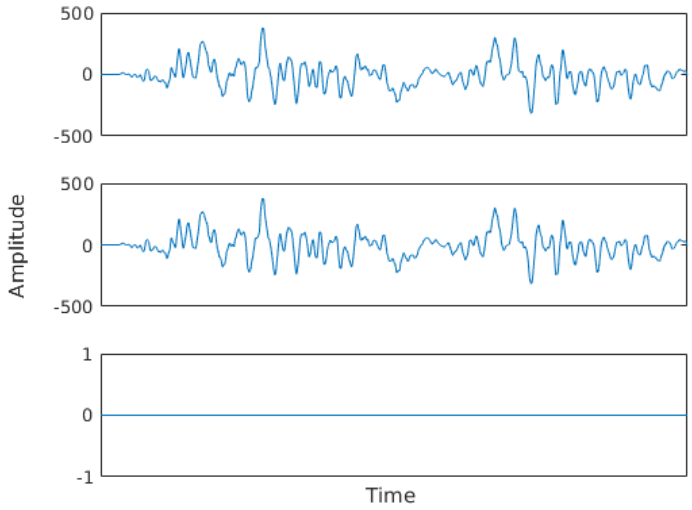


Figure 5.12: Test vector 3: Comparison of band scan stage 3 in (a) Matlab and (b) ARM SoC Designer and (c) the difference between them

Even if only the result from test vector 2 contained the desired NPSS, all three test vectors used and processed though the system describes real user scenarios. The fact that all signals independent of at what stage the data has been extracted from, gives the same result when it is compared to the Matlab reference model. This indicates the validity of the system and that the model implemented in ARM SoC Designer can be used as a valid representation of the digital signal processing system of the NB-IoT design.

5.3 Performance

In order to evaluate performance, the simulation speed had to be measured and assessed. The SoC Designer Simulator measured the frequency of the simulation during execution of the final system to approximately 13kHz.

During hardware development, the size of the system increased for each component added onto the platform. As the hardware system became larger, the performance started to drop. By running the simulation multiple times, where the first simulation only contained the processor system and at each new iteration adding subsystems one at a time until the entire system were implemented, the performance at different hardware sizes of the system could be measured. The result of the measurement is presented in Table 5.1 and shows the performance of the simulation is dependent on the size of the implemented hardware into SoC Designer. When only the processor system is implemented, the system performs with a frequency of 67.1kHz compared to when the entire HW system is implemented and the performance has decreased to only operate at 13kHz. Worth noticing is the non-linearity to the loss of performance while adding subparts into the system. As a result, the performance is more dependent on the complexity for each added hardware block and not in the number of blocks added to the system.

To investigate if the hardware of the workstation hosting the simulation and if the size of the implemented system had any impact on the computer load during simulation. A monitor of the CPU was opened, as the one seen in Figure 5.13. By enabling this monitor it could be seen that only one of the cores in the processor during simulation operated at 100% while the workload of the other cores were almost 0%. Revealing a major bottleneck with the SoC Designer simulation, where it does not support the use of multiple cores during execution.

Table 5.1: Measurements of how the performance is dependent on the size of the hardware system

Hardware size	Performance (kHz)
Processor system	67.1
+ Subsystem 1	26.3
+ Subsystem 2	19.2
+ Subsystem 3	17.9
+ Subsystem 4	15.4
Final HW System	13.0

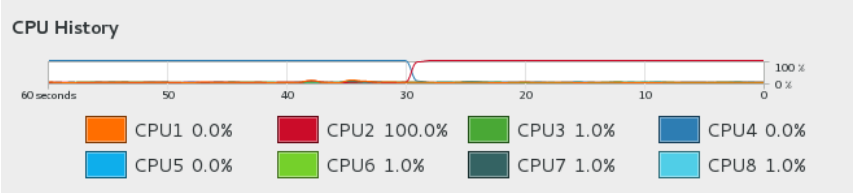


Figure 5.13: Computer load during simulation

5.3.1 Band Scan

Since a band scan involves scanning an entire frequency band and evaluating every frequency segment within it, the worst case scenario would be to perform a band scan on a very wide band. By considering one of the widest bands to be 45MHz wide, the number of frequencies required to scan the whole band would be 150 different frequencies if each frequency segment is separated with 300kHz. In a real case scenario this would not be of any problem since the radio module would scan each frequency segment one at a time and process it though the digital signal processing system. But due to the RF interface replacement solution discussed in section 4.2.3 and the memory restrictions in the system, a band scan containing 150 different data sets would not be possible to perform, since 150 sets of data would not fit in the memory used for this system. Instead a band containing only seven different frequencies were performed. This was performed by placing seven different test vectors in the memory and starting the band scan procedure, which involved one initialization phase and seven individual frequency evaluations performed in the system. As can be seen in Figure 5.14, the number of clock cycles it takes to evaluate a certain frequency containing sampled data is the same regardless of it being the first or the last frequency to be processed. By taking this into account together with the obtained operating frequency of the system, the time it takes to perform a full scale band scan can be calculated as equation 5.1. Where t_{init} is the time needed to run the initialization phase, t_{fscan} is the time needed to evaluate a singular frequency and t_{total} is the total time needed to run the initialization phase plus the number of different frequencies in the band. Equation 5.2 and 5.3 show the time difference and the time needed to simulate the worst case scenario of an entire band of 150 frequencies with the speed of only the processor system (5.2) compared to the complete larger system (5.3).

When performing the band scan evaluation of seven different test vectors, only one of the vectors contained any valuable information regarding the band scan procedure. By taking this into account while looking at Figure 5.14, it can be concluded that the evaluation of each frequency takes the same amount of time to perform, regardless if the desired NPSS is found or not. Which indicates the estimated time calculated in Equation 5.2 and 5.3 to be a valid estimation.

How to calculate the time it takes to perform a band scan

$$
\begin{aligned}
t_{init} &= \frac{\text{nbr of clock cycles for initialization phase}}{\text{simulation speed}} \\
t_{fscan} &= \frac{\text{nbr of clock cycles per freq}}{\text{simulation speed}} \\
t_{total} &= t_{init} + (\text{nbr of freq in a band}) \times t_{fscan}
\end{aligned}
\tag{5.1}
$$

From Figure 5.14 the following data can be extracted:
nbr of clock cycles for initialization phase = 2 083 655
nbr of clock cycles per freq = 697 890

Considering bandwidth of 45MHz:
nbr of freq in a band = 150

Time consumption **if** the system would work with
the processor system only, simulation speed = 67.1 kHz

$$
\begin{aligned}
t_{init} &= \frac{2\,083\,655}{67\,100} = 31.1\text{s} \\
t_{fscan} &= \frac{697\,890}{67\,100} = 10.4\text{s} \\
t_{total} &= t_{init} + 150 \times t_{fscan} = 1591.1\text{s} \approx 27\text{min}
\end{aligned}
\tag{5.2}
$$

Time consumption with complete hardware system,
simulation speed = 13.0 kHz

$$
\begin{aligned}
t_{init} &= \frac{2\,083\,655}{13\,000} = 160.2\text{s} \\
t_{fscan} &= \frac{697\,890}{13\,000} = 53.7\text{s} \\
t_{total} &= t_{init} + 150 \times t_{fscan} = 8212.9\text{s} \approx 2\text{h } 16 \text{ min}
\end{aligned}
\tag{5.3}
$$

Table 5.2: Measurements of the number of clock cycles it takes to handle an interrupt in regular and in writing-to-file mode.

Procedure	Clock Cycles	Time (s)
Regular	22 695	1.75
Write to file	5 517 970	424.5

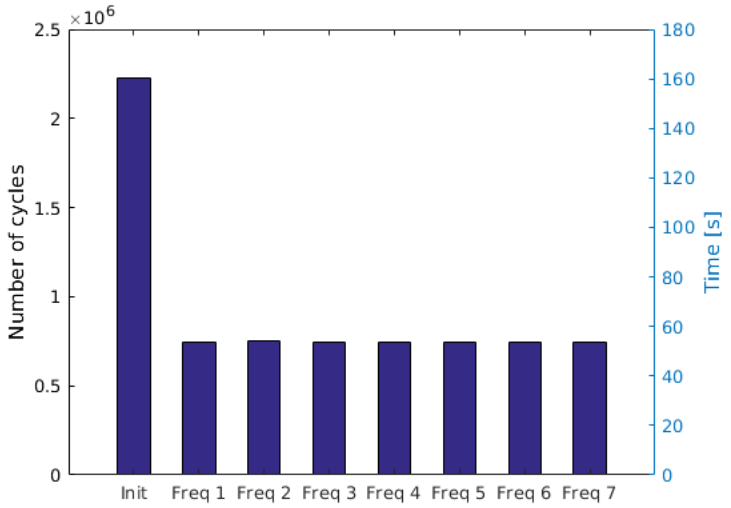


Figure 5.14: Bar plot of the number of clock cycles it takes at each step in the simulation and the approximate time at a clock speed of 13kHz

5.3.2 Data Extraction

As mentioned in 5.1 the data had to be extracted from the system after certain stages of the process in order to compare and verify it against the Matlab computed results. The procedure of doing this was to use an UART to print the content of the memory to a text file once an interrupt declared the processed data had been transfered into memory. This was of course a very time demanding procedure and by measuring the number of clock cycles it took to handle an interrupt with and without writing to a text file, the result in Table 5.2 could be extracted. The difference between the two could be calculated to be approximately 5.5 million clock cycles and and adding 423.75s on top of the regular mode simulation time to run in writing-to-file mode.

Chapter 6
Discussion

In this section the subject of the Master's Thesis will be discussed with respect to the research questions and sub-questions from Chapter 1. This will lead to a discussion about ARM SoC Designers current uses, possible improvements and other general thoughts about the project.

6.1 ARM SoC Designer

ARM SoC Designer is a powerful tool when it comes to developing hardware and software together. It gives great support for debugging and verification at a cycle accurate level. SoC Designer also gives a very precise picture of the system and how it performs. However, the platform may be powerful to use during development, but there is also a learning threshold with the user experience and documentation. In this section the experience gained working with ARM SoC Designer during the project will be discussed.

As mentioned earlier the debugging capabilities in this program is very powerful. Starting from a hardware perspective it is really beneficial to have the ability to monitor all the signals of each component in the entire system at every individual clock cycle. By having such control it becomes very easy to validate the correctness and performance of the system. Thus therefore also simple to find where and when something in the system is failing or when a component is not working as it should. Another good feature is the ability to count and measure clock cycles for a specific component to perform its task, for instance to find bottlenecks somewhere in a system. By also adding breakpoints to the in- and out ports of the component makes it easy to perform measurements. However, this could be made easier by adding support to set breakpoints directly on the wires instead. As it is right now, only specific protocols are supported to set a breakpoint in SoC Designer. This is not a crucial problem, but the discontinuity adds to a lesser user experience. More about the user experience and how it is to work with the program will be discussed further down in this section.

Developing and debugging embedded software using ARM SoC Designer is also a great experience. The possibility to use the processor profiling function to follow how the software are being executed at every clock cycle gives a great insight in how the software is developed and even for the most complex systems it is relatively easy to get involved in how the system is designed by only following the

execution flow. This feature makes it also very easy to debug since every execution step can be visualized. One drawback of developing software with the help of ARM SoC Designer is the time it takes to execute. In order to execute software in SoC Designer the CPU and necessary memories needs to be implemented in hardware, this will make the software dependent on that hardware, which in return will slow down the execution. This is of course the price to pay in order to get all the debugging capabilities the tool is offering. Another drawback with the tool is the incomparability to use any other compiler than ARMClang. Even if changing from one compiler to another should be a relatively straight forward approach to do, it is still a drawback that will slow down the development process.

One of the greatest strengths using ARM SoC Designer is the ability to co-simulate hardware and software. By doing this, both hardware and software can be verified to work properly as an integrated system and also to find out if any system requirements are not fulfilled by either part. Using the program also give the possibility to investigate special use cases at a cycle accurate level, in order to find the exact amount of required memory, or if there is any bottlenecks in the hardware design that will slow the system down or causing failures. Unfortunately the execution speed is a drawback which makes it hard to develop entire systems on the platform. One drawback especially crucial to the co-simulation mode is the lack of ways to export data in a sufficient way. The current solution to extract data is to use a UART to print from memory to a text file char by char which is a very time-consuming process. It would be much more effective if there instead would be a way to export a fraction of the memory to a text document all at once.

Even if ARM SoC Designer is a great tool to use during hardware and software development, there is a couple of things that make the program lose some of its potential. This is not because of some important feature missing, instead as mentioned before the user experience does leave a lot to wish for. First thing to notice when using the program is how tiresome it is to work in the Canvas. The environment which is designed as a block diagram editor would really need a fix-up. For instance is the procedure of rearranging and connecting blocks to each other a rather clunky procedure. Sometimes it works fine, but most of the time something completely different will happen after adjusting the block in the canvas as seen in Figure 6.1. Another thing that can be really painful is the lack of information when importing components to the library. A small error in a wrapped component can be very hard to find since the failure will make the program refuse importing the component without any error or crash messages.

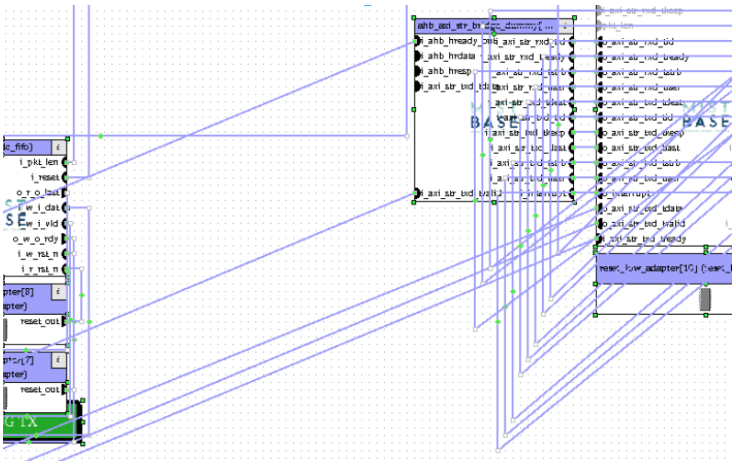


Figure 6.1: Snapshot from ARM SoC Designer Canvas after rearranging blocks

Once the system is built up in the canvas and ready to be tested in the simulation environment some other bugs worth knowing of might appear. To start with, the simulation will not open if any of the master ports from the wrapped components are unconnected. This becomes quite annoying since there are no way to disable unused ports in the program, instead the only way to get the simulation running is to create a stub with the exact same in port type as the master port that will absorb the signal. Another thing worth noting is the inability to connect one master port to more than one slave port and to make it work properly. Somehow the program does not have the ability to drive a signal from one out port to more than one in port. So instead of connecting an out port to multiple in ports, a splitter needs to be used in order to benefit one signal to multiple targets.

6.2 System Accuracy

ARM SoC Designer as a service have a lot of potential. But if the systems implemented are not a valid representation of the real world during cycle accurate simulation it would not be of any use in the end. Therefor it is necessary showing comparison data between SoC Designer and Matlab model to support and verify its accuracy. In this report three different data sets have been used to represent three different frequencies in a baseband. These can be seen as the input signal, the band scan stage 1, band scan stage 2 and band scan stage 3 in Figure 5.1, 5.2 and 5.3. Though only one of the data sets contain a sought energy peak after the last band scan stage (see Figure 5.2) with the other two containing no such energy peaks. This represents a small scenario of scanning a baseband with frequency segments both with energy peaks and without for the best one to transmit and receive data over.

As mentioned in Section 5.2, Figure 5.4 to 5.12 shows a comparison between the processed signal through ARM SoC Designer and Matlab and that the differences of the modified signals are zero to show that they are identical. This establishes that the implementation in SoC Designer are valid and that the tool can represent an entire system without modifying the in or out data. This open up SoC Designer to being a very potent tool when evaluating memory requirements and data throughput since the modulation in SoC Designer can be seen as a valid representation of real hardware during the simulation and can therefor give an indication of when memory size and data throughput should be enough to support a specific use case. Even if the focus of this thesis have been to state a proof of concept if it is possible to integrate a wireless system into SoC Designer, the correctness of the results indicates that the possibilities to use the program for designing sizes of buses and memories could be very beneficial.

6.3 System Performance

When evaluating the performance of the ARM SoC Designer model the first thing to notice is that the initiation phase of the band scan procedure take up quiet a lot of time before the actual scanning procedure start and when the scan procedure has started it is pretty obvious from the results in Figure 5.14 that the number of clock cycles and therefore the time consumption are not dependent on what type of data it receives on each frequency. Instead the factors affecting the time consumption of a band scan is the size of the frequency band and maybe most significantly the frequency of the clock the system runs at.

One of the biggest worries in the beginning when deciding to implement a complete system in ARM SoC Designer was how much of an impact the HW size would have on the simulation performance. As the results shown in Table 5.1, these worries were confirmed by performing performance measurements of smaller subsystems instead of the entire design. It is pretty significant going from 67.1 kHz with only the processor system to the final HW system with a clock cycle speed of 13.0 kHz (Table 5.1). With a frequency of only 13 kHz the the time it takes to initialize the system will be 160.2s (5.3) instead of only 31.1s (5.2) at 67.1 kHz. Not to mention if the system needed to go through an entire band scan, which could involve analyze up to 150 different frequencies where at 13.0 kHz that would take approximately 2h 16min (5.3) compared to only 27min (5.2) at 67.1 kHz.

Unfortunately this performance is not good enough for most use cases on a complete system. Since a full scale system usually would have multiple other processes ongoing which are needed to be executed before and during the specific task to debugging or troubleshooting, the speed will probably not being fast enough. This would create much more down time for the developers and testers than what many companies would probably want. Also worth mentioning is that these simulations are done at a rather good workstation with an Intel Core i7 processor and 16GB of RAM. But still this is not enough.

Different types of solutions to this problem are only speculative claims since no data are produced to support them. But one type of solution to this problem might be to run these simulations in a computer cluster with multiple connected

computers to bring up the performance. But, as can be seen in Figure 5.13, the tool only seems to make use of one core as it runs and therefore this may not be the solution to look for.

Another opportunity, which seems like the more interesting alternative, is to use ARM SoC Designers non cycle accurate branch Fast Models. Where the simulation is executed at a higher pace, but at the expense of not being cycle accurate to a specific point in the execution. The most beneficial approach would probably be if these two parts could be combined to run a single simulation in both modes, where only the part under investigation would run in cycle accurate mode. Unfortunately the methodology of using and implementing IP into Fast Models today deviates significant compared to SoC Designer, which makes the integrated design in this project not compatible for this kind of simulations.

6.4 Summation

To summarize the experience and results gained of working with ARM SoC Designer it can be said that it has a lot of potential to be a widely used complement to the traditional SoC development using an FPGA. However, the virtual platform is still not mature enough to replace the FPGA entirely. With that said, companies can have a lot to gain using it as a test program for standalone components or subsystems, since the flexibility of adding components in the drag and drop environment is very useful. But to develop an entire system only by using SoC Designer as the target platform will today be a too slow development method compared to an FPGA. One of the main things to this is the slow turnaround time due to the clock frequency. As mention before one possible solution to this may be to make the program make use of more than one of the hosting workstations processor cores at the same time to speed up the program.

Even if the tool today is not fast enough to develop entire systems, it can still be very beneficial to use it for final design evaluation. If a system is verified to work properly on an FPGA an integration of the system to the virtual platform could then be used to assert the exact hardware requirements to use before taping on silicon.

Chapter 7
Conclusion

Companies are always looking for different ways to improve their development process to give them an upper hand on their competitors. The software tool ARM SoC Designer tries to accomplish this through adjusting and streamlining traditional System-on-Chip development procedures and by adding debugging and troubleshooting capabilities which are not present on current FPGA boards.

This Master's Thesis explored the possibilities of ARM SoC Designer and how it can be used in combination with a complete wireless NB-IoT system to develop SoC's in a virtual environment. The system have been evaluated from aspects such as turnaround time, flexibility and from the debugging capabilities the tool enables. The tool have also been evaluated from a subjective point of view to present how it is to work in SoC Designer.

The thesis have shown that the implementation in ARM SoC Designer is a valid representation of the preexisting wireless NB-IoT system currently developed on an FPGA. The results have been proven to be correct by comparing them to a reference model in Matlab and proof of concept have been shown by successfully performing a band scan. It have also been shown that SoC Designer adds some very useful functionality to traditional SoC development techniques but not without some significant flaws, which at the moment will not make it possible to completely replace FPGA boards with ARM SoC Designer.

Chapter 8
Future Works

This Master's Thesis have presented a start of how ARM SoC Designer could be used as a development tool when designing System-on-Chip's. It has been proven that it is fully possible to integrate a wireless system onto the virtual platform and also how developers can benefit from the debugging and verification capabilities the program offers. In this chapter discussions regarding the remaining work building up the entire NB-IoT system in SoC Designer will be emphasized together with some general ideas of things that can be done to develop SoC Designer further.

8.1 Scaling

So far in this project the required components needed to perform a band scan has been implemented into ARM SoC Designer, but to get the entire NB-IoT system to completely run in a virtual environment there is still some work to do. Even if the main parts of the downlink can be considered to being integrated, the uplink is still needed to be deployed in order to get the complete system integrated.

In order to simulate the entire layer 1 system virtually the back end of the system also needs to be implemented. Theoretically this could be done but the question to answer here is how the frequency will be affected of the much bigger implemented hardware. As already seen in table 5.1 the performance of the system are strongly dependent on the size of the hardware.

8.2 Performance

As discussed in the earlier part of this chapter the decline of performance as the system increases is very crucial for the use of SoC Designer as a development tool. To avoid this loss of performance while the projects is getting bigger, the first step would probably be to investigate if there is any possibility for the program to run on several cores at once. As noticed in figure 5.13 the current version of the program only make use of one core. Another interesting thing to investigate regarding the performance is the possibility of implementing Fast Models into the cycle accurate design and how such of an implementation would affect the turnaround time.

8.3 Radio Module

Even if the wireless system one day will be represented completely virtually, there will still be some problem to represent it as a good reference model if there is not any proper solution to the radio and how to stream real data into the system. As mentioned in the Chapter 1, one part of this project was to evaluate the possibilities to streaming data instead of using test vectors to pass in to the system. When starting this Master's Thesis one could relatively fast conclude that the workload of such a investigation would probably be of the size of a second Master's Thesis and has therefore not been analyzed. But the investigation would still be interesting to perform. One interesting thing would be to investigate the possibilities to integrate a radio card from the hosting workstation to be accessed of the virtual platform. If this could be done, a bridge between the real world and the simulation environment will be created which will enable the system to directly test real data transfered over the air.

References

[1] ARM. Esl design using arm realview soc designer. In *ARM Connected Community Technical Symposium*, pages 3–5, 2006. http://read.pudn.com/downloads81/doc/312825/12-esl%20design%20using%20arm%20realview%20soc%20designer.pdf.

[2] ARM Limited. Runtime user guide. *SoC Designer*, 2016.

[3] Cisco. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 (White Paper)*. Cisco, 2017.

[4] Luke Ibbetson. Nb-iot innovation: towards a truly 'all connected world'. http://www-file.huawei.com/~/media/CORPORATE/PDF/event/hid2016/Transforming_Businesses_with_IoT_Innovation.pdf?la=en. Published: June 2016.

[5] 3GPP. Standardization of nb-iot completed. http://www.3gpp.org/news-events/3gpp-news/1785-nb_iot_complete. Accessed: 2017-02-13.

[6] 3GPP. About 3gpp home. http://www.3gpp.org/about-3gpp/about-3gpp. Accessed: 2017-02-13.

[7] 3GPP. Specifications. http://www.3gpp.org/specifications. Accessed: 2017-08-14.

[8] Keysight Technologies. Lte physical layer overview. http://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/lte/content/lte_overview.htm. Accessed: 2017-08-14.

[9] S. Landström J. Bergström E. Westberg D.Hammarwall. Nb-iot: A sustainable technology for connecting billions of devices. *Ericsson Technology Review*, 2016.

[10] The Tech-FAQ. The osi model – what it is; why it matters; why it doesn't matter. http://www.tech-faq.com/osi-model.html. Accessed: 2017-02-07.

[11] D. Raddino J. Schlienz. *Narrowband Internet of Things (White Paper)*. Rohde & Schwarz, 2016.

[12] A. F. Harvey and Data Acquisition Division Staff. Dma fundamentals on various pc platforms. *NATIONAL INSTRUMENTS, Application Note 011*, 1991.

[13] European Space Agency. Electronic system-level design methodology. `http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/Electronic_System-Level_Design_Methodology`. Accessed: 2017-05-25.

[14] National Instruments. What is a real-time operating system (rtos)? `www.ni.com/white-paper/3938/en`. Accessed: 2017-06-05.

[15] Amy Brown. *The Architecture of Open Source Applications, Volume II: Structure, Scale, and a Few More Fearless Hacks*. lulu.com, 2008.

[16] FreeRTOS. About freertos. `http://www.freertos.org/RTOS.html`. Accessed: 2017-07-13.

[17] LaTonya Pearson. The four levels of software testing. `http://www.seguetech.com/the-four-levels-of-software-testing/`. Accessed: 2017-07-11.