2nd Year Internship Report

RIOT Operating System & the Internet of things

By

MAXIME BLANLOEIL

FROM 23/06/2014 TO 15/08/2014 --- 8 WEEKS

Host Establishment Address :
HAW Hamburg University of Applied Sciences, Berliner Tor 5, 20099 Hamburg, Germany

Dedication and acknowledgements

I take this opportunity to express my profound gratitude and deep regards to my guide Prof. Dr. Thomas Schmidt for his understanding, his time and for having given me the chance to carry out my internship at HAW Hamburg.

I also take this opportunity to express a deep sense of gratitude to HAW Hamburg for having welcomed me within its establishment during these two months.

I am obliged to working group members Martin L. and Peter K. , for the valuable information provided by them in their respective fields and their help along this internship. I am grateful for their cooperation during the period of my assignment.

Table of Contents

CHAPTER 1

INTRODUCTION

This report is a description of my two months internship carried out as compulsory component of my engineering studies. The internship was carried out within the University of Applied Sciences of Hamburg, Germany, between June and August 2014. This internship aims us to discover the business/research world, its values and its functioning. It is also the opportunity to develop my own skills and to perform in a real context the knowledge I learnt in my university. At the beginning of the internship I formulated several learning goals, which I wanted to achieve:

- to see what is like to work in a research project environment;
- to see if this kind of work is a possibility for my future career;
- to use my gained skills and knowledge, and see wich ones I still need to work for a professional environment;
- to get experience in working in another country/with persons from another culture;
- to enhance my communication skills.

This internship report contains the activities that have contributed to achieve a number of my stated goals. In the following chapter a description of the University of Applied Sciences of Hamburg and the working group in which I worked are given. Then I will describe what exactly I worked on during this internship. After this a reflection on my functioning and the learning goals achieved during the internship are described. Finally I give a conclusion on the internship experience according to my learning goals.

CHAPTER 2

PRESENTATION OF THE HOST ESTABLISHMENT

2.1 Hamburg University of Applied Sciences - HAW Hamburg

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hamburg University of Applied Sciences was founded in 1970 and is the second largest university in Hamburg with over 16,000 students. Since 2007, the university has a new structure with four faculties at four different campus locations in Hamburg:

- Engineering and Computer Science
- Life Sciences
- Design, Media and Information
- Business and Social Sciences

These four faculties offer a wide range of undergraduate and postgraduate degree programmes leading to the academic qualifications "Bachelor" and "Master". The particularity of the University of Applied Sciences of Hamburg is its orientation on projects. While common universities are mainly based on theoretical lectures, HAW Hamburg favours smalls groups and interdisciplinary projects. It is in this context that I took part in one of the university projects.

2.2 The iNET Group

The Internet Technologies Group (iNET) is a working group headed by Pr. Dr. Thomas C. Schmidt in the Department of Computer Science at the Faculty of Engineering and Computer Science of HAW Hamburg. This working group performs research and development on technologies and applications for a next generation Internet and mainly on mobility, multimedia and knowledge-based systems. The iNET group currently concentrates on the following activities:

- Mobility & Multicast in IPv6, Internet Measurement & Analysis
- Contributions to Designing a Future Multiservice Internet
- Mobile Videoconferencing and Networked Multimedia
- Peer-to-Peer Networking, Overlay & Hybrid Content Distribution
- Peer-centric Hypermedia, Educational Content Management, Semantic Web

CHAPTER 3

PRESENTATION OF THE INTERNSHIP

3.1 Research and obtaining of the internship

3.1.1 Research of the internship

I had some constraints for the research of my internship. The first one was the time because as I go to Canada for my next semester I could not extend my internship until mid-september like some of my classmates; and the second constraint was imposed by my-self because I wanted to carry out my internship abroad. After sending some unsolicited applications without success, I joined the Alumni network of Grenoble INP to get some support from its members who are now abroad. Finally I was about to look for an internship in France but after all I looked for some Universities in Germany and in Spain that were interesting for me for their research groups. For this purpose I contacted Pr.Dr Thomas C. Schmidt who is the head of the iNET working group of the University of Applied Sciences of Hamburg.

3.1.2 Choice of the internship

After having exchanged several mails with Pr.Dr Thomas C. Schmidt to get some information about the projects of the iNET working group I finally chose to do my internship at the University of Applied Sciences of Hamburg for three reasons:

- **The place :** It was the opportunity to work in Germany with another culture;
- **A working group :** Even if I wanted at first to work in a company, it was the opportunity to discover the environment of a working group;

- **The Explo'RA scholarship :** As I was not paid, it was a necessity to have this scholarship. I probably could have not chosen this internship without the possibility to have this scholarship.

3.2 Context of the project

The project in which my internship takes part is focused around the development of an OS for embedded systems called RIOT [9]; this project is also around the notion of Internet of things.

3.2.1 What is Internet of things (IoT) ?

The Internet of Things (IoT) is a scenario in which objects, sensors or actuators are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. IoT has evolved from the convergence of wireless technologies, micro-electromechanical systems (MEMS) and the Internet. IPv6 's huge increase in address space is an important factor in the development of the Internet of Things.
A thing in the Internet of Things can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built-in sensors to alert the driver when tire pressure is low – or any other natural or man-made object that can be assigned an IP address and provided with the ability to transfer data over a network. So far, the Internet of Things has been most closely associated with machine-to-machine (M2M) communication and internet connectivity.

3.2.2 Presentation of the project

I worked with members of iNET working group on the development of an open-source OS designed for embedded systems, called RIOT [9], which is a platform for the particular requirements of the Internet of Things (IoT) scenarios. These requirements comprise a low memory footprint, high energy efficiency, real-time capabilities, and support for a wide range of low-power devices. Besides RIOT has three good reasons to be adopted as the new OS for the IoT:

- **Developer Friendly:** You do not work in a complex or new environment. Thus you can program in C or C++, tools such as gcc, gdb or valgrind are available, you can work both on 16-bit platforms (e.g. MSP430) and on 32-bit platforms (e.g. ARM) and you can develop under Linux or MacOS.

- **Resource Friendly:** RIOT is built on a microkernel and provides a real-time capability due to ultra-low interrupt latency($\tilde{5}0$ clock cycles) and priority-based scheduling and multi-threading with ultra-low threading overhead(<25 bytes per thread)

- **IoT Friendly:** You can make your applications ready for the smaller things in the Internet with common system support like 6LoWPAN, IPv6, RPL, TCP, and UDP, a satic and dynamic memory allocation or tools and utilities (system shell, SHA-256, Bloom filters, ...)

To finish this brief presentation, RIOT is not only developped in Hamburg but is a collaborative project with the Free University of Berlin, Germany and the INRIA Lab of Saclay, France.

3.3 Objectives

3.3.1 Different approaches

I have been proposed two kinds of activities to contribute to the development of the RIOT project:

- to work on cryptographic library with RIOT

- to work on hardware such as implementing drivers to use RIOT

3.3.2 Current status of the project

For the first activity, it has to be known that it exists some cryptographic libraries in RIOT but it was required to have a library that can use ellyptic curves and big numbers in a first step and then to import some others cryptographic utilities.
For the second activity, a lot of tasks have to be done such as implementing SPI libraries for all the boards that have SPI pins on which RIOT runs. SPI and its functionning is described in appendices.
It is also not possible to debug a RIOT project with Eclipse, for now you can just use gdb in a terminal so to make it easy for developers it would be interesting to run and debug projects via Eclipse.

3.3.3 The chosen solution

In order to make my choice I worked one week on each of these activities during the first two weeks. Finally I decided to work on the hardware part. Then I have been given several tasks:

- to permit the use of Eclipse to run/debug RIOT projects

- to implement SPI library for RIOT on the Arduino Due board

The main reasons for which I made this choice are that people with whom I work are more specialized in the hardware field so they can help me easily; then I feel less capable to work on cryptographic libraries as I am not an expert in cryptography, it is not really interesting to work on something without understanding the theory that is behind in my opinion; Finally I naturally felt more interested to work on hardware because I have knowledge in electronics and I like this melting between the physics approach of electronics and the programming approach of computer sciences that are gathered in hardware field.

3.4 Description of my work

3.4.1 Organization

As already described in the previous part, during the first two weeks I worked firstly on importing a cryptographic library into RIOT and secondly I worked on the hardware field to see in which part I wanted to work further.
For the first part, I had to configure RIOT with Relic [6], which is a library that implements ellyptic curves and big numbers. This library and its functions had to be usable from a RIOT project. The main constraint was to reduce the size of this library, because some devices only have 5kB memory !
For the second part, I had to permit the debugging of RIOT projects with Eclipse. The main goal was to permit this on remote boards like the stm32f4discovery [4] which is a ST-Microelectronics board that is very used for RIOT.
Then I worked on the hardware side and the main task I had to do was to write SPI drivers in RIOT for the Arduino Due board [2]. Firstly I set a SPI communication between two Arduino boards (Arduino Duemilanove [3]) thanks to the IDE provided by Arduino. Secondly I configured one Arduino board to communicate via SPI with a stm32f4discovery board that uses SPI on RIOT. The SPI drivers of this board were written by one of the people with whom I worked. This communication was aimed to test if his implementation was right and permited him to fix some bugs he did not know how to fix.
After these first steps to get familiar with SPI, I started to implement the SPI drivers for the Arduino Due board in RIOT like my collegue did for the stm32f4discovery one. My organization for this task can be split into three steps: reading the documentation provided by the CPU constructor of this board, implementation of my code in the RIOT repository and of a test for this code and finally checking signals and values during the SPI communication with an oscilloscope and prints in a console.

3.4.2 Implementation

3.4.2.1 Get initiated during the first two weeks

a. Importation of Relic into RIOT

To configure RIOT with Relic, build instructions are provided on Relic website for platforms supported by CMake or other platforms such as MSP430 or Arduino; but a lot of boards on which RIOT runs do not have any build instructions to install Relic. Then inspiring by the build instructions provided for these boards and the build options that are also available from Relic website, I configured this library to work with RIOT on the native board (basically RIOT is emulated in a process of Linux) and the stm32f4discovery board.

Then I had to write a "howto" tutorial[5] on the wikipage of RIOT GitHub, which is the place where developers share there issues, solutions and tutorials.

b. Configuring Eclipse to debug RIOT projects

Firstly I followed a tutorial available on the wikipage of RIOT GitHub to run and debug a project on a stm32f4discovery board using a terminal. Then with one of my collegue, we configured Eclipse to run RIOT on several boards, I did it for the native and for the stm32f4discovery. Then I had to create some tools on Eclipse for the debugging to use easily some commands that were executed manually in a console before. I faced some problems such as commands that required sudo permissions to be executed in the shell console and I did not want to create scripts to execute these commands. My goal was to make this configuration the easiest as possible so it can be set by someone who is not familiar with scripts and console. A lot of developers work with Eclipse on Windows so they are not familiar with those tools for example. Finally it is now very simple to use Eclipse for RIOT, one of my collegue works with it everyday now. This tutorial is also available on the wikipage of RIOT GitHub [8].

3.4.2.2 Set SPI communication between Arduino board & one board with RIOT

As the SPI drivers on RIOT are still experimental, to introduce myself with SPI and Arduino boards, I started to set a SPI communication between two Arduino boards. Some tutorials are availables on the Internet to understand how you can use the Arduino board as a master and as a slave.

After this first try, I got the project on which the stm32f4discovery board uses SPI from one of my collegue who implemented the SPI drivers on it. Firstly I configured my communication to have the stm32f4discovery board as a master and the Arduino Duemilanove board as a slave. I needed to configure them on the same way: the same clock polarity, the same clock phase. Then I also needed to run the master board at the slowest speed as possible, this is possible by changing the clock divider value. After this I tested my transmisson with an example that I already did with the two Arduino boards. This example consists in sending 8 bytes consecutively with differents values from master to slave. I faced several problems to receive the right value on

the slave because I did not know if my problem was due to a wrong pin, a wrong type of variable in my program or another reason. Then I wanted to reply to the master board and I faced again several problems that are still not really solved. I found a solution that delays the next interrupt and this permits to answer to the master but it is not very efficient. It should be more generic. Finally I was able to have a transmission between my two boards, with the stm32f4discovery as a master and the Arduino Duemilanove as a slave. The reply could be however enhanced to be more generic.
Secondly I configured my SPI transmission in the opposite way: the Arduino Duemilanove as a master and the stm32f4discovery as a slave. It was a good test to check if the stm32discovery with RIOT could receive data from another board than a RIOT board which is specifically designed to work with it. As in the previous part, I started to send data from master to slave, without replying. For doing this, I computed a program that is very similar to some commands used as a master board in RIOT because I wanted to avoid problems caused by a program mistake, so my program just sent 8 bytes to the stm32f4discovery board. I had some electrical problems at this point because of some pins that are unsteady. Then I managed to reply by editing the RIOT drivers that were basically unable to answer anything more than a constant. I had again some electrical issues, it seems like I lost some bits. The most probable reason is that the Arduino clock was too fast for the STM one, and the STM board was not enable to catch each bit.
To conclude this SPI communication, I managed to communicate between an Arduino Duemilanove board and a stm32f4discovery board that runs on RIOT. This communication can be established as a master or as a slave for both boards. Yet I faced some issues, I was obliged to add some delays to reply when the Arduino board was slave and I also had some data loss probably caused by electrical instability.

3.4.2.3 Implementation of SPI drivers for RIOT on the Arduino Due board

To work with the Arduino Due board, I needed to get some documentation about this board. This documentation is provided by Atmel which is the CPU constructor for this board [1]. As the documentation is about 1500 pages, I focused on the SPI part to begin. This part provides a description of the SPI circuit, the embedded characteristics, the block diagrams, the product dependencies, the functionnal description and the description of each register involved in the SPI circuit.
For each board and each cpu compatible with RIOT, a directory exists in the RIOT repository. Thus there is a directory for the Arduino-due board and there is a directory for the Arduino-due CPU which is SAM3X8E. All the information provided by Atmel for the CPU are available in the SAM3X8E directory. My first step was to gather the constants and the values of the CPU that were about SPI in a header file in the Arduino-due board directory named "periph_conf.h". This values are given to this header file through a serie of define. Three different types of value were necessary to implement, according to the "Product dependencies" part of the SPI documentation

that describes three different interfaces to configure:

- Parallel Input/Output Controller (PIO)
- Power Management
- Interrupt

Some pins on the board need to be configured to work as SPI pins. The controllers that manage pins on a board are the PIO controllers, so I needed to configure the right PIO controller to assign the SPI pins to their peripheral function. The SPI needs to be clocked through the PMC, thus I had to configure the PMC to enable the SPI clock. And the SPI interface has also an interrupt line connected to the Interrupt Controller, so I needed to program the interrupt controller before configuring the SPI.
Firstly I looked into the files related to these controllers to find the value that were concerning the SPI. Then I could complete the define and the macros in the header file "periph_conf.h".
Secondly I had to implement functions that enable the SPI communication in a file named "spi.c" in the SAM3X8E directory. The prototypes are already provided in a file named "spi.h", they are the same for each CPU. These functions are for example "spi_poweron", "init_spi_master", "init_spi_slave" or "transfer_byte".

- spi_poweron: this function gathers the macros that are written in the "periph_conf.h" file, it enables the SPI clock and the PIO clock.
- init_spi_master: this function writes in several registers that can be split in two parts: the PIO registers and the SPI registers. This consists in setting or clearing bits in these registers to enable or disable some functionalities. For example, for the PIO registers, to enable pins to work as SPI pins and not classical I/O pins, I needed to set some bits. Each bit in the 32-bit register matches with a pin on the board, then thanks to the values I defined in the "periph_conf.h", I could set the bits corresponding to the SPI pins with bit operations like OR, AND, NOR or NAND. For the SPI registers, with the same bit operations, I initialized the polarity, the phase, the speed, the number of bits for a transmission or the delay between two consecutive transmissions.
- init_spi_slave: It is not necessary to repeat everything, because a lot are the same than for the init_spi_master function. The difference are that you do not initialize speed, polarity and so on for the slave as the master imposes its settings during a communication; and for the slave you have to enable interrupts and its priority level, because the slave is waiting for interrupts for the reading and writting operations.
- transfer_byte: For this function, three registers are involved: the SPI status register, the SPI transmit data register and the SPI receive data register. The master executes this

function to exchange data with the slave, so before transmitting data it must check in the status register if the bit "TDRE: Transmit Data Register Empty" is set, this avoids to overwrite a current transmission. Then the master can write in the Transmit Data Register, the data to be transmitted is stored in this register. This is then automatically and immediately transferred on the SPI Bus. Then before reading the received data, in the same way, the master must check in the satus register if the bit "RDRF: Receive Data Register Full" is set. When this bit is set, the content of the Receive Data Register is returned by the function.

Thirdly I had to write a test to show the other people how to use the functions of my "spi.c" file. This test is actually a RIOT project where you have a "main.c" file and a Makefile. When you execute your project on the Arduino-Due, a shell is provided where you have several commands available. Thus I implemented some commands that use the SPI functions. Basically I have three functions: init_master, init_slave and send_data. To execute this test project you need two boards: you run RIOT on the two boards via two consoles in Linux, then you initialize one as a master and the other one as a slave. After this, you can use the send_data command from the master. In this function I sent several bytes, and I printed the transmitted data and the received data so I could check what the master exactly sent and what it exactly received. For each interrupts detected by the slave, I also printed the received data to compare with what was sent by the master.

3.5 Results & Analysis

3.5.1 Evaluation of my results

3.5.1.1 Technical evaluation

Excepted for configuring a software like Eclipse, my main tasks are focused on programming in a RIOT project. For this reason, I work in a very robust environment, indeed the RIOT Makefile organization does not pass over any errors. Whether this is a new file I code or the integration of a library like Relic, where the work is about configuring with different compiler options, when you compile your project, you already see your errors and warnings. This is very helpful because then, when your project is build, you can be barely sure that it works.
Some other tasks can also be tested quite efficiently like SPI communication between two boards for example. Indeed it is quite easy to check if the communication works or not by printing the sent and the received value on each board. Then you just need to verify if these values are the same. Nevertheless you must be careful because as these SPI communication are very fast, you need to store each bit sent before printing all of them. The print functions are sometimes too expansive in time to be executed between two SPI interrupts. And as printing the sent and the received values are not enough to be sure everything works, it can also be interesting to check

the registers value.
Therefore a task like configuring Eclipse to debug and run projects cannot be evaluated only by a compiler because Eclipse is sometimes unsteady depending on your version (e.g. Eclipse has a lot of different versions), your operating system, your compiler or your version of RIOT (a lot of branches are available). That is why, it is hard to be sure if it works for everyone and that is why I had to evaluate my work by other people.

3.5.1.2 Human evaluation

As I work with people who have been in this project for a long time, I can always ask them their opinion on my results because they often had the same problems or wrong results I get. That is very helpful to move forward when I do not understand my results. Then when my task is finished, I need to write a tutorial to explain how I did it because some other people could be interested to do the same. This step requires me to think over what I exactly did and to make it clear. When my tutorial is written, I follow it from the begining as if I did not know anything to check if nothing has been forgotten and finally I ask one of my collegue to do it again on his side to be sure everything works for another people.

3.5.2 Validation for the projects

My results are considered as validated when they matches with what we expected and when they are the same for my collegues. This is the first step. After that, I push my work on my branch in my own repository on GitHub. Then If I want my work to be integrated into the master branch of the RIOT repository, I need to create a Pull Request. This Pull Request is visible for everyone who is working on RIOT, so people who are concerned by my work comment on this Pull Request if I made some mistakes. These mistakes can be really important for the stability of RIOT or they can just be about coding style. When everything is validated we push my work on the master branch . Thus it is really important my work is previously validated.

3.5.3 Outlook

As I subscribed the developers mailing list, I can see if any reactions appear concerning what I did. Indeed some people sometimes reacted and had issues when they did what I wrote in my tutorials. Even if it was checked by my collegues and myself, it was not working for others so after discussing with them I enhanced, cleared or changed my tutorials to make them the most accurate as possible. This mailing list is effectively very interesting to share our reactions on other people work. You can ask help for everything and it keeps the community around RIOT very active.

CHAPTER 4

REFLECTION ON THE INTERNSHIP

4.1 Main issues

4.1.1 Description of the main issues

During the different tasks I achieved, I faced several issues:

- Importation of Relic into RIOT: This task was mainly about configuring Makefiles and CMake files to make Relic and RIOT compatibles. That is why serious knowledge for configuring such files was necessary. It is obvious that I had a lack of knowledge for this but it has been a real good way to learn how to handle flags and linkers.

- Configuring Eclipse to debug RIOT projects: As I wanted to make this the easiest as possible for people who are not used to work with a console and scripts I faced some difficulties. Indeed Eclipse tools do not permit to execute consecutive commands and I required sudo permissions for several commands.

- SPI communication between a board with RIOT & an Arduino board with the Arduino IDE: Some transmissions were unsteady and may be due to CPU speed between the two boards, some interrupts occured before the last one had enough time to finish its routine.

- Implementation of SPI drivers for RIOT: The main difficulty was that writting such drivers was unknown for me. I was not sure how to check my work step by step. Because the SPI transmissions are really fast, the use of a debugger like GDB is not possible because I should have launched two debuggers for the two boards and it is a nonsense to stop interrupts during the SPI transmissions. Thus my main preoccupation when my test did not work was knowing where the error comes from.

4.1.2 Solution

This part provides the solution I chose to solve the issues that have been mentionned in the previous part:

- Importation of Relic into RIOT: With one of my collegue help who works on the Relic library and who is very comfortable with the use of Cmake files, especially in RIOT, I managed to use the right flags and the right linkers to configure Relic with the architecture of the stm32f4discovery board. I am very grateful for his help because, beyond it permitted me to achieve my work, I learned a lot.

- Configuring Eclipse to debug RIOT projects: As it is not possible to execute several commands with Eclipse tools, especially "sudo", I only executed the "sudo" command but instead of adding the other command then like in a console, I put the command that requires sudo permissions into the "argument" field. Then a new problem came: the user needs to enter his sudo password to enable the execution of such commands so I had to edit the configuration file "/etc/sudo.conf". This permitted to display a window dynamically that asks you your sudo password. This tool made it really easy to work with Eclipse now, without using a console or scripts.

- SPI communication between a board with RIOT & an Arduino board with the Arduino IDE: For this issue, the solution is still not found. We have some ideas why it does not work, like the difference of CPU speed, but we did not find how to fix this issues.

- Implementation of SPI drivers for RIOT: My problem was not a difference between what I sent from a board with what I received from the other one, I could not just enable the communication. As I did not know where come from my errors, I started to simply print the registers content and checking the behaviour of each signal with an oscilloscope. Then I only read again and again the documentation for the board to check if I have not forgotten anything, and sometimes I found some mistakes. For example the SPI clock was not correctly enabled with the system clock. Finally when the first communication has been possible, the data transmitted between the two boards were available so it was much easier to debug when I had some other problems.

4.2 Balance sheet on my personal goals

As I formulated several learning goals, I made a balance sheet for them at the end of my internship in HAW. Firstly I wanted to see what is like to work in a working group and if it could be a possibility for my career. Then it was an exciting experience, I learned a lot, and I realized that people who work on such projects are really passionate by what they do. Most people work much more than what they are supposed to be because they just feel involved by their project. The

atmosphere was consequently really pleasant, thanks to the mailing list everyone is ready to help you if you encounter some difficulties. To compare with the work in a company where it is more individualist, probably due to the competition that sometimes exists between employees, I felt here a real support from the community that works on the RIOT project. Thus even if it is economically more reasonable to work in a company for a long-term job, the idea of working on such a project reminds in the back of my mind.

Secondly I wanted to use my skills in a concrete project. At the beginning I was affraid of being useless because the project was already well under way but I realized that even if I was not used to work with embedded systems I was able to adapt my skills to this project. That is also why doing an internship is very interesting, it permits to highlight my knowledge and my skills on a lot of things that are possile to do even if they are not a direct application of what I studied. Nevertheless I learned a lot to be able to do everything I did during this internship and that was also one of my objectives.

Finally I wanted to work in another country with another culture and to improve my English language as I do not speak German. And while the two previous parts were directly related to my work, this one was also affected by my free time. Thus it was very exciting to discover Germany and the German culture. The way of work is quite similar with the French one excepted for the lunch time which is shorter than in France where we take a (too?) long lunch break. Thanks to the internship I have been practicing English for two months, and I think I improved my English which bodes well for my next Canadian semester. It was also the opportunity to discover the German daily life and to step back from the French one, thus a lot of events happened in Hamburg like of course the great passion of German people for their national team during the World Cup where a lot a giant screens were spread in Hamburg. I also had the opportunity to visit Berlin for a week-end, which is a historical city. This allowed me to better understand the German culture, that was something I expected from these two months in Germany, and even if it was not directly related to my internship I think it was important to also have this immersion.

CHAPTER 5

CONCLUSION

This internship allowed me to see what is like to work as an engineer and what is a working group, during two months. I appreciated to use my skills and my knowledge in a concrete project. I was pleasantly surprised to see that even if this project was not a direct application of what I am studying, the tools I used and the structure of the RIOT project were familiar to me because I already knew them. This was as a consequence quite easy and quick to enhance my skills in order to be able to work on the tasks they gave me. As I learned a lot of things, I tried to bring the more I could to the project, to do my best and to finish every task they gave me. Finally this was a grateful experience because I feel that I contributed to the project and I managed to achieve the main tasks and in the same time, this internship brought me a lot too. It is also frustrating to work on a project during only two months, and I think I will keep an eye on the RIOT project because it is a very young project and I hope I will hear about RIOT in the next years with the increasing development of the Internet of Things in our lives.

APPENDIX A

A BRIEF INTRODUCTION TO THE SERIAL PERIPHERAL INTERFACE

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers (this was typically the way I used the SPI during my internship).
With a SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:

- **MISO** (Master In Slave Out) - The Slave line for sending data to the master,
- **MOSI** (Master Out Slave In) - The Master line for sending data to the peripherals,
- **SCK** (Serial Clock) - The clock pulses which synchronize data transmission generated by the master and one line specific for every device:
- **SS** (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.

When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.

The next two figures give a visual interpretation of a communication between master and one or several slave(s).

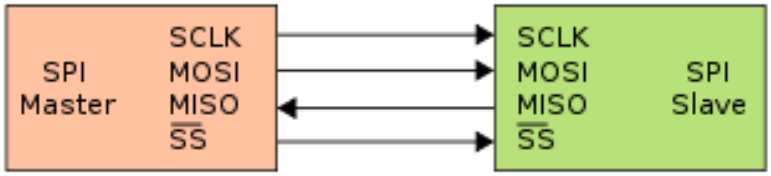

FIGURE A.1. SPI bus: single master and single slave [7].

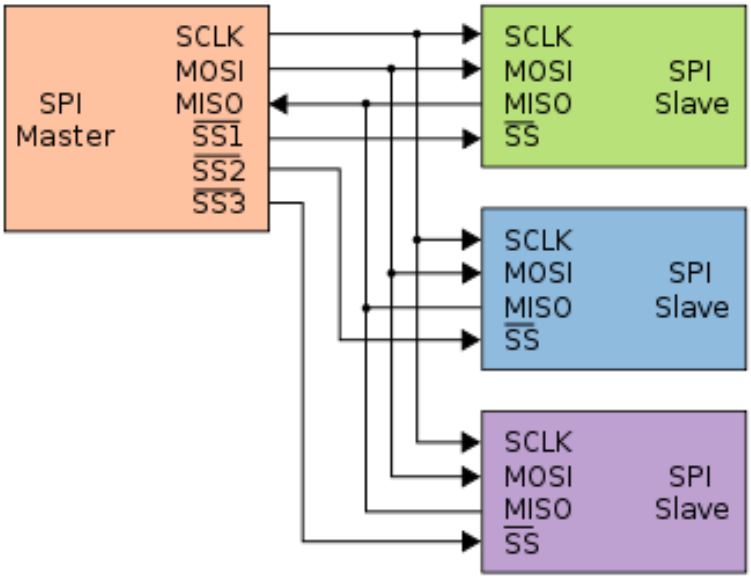

FIGURE A.2. Typical SPI bus: master and three independent slaves [7].

Generally speaking, there are four modes of transmission. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal (called the clock phase), and whether the clock is idle when high or low (called the clock polarity). The four modes combine polarity and phase according to this table:

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

References

[1] *Atmel. Arduino Due: Datasheet and Reference Manual*, http://www.atmel.com/Images/doc11057.pdf.

[2] *The description page for Arduino-Due from Arduino*, http://arduino.cc/en/Main/arduinoBoardDue.

[3] *The description page for Arduino-Duemilanove from Arduino*, http://arduino.cc/en/Main/arduinoBoardDuemilanove.

[4] *The description page for the Stm32f4discovery from ST*, http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419?sc=internet/evalboard/product/252419.jsp.

[5] *Intregate Relic (a library for cryptography) into RIOT*, https://github.com/RIOT-OS/RIOT/wiki/Intregate-Relic-(a-library-for-cryptography)-into-RIOT.

[6] *RELIC is an Efficient LIbrary for Cryptography*, https://code.google.com/p/relic-toolkit/.

[7] *Serial Peripheral Interface Bus - Wikipedia*, http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

[8] *Using the Eclipse IDE for C and CPP Developers, Howto*, https://github.com/RIOT-OS/RIOT/wiki/Using-the-Eclipse-IDE-for-C-and-CPP-Developers

[9] *RIOT, The friendly Operating System for the Internet of Things*, http://www.riot-os.org/, 2013-2014.