UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA

University of Padua

Mathematics Department "Tullio Levi Civita"

Degree in *Computer Science*

Improving product development of an automotive interior lighting application using SysML

Thesis supervisor:
Prof. Ombretta Gaggi

Student:
Marta Greggio
matr. 1222777

A.A. 2021/2022

Ai miei genitori che da sempre mi supportano negli studi,
Alla prof. Gaggi, relatrice di questa tesi, che mi ha seguita e sostenuta durante il percorso,
Ad Andrea e Christopher,
A Giulia, a Simone, ad Elia,
A tutti coloro che con un piccolo o un grande gesto hanno reso speciali questi anni.

Abstract

The following document describes the internship experience made by Marta Greggio, at *Infineon Technologies Italia S.r.l* based in Padova (PD).
It has been done at the end of the Bachelor's degree course in Computer Science and it lasted about 300 hours.
The internship is about improving the product development using SysML to model typical use cases. The document illustrates the company itself, the carried out activities and an evaluation of the work performed as well.

Contents

List of Figures

1 Introduction

1.1 The company

Infineon Technologies AG (figure 1) is a German semiconductor manufacturer founded in 1999 with headquarter in Munich and different operational sites in Europe, Middle East, Africa and America [3].
It is effectively one of the leading players regarding semiconductor solutions in different markets such as automotive, industrial, multi-market sectors, chip cards and security products as well.
Infineon Technologies Italia S.r.l. is a legally independent subsidiary with its headquarters located in Milano and it is fully owned by *Infineon Technologies* AG in Germany. The biggest Research and Development centre is located in Padova where almost 200 engineers and researchers develop products mainly for the Automotive industry and in particular drivers for conventional and LED lighting.

Figure 1: *Infineon Technologies*' logo

1.2 Company divisions

The company is comprised of four different divisions [2]:

- **Automotive**: this division aims to shape the future of mobility enabling clean, safe and smart cars. Semiconductors are essential for supporting trends like electromobility and autonomous driving;
- **Industrial Power Control**: power semiconductors play an important role for increasing efficiency and reducing energy losses. This division works for example on components for solar energy systems as well as power supplies or home appliances;
- **Power & Sensor Systems**: this division focuses on power management and data transfer capabilities. *Infineon*'s semiconductors enable intelligent smart sensitivity and fast data processing in steadily increasing digitized world;
- **Connected Secure Systems**: this division is focusing on the heart of the Internet of Things. IoT is having a huge impact on our lives and *Infineon* provides solutions for computing, wireless connectivity and trusted technologies in order to help to securely connect the networked systems.

1.3 Way of working

1.3.1 V-model

The company organizes the development of a product following the V-model (see figure 2). The V-model is a product development life cycle methodology that describes the activities to be performed and the results that have to be produced. It is also known as verification and validation model. It consists of four different phases:

- **Requirement phase**: requirements contain the essential information regarding functionality and performance of a product;
- **Architectural design phase**: it is referred to as high-level design in which all the information from the requirements phase is split into easy to handle modules, the functionality of each module is defined, as well as its relationships, dependencies, architecture diagrams and technology design;
- **Detailed design phase**: it is referred to as low-level design in which for each module a model fulfilling the required functionality is created;
- **Implementation**: final phase that consists in realizing the product.

The approach of the V-model helps to avoid defects in later stages of development and allows testers to have the test cases ready when required.
Moreover, testing the product during every stage of the development helps to deliver a high quality product. The model is considered to be quite strict since it lacks flexibility when it comes to making even slight changes: this prevents changes to be made without proper alignment.

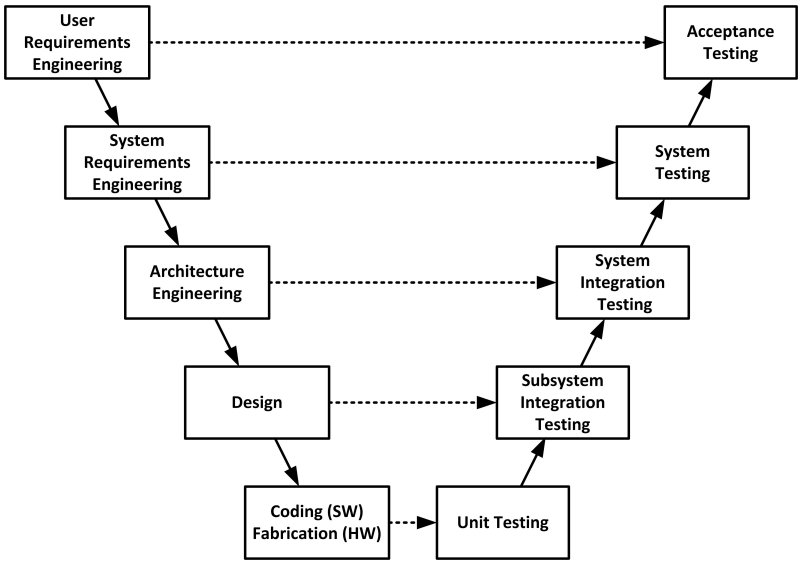
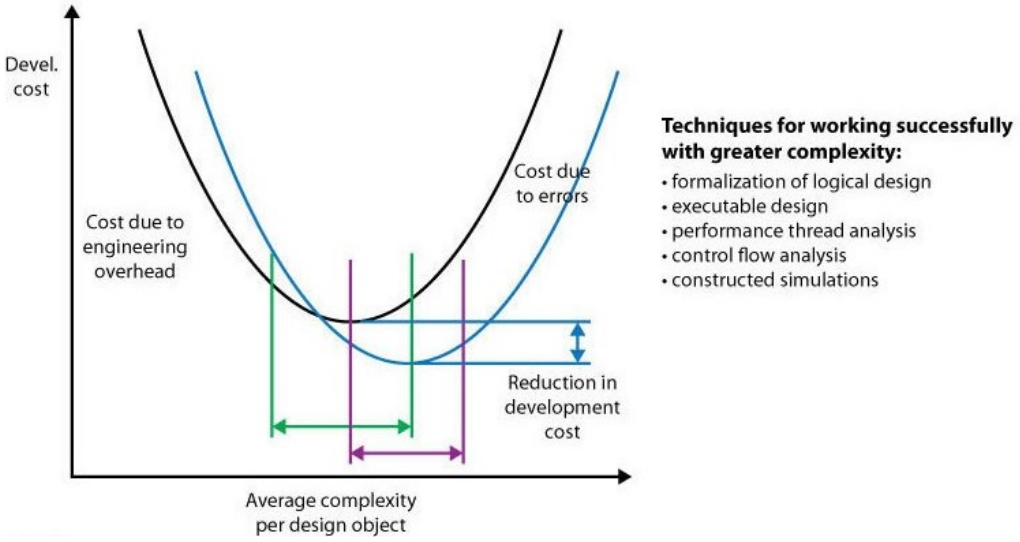


Figure 2: V-model diagram

1.3.2 MBSE

After encountering some difficulties, the company decided to study a new paradigm in order to design products more efficiently, reducing development costs and timescales. Risk due to technology is also reduced, because of a greater focus on formalizing how the technology works.
Model Based System Engineering (MBSE) [8] is a methodology used to support the development of complex systems. MBSE, applied in the right circumstances, allows designers to work successfully at greater degrees of complexity, as illustrated in figure 3.
MBSE brings together three concepts:

- **Model**: a simplified version of something that abstracts reality to eliminate some complexity;
- **Systems thinking**: way of looking at a system under consideration not as a self-sufficient entity but as a part of a larger system;
- **System engineering**: integrative approach to enable the successful realization, use and withdrawal of engineered systems using system principles and concepts.



Source: Benefits of Model-Based System Engineering [1]
Figure 3: Benefits of MBSE

1.4 Race UP Team

From an early age on I always watched Formula 1 with my dad and since then I developed more and more passion for motorsport.
In September 2020 I applied to the Formula SAE_G Team of the University of Padua, the Race UP Team. Formula Student is a competition where teams from different universities design and realize a formula-style car.
The University of Padua's team actually has two divisions, building both an electric car and one with a combustion engine. I am currently head of the software department of the electric division.
As a proper team, we are also responsible for finding sponsors willing to help us reach our goals. *Infineon Technologies*, who is one of our sponsors, offered me an internship, which I was honored to accept.
Even if a semiconductor manufacturer didn't seem to fit well with my bachelor's career, I realized I could still apply concepts learned in class and I thought it would be a great chance to extend my knowledge.
Moreover, since *Infineon* is one of the leading companies in the automotive market, this experience helped me understand what it is like to work in such a large and complex environment.

1.5 Thesis structure

The thesis is divided into four chapters. The **first chapter** describes the company, *Infineon Technologies*, starting from its history to the products and the services it offers.
The **second chapter** motivates the internship experience, the choice of the company and highlights the advantages of an internship for both the student and the company itself.
The **third chapter** focuses on the activities of the stage, starting from the study of the current trends in the automotive field, to the realization of the use cases and the activity diagrams of the product to the "hands on" part where I interacted with an Infineon product and made use of some offered features.
Last but not least, the **fourth chapter** contains a short review of the experience from my point of view, concerning the acquired knowledge and an evaluation of the achieved goals.

1.6 Typographic conventions

Acronyms, abbreviations and ambiguous words are defined in Section 5 and are identified by a subscript G.

2 Internship proposal

2.1 Project presentation

During the internship experience I worked in the Technical Marketing and Software team. My job there was to understand products *Infineon* is working on in order to detect use cases and have a clear idea on how the customer will interact with the final products.
In fact, with *Infineon* being the unchallenged leader in the automotive market, it is very important to consider every scenario and even any failures in the product during development in order to provide high quality products, having considered the strengths and detecting and reacting to weaknesses at an early stage. My work was also useful for the company itself: since *Infineon* is a very big company, even if people work on the same product, they all operates in different stages of the development. For example, the different teams are mainly working within their own scope on the product and, depending on the product complexity, may only have an abstracted view of the full product.

2.2 Introduction

2.2.1 Interior lighting

"Interior lighting" refers to all the lighting devices inside of a car that provide comfort and safety for the driver and passengers, such as footwell lights (see figure 4 at page 10), interior decorations, as shown in figure 5 at page 10, door panel illumination, light guides and also assist in alerting the driver, for example by indicating obstacle proximity or unfastened seat belts, to name a few examples.
The developments in the LED lighting industry affect topics like:

- Brand identification;
- Comfort;
- Safety;
- User experience;
- Communication and information.

Moreover, in the past years, expecially due to global warming and the more and more strict regulations regarding the environmental protection, the focus has moved to electric cars.
This big change was a great opportunity to re-design the interior of the car, not only the architecture but also the comfort and the interior lighting itself.
In addition, if we consider the general transformation of the automotive field towards autonomous vehicles, we notice how important it is to improve the user experience. This could be done for example by trying to recreate the comfort of the living room, and what is a better way to do so than with lights?

Source: Interior lighting report [7]
Figure 4: Example of footwell lights

Source: Interior lighting report [7]
Figure 5: Example of interior lighting

2.2.2 Application Engineer

An Application Engineer (AE) is the application and product expert.
AE works throughout the product development life cycle in supporting application design, analysis, development and testing processes, he coordinates and helps in product development by focusing on the application from both user and technical standpoints.
He understands all the features, the advantages of each product, how the product is commonly employed, and how to troubleshoot common problems. An application engineer is asked to confirm specifications of the product, standards and changes and to solve technical problems the consumer may have using it. In particular at Infineon there are different kinds of Application Engineers, such as Product AE and System AE.

3 Internship experience

3.1 Technology requirements

3.1.1 Enterprise Architect

Enterprise Architect (figure 6) is a visual modeling and design tool developed by Sparx Systems based on the OMG_G UML_G .
The platform supports different operations such as design and construction of software systems, modeling business processes and modeling industry based domains.

Figure 6: Enterprise Architect's logo

3.1.2 SysML

The System Modeling Language (SysML) (Figure 7) is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include software but also hardware, information, procedures and facilities [9].
SysML is defined as a subset of UML with extensions needed to satisfy system requirements.

Figure 7: SysML's logo

Comparison with UML

- SysML reduces some of the software-centric restrictions in UML and thus SysML diagrams are more flexible and expressive;
- SysML can model a wider range of systems, compared to UML being designed only for software development;
- SysML is easier to learn and apply, since it is a smaller language with fewer diagram types;
- SysML adds two diagram types: "Requirements" and "Parametric".

An overview of the different types of diagrams offered by both SysML and UML can be seen in Figure 8.

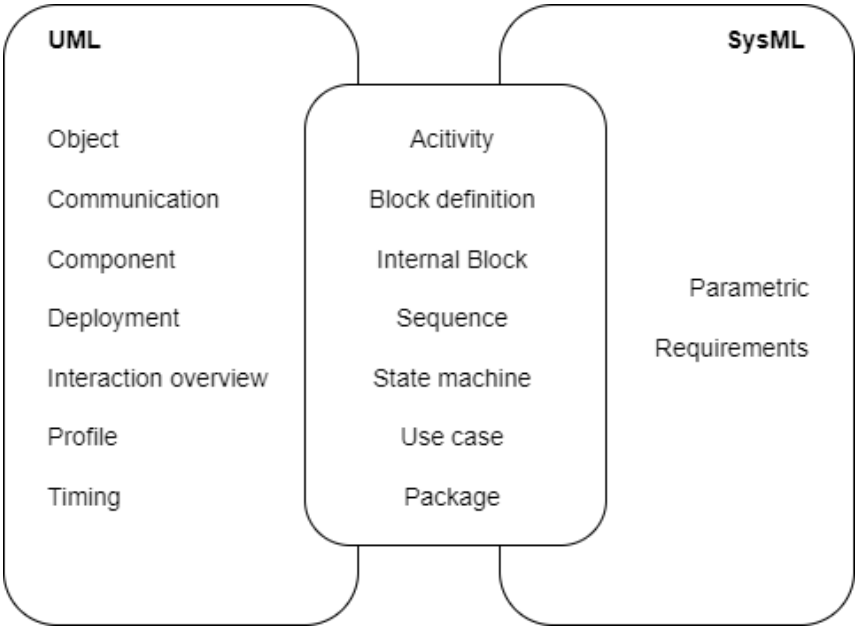


Figure 8: Differences between SysML and UML diagrams

3.1.3 Python

Python (Figure 9) is an "interpreted, object-oriented, high-level programming language with dynamic semantics", as described in the official website [11].
It provides high-level built in data structures, combined with dynamic typing and dynamic binding and easy to learn syntax that emphasizes readability and therefore reduces the cost of program maintenance.
Moreover, supporting modules and packages, it encourages program modularity and code reuse. Extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Figure 9: Python's logo

3.2 Use case diagrams

In software and systems engineering the use case is a list of actions and events that define interactions between an actor and a system, in order to achieve a goal.
Use case diagram is a behavioral diagram in UML and it is a visual representation of a use case scenario.
These diagrams are used to model the functionality of a system, to describe its functional requirements and to present a simple but compelling picture of how the system will be used.
Before starting to create the use case diagrams in Enterprise Architect, I found it helpful to summarize some key concepts in a tabular structure.

For each use case I could think of, I followed a company template table in order to write the use case specification. In fact, even if the diagram helps creating a visual representation of the interactions, it is also important to have a textual or tabular description that includes additional information, such as pre-conditions, post-conditions, ecc. An example of the template is shown in Table 1. Moreover, I also reported the sequence of events in another table, as shown in Table 2

UC X.X	**Change $\mathbf{BSL_G}$ windows time**
Type	System
Actor	User
Result	User has configured the communication window with the BSL
Pre condition	BSL interface is activated and BSL is enabled
Post condition	BSL has a different configuration for the time window

Table 1: Example of use case table template

Use case scenario	
Basic Path	1. User turns the device on 2. User establishes a connection with the device 3. User writes a number between 1 and 28 as the NAC_G value on the register 4. User performs a cold or warm restart on the device 5. BSL time window has changed and set to the NAC value $\times 5ms$

Alternate path	1. User turns the device on 2. User establishes a connection with the device 3. User writes a number bigger than 28 that is the NAC value on the register 4. User performs a cold or warm restart on the device 5. BSL time window has changed and set to forever

Table 2: Example of use case scenario table template

After that, thanks to an Enterprise Architect feature I was able to import all the tables inside each use case, so that every step and a brief description could be included inside the diagrams.

3.2.1 Identified actors

I identified three actors:

- Customer;
- Technician in the garage;
- Test Engineer.

The actor I focused on more is the customer, who is mainly a software developer who interacts with the device. There are three ways he can do so (see Figure 10):

- via interface, both BSL and SWD_G (UC1);
- via the API provided in the header file (UC10);
- via startup service routine (UC11).

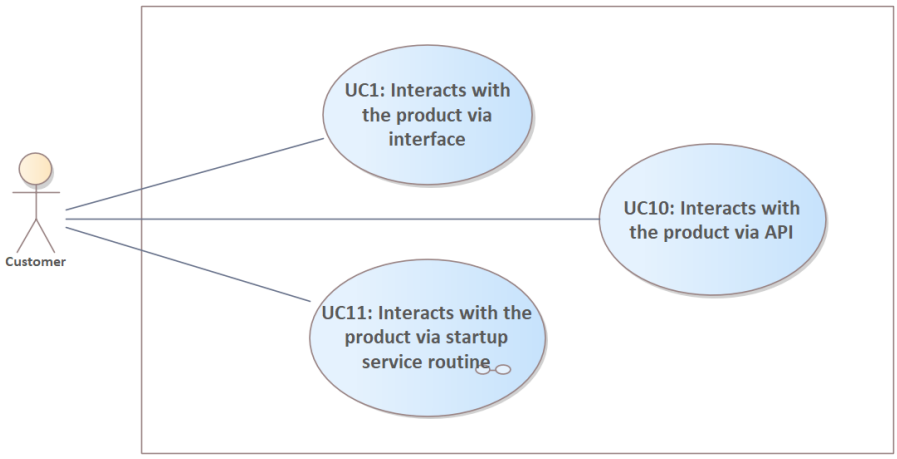


Figure 10: Different ways to interact with the device

Depending on the interaction, customers can perform different actions. The previous diagram (Figure 10) can then be extended as the following (Figure 11):

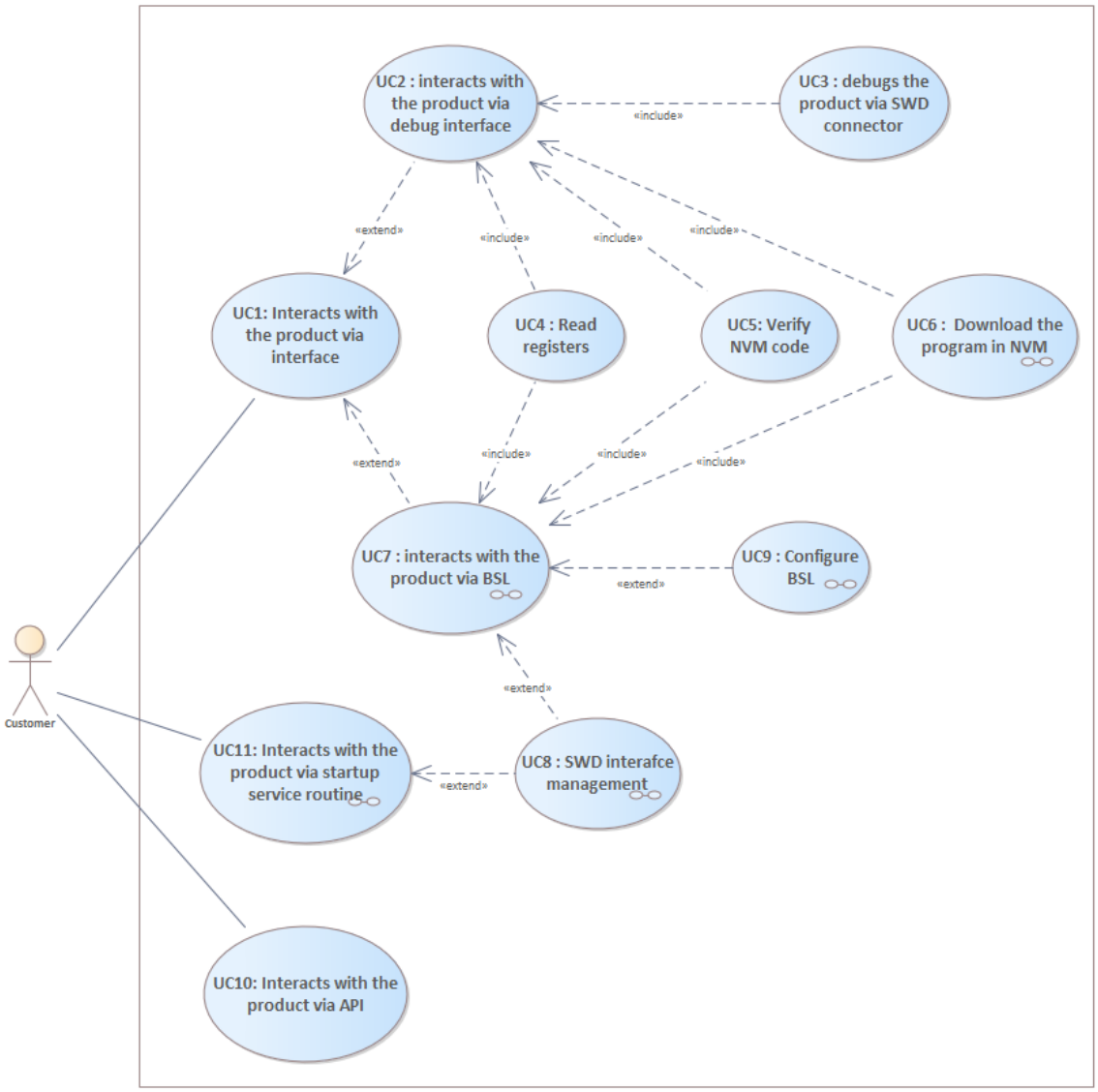


Figure 11: Extended UC diagram

Then I could also extend the *UC8 - SWD Interface management* and the *UC9 - Configure BSL* diagrams (Figures 12 and 13 at page 16).

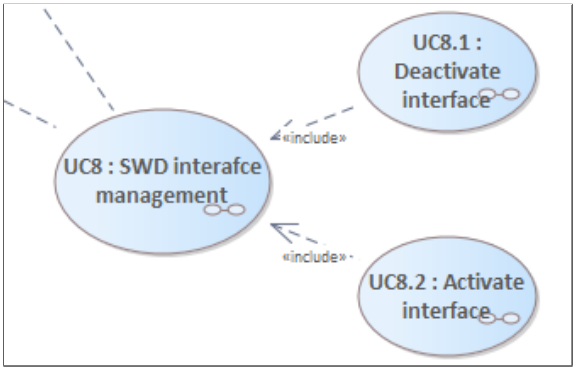


Figure 12: Sub-use cases of UC8 - SWD interface management

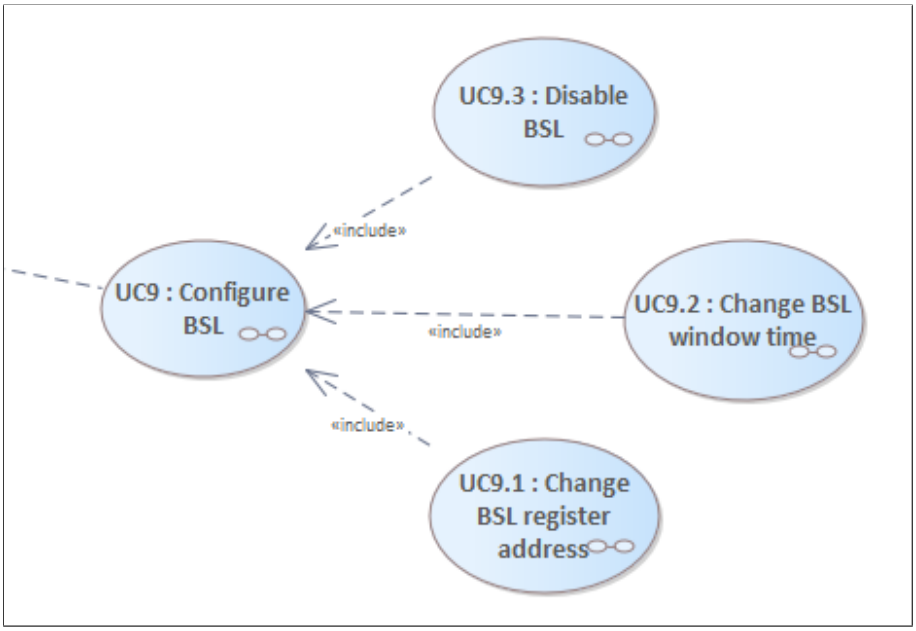


Figure 13: Sub-use cases of UC9 - Configure BSL

3.2.2 BSL interaction

Even if the use you could make of these diagrams is wide, I focused on the *UC1 - Interaction via interface* and its extensions, expecially the interaction via BSL.
BSL stands for Bootstrap Loader and it's a communication protocol over the serial LIN_{G} interface. LIN stands for Local Interconnect Network and it is a serial network protocol used for low-speed communication between components in vehicles. More details will be provided later in this thesis in section 3.4.

3.3 Activity diagrams

An activity diagram is a UML behavioral diagram that provides a dynamic view of the system and highlights the sequence of actions that describes the behaviour of a structural element.
Common usage of an activity diagram in SysML includes creating graphical use case specifications. Having considered what could be the usual steps the user has to make to perform some kind of action, it was easy for me to create the activity diagrams in SysML.

To describe the flow of events already identified in the tabular description of the use cases, I also created some activity diagrams.
Some use cases did not require a more detailed view, so I will only report in this thesis the most relevant ones.

UC6 - Download the program in the NVM_G

One of the operations the user can perform is to download his code in the NVM of the device.
The Non-volatile memory (NVM) is a type of memory that can retain stored information even after power is removed. The user code must be downloaded in the NVM in order to allow the device to execute it.
This action can be performed in two different ways: via SWD debugger and via BSL protocol.
In Figure 14 at page 18 you can see the corresponding activity diagram.

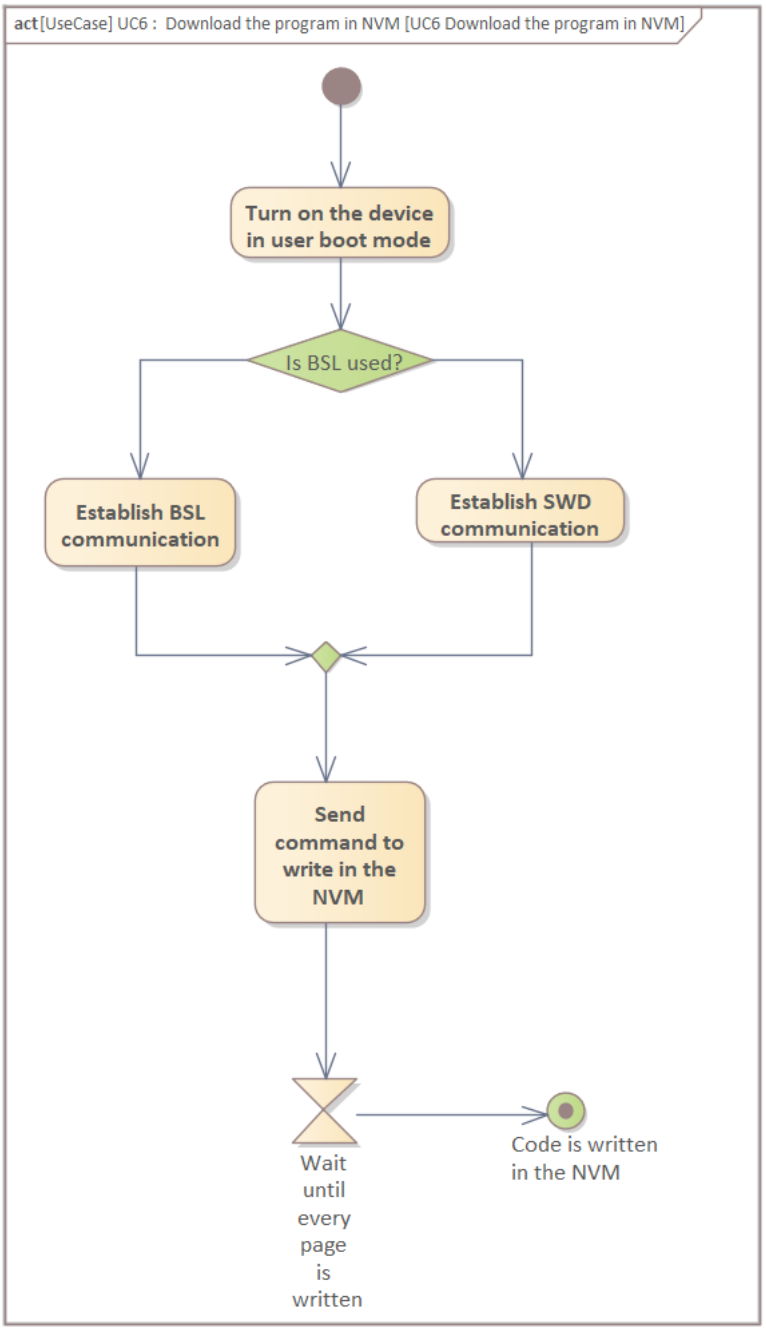


Figure 14: Activity diagram for the UC6 - Download the program in the NVM

UC7 - Interacts with the product via BSL

This use case extends the less detailed use case *UC1 - Interacts with the product via interface*. For interface we mean both the SWD debugger and the BSL.
Through BSL, for example,the user can:

- Read registers (UC4);

- Verify NVM code (UC5);
- Download the program in the NVM (UC6);
- SWD interface management (UC8);
- Configure BSL (UC9).

In order to use the BSL, the relative interface must be activated.
In Figure 15 at page 19 you can see the corresponding activity diagram.

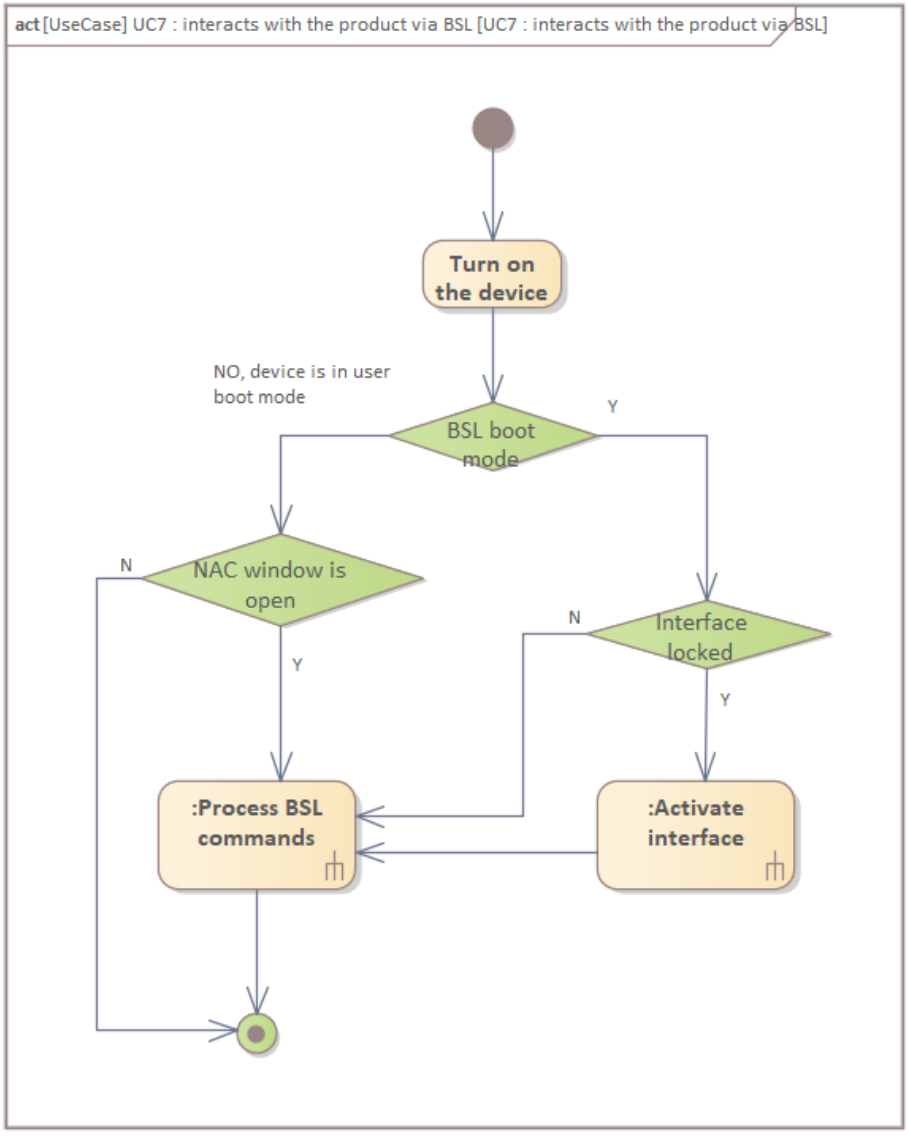


Figure 15: Activity diagram for the UC7 - Interacts with the product via BSL

In case the NAC window is open, the device processes the relative command as shown in Figure 16 at page 20.

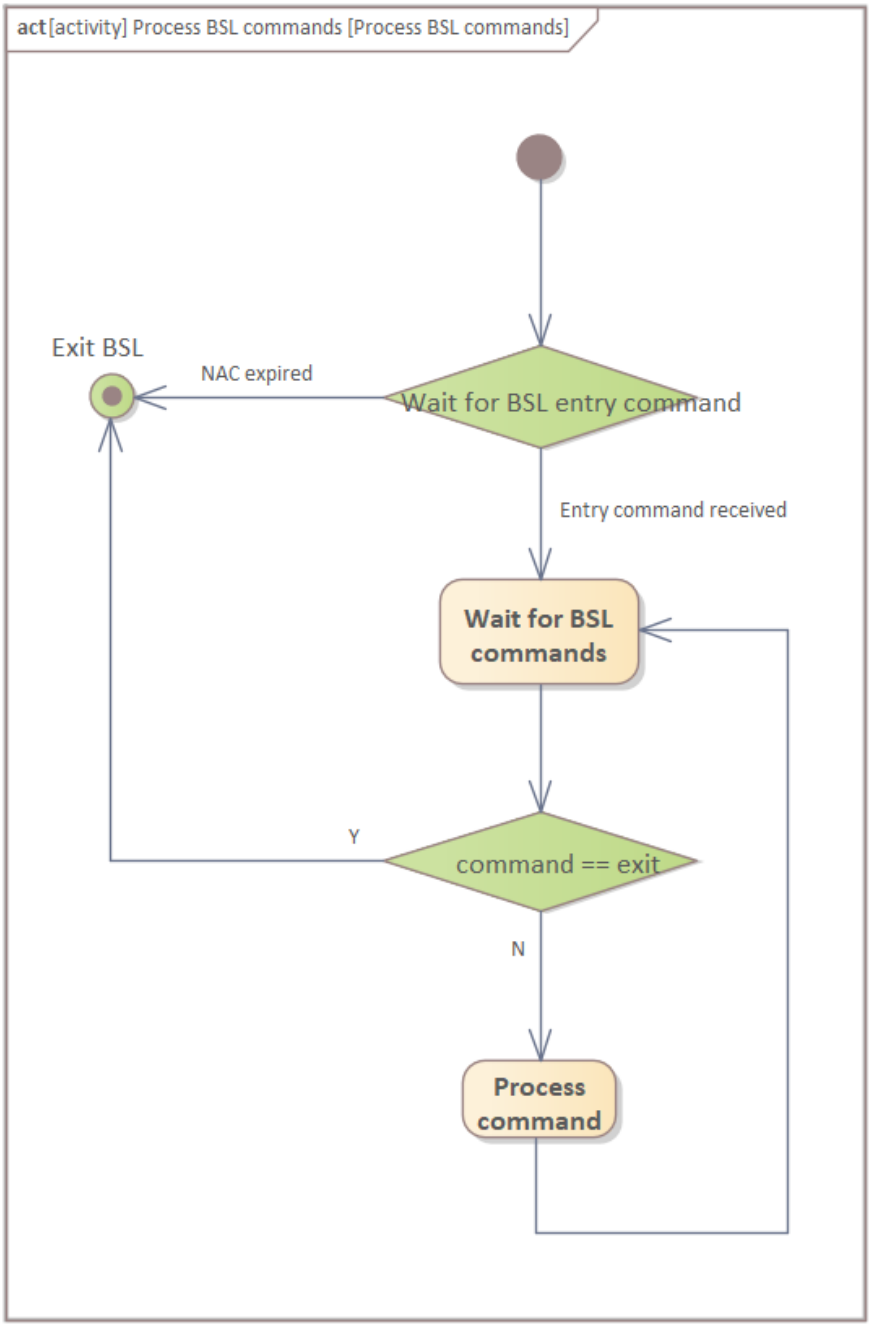


Figure 16: BSL command processing

UC8 - Interface management

This use case can be done in two different ways, via BSL and via startup routine. It extends the respective use cases UC7 and UC11.
The user can decide to deactivate (UC8.1, Figure 18) or to activate (UC8.2, Figure 19) the interface (both BSL and SWD).

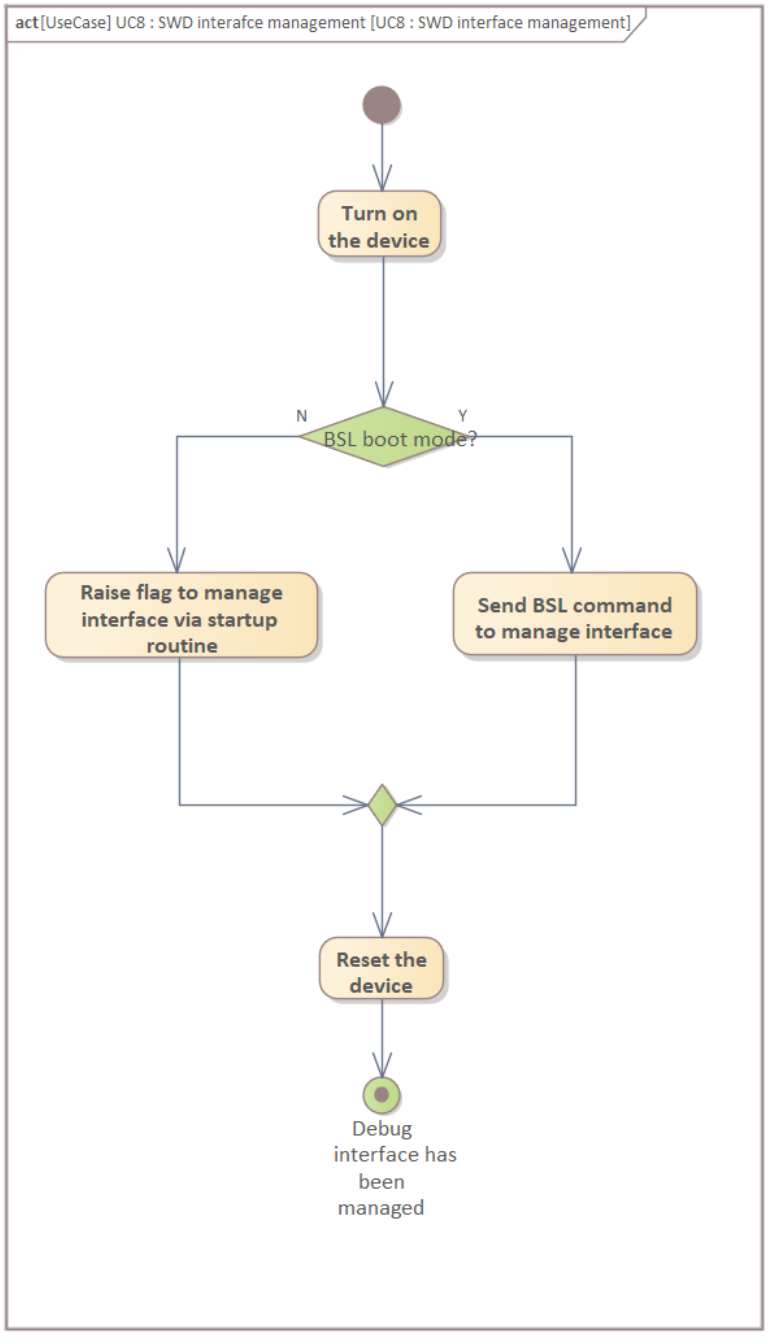


Figure 17: Activity diagram for the UC8 - Interface management

UC8.1 - Deactivate interface

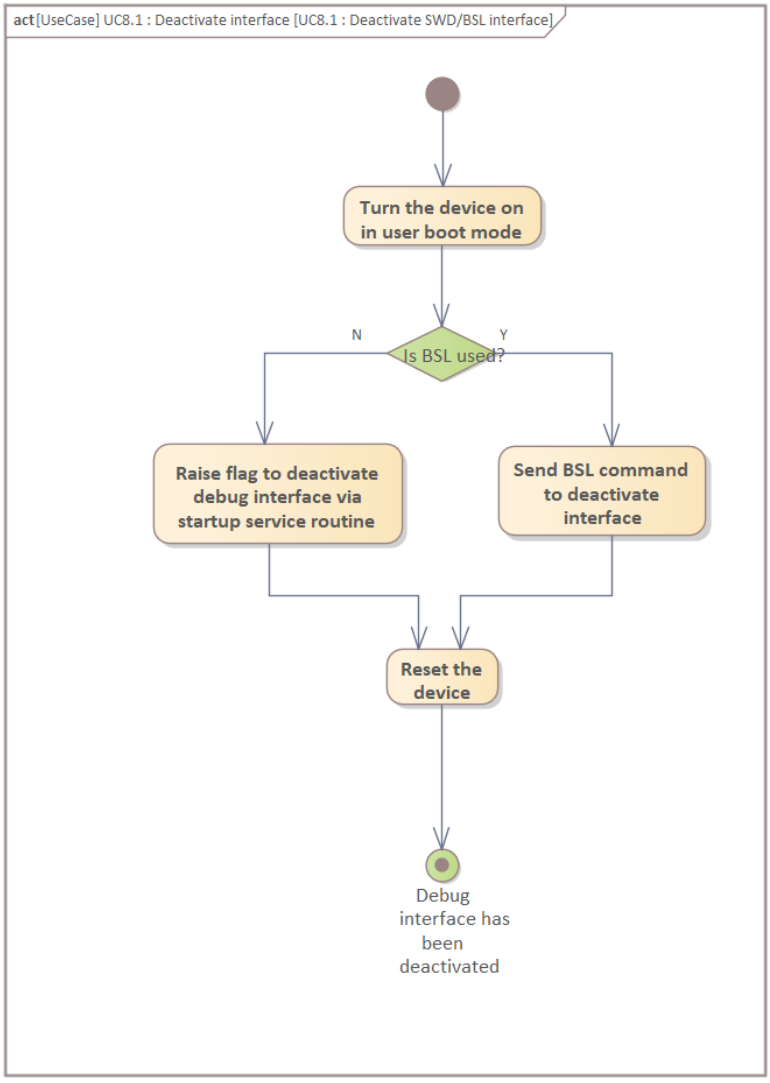


Figure 18: Activity diagram for the UC8.1 - Deactivate interface

UC8.2 - Activate interface

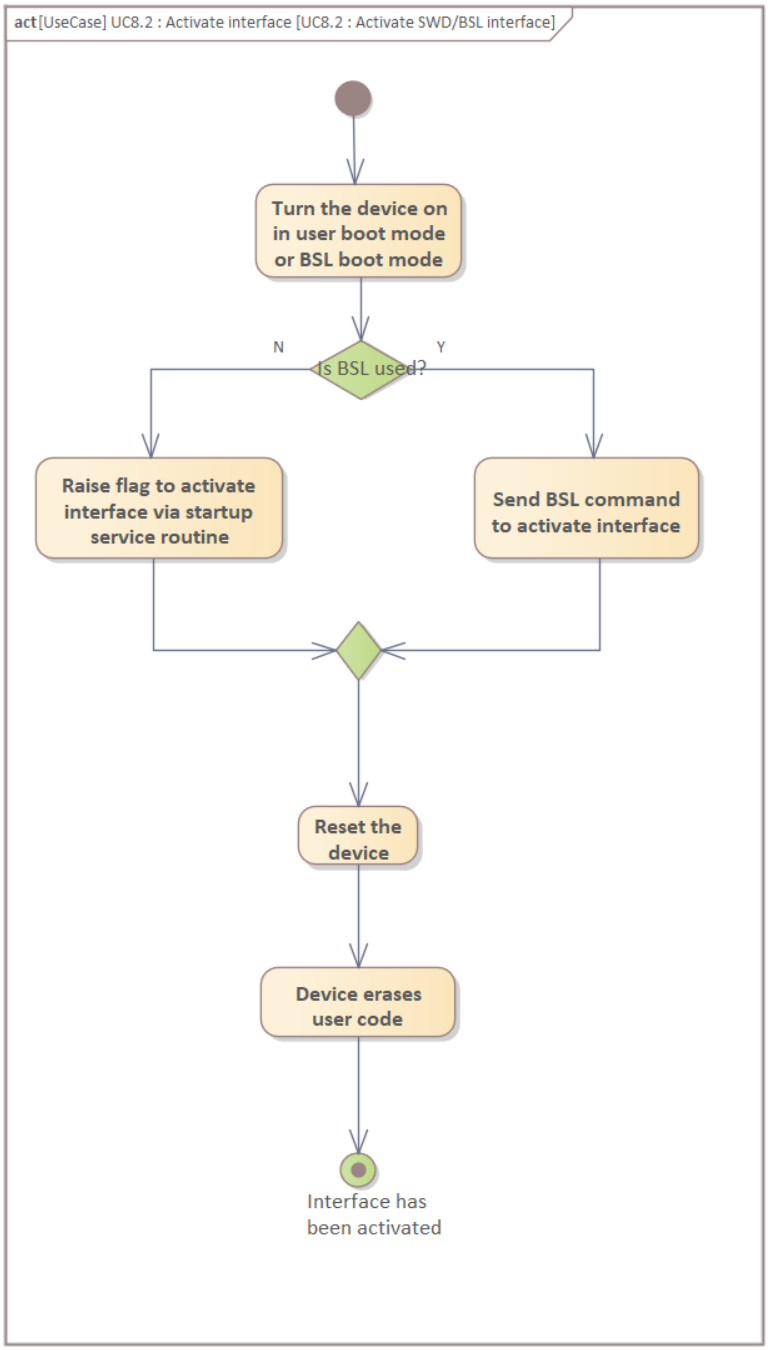


Figure 19: Activity diagram for the UC8.2 - Activate interface

UC9 - Configure BSL

The BSL can be configured in different ways: for example user can decide to extend the NAC window (UC9.2, Figure 21), to disable it (UC9.3, Figure 22) or to change

the BSL address from its default value (Figure 20).

UC9.1 - Change BSL register address

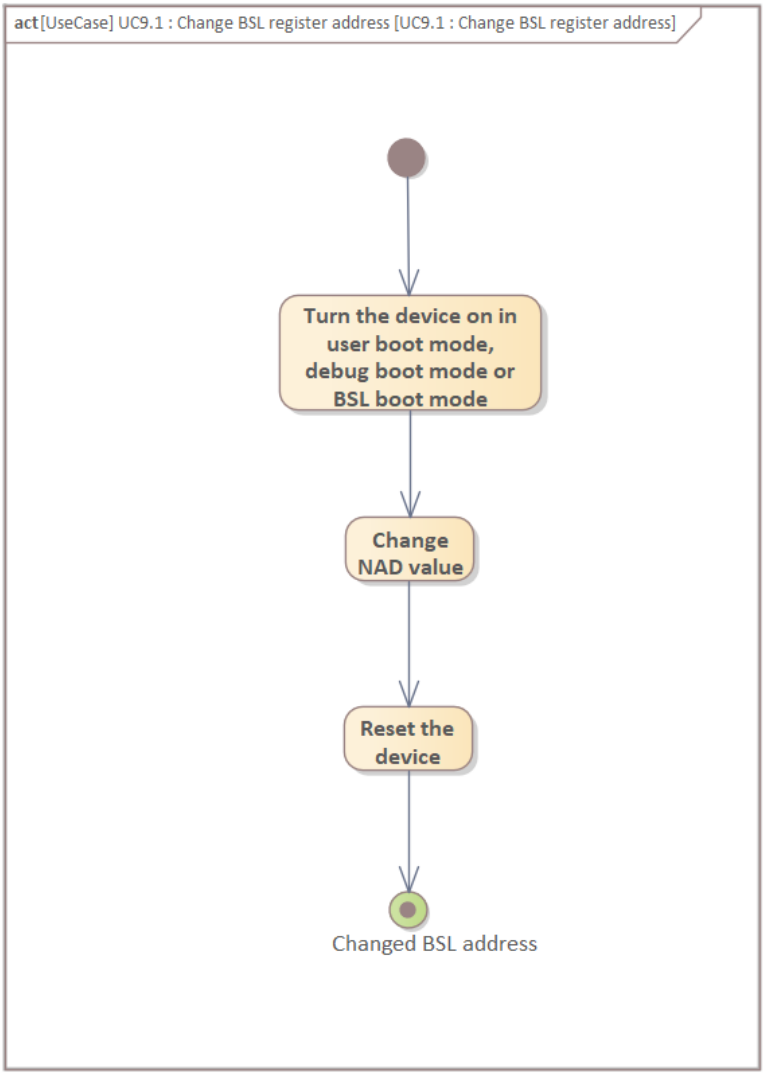


Figure 20: Activity diagram for the UC9.1 - Change BSL register address

UC9.2 - Change BSL window time

In this case the user can change the BSL window time. In particular he sets the NAC value. If the value is between 1 and 28, the $Window\_time = NAC \times 5ms$. After that time the device jumps to the user code stored in the NVM. Otherwise, if the value is bigger than 28, the device enters a loop where it waits forever for BSL commands and it never executes user code.

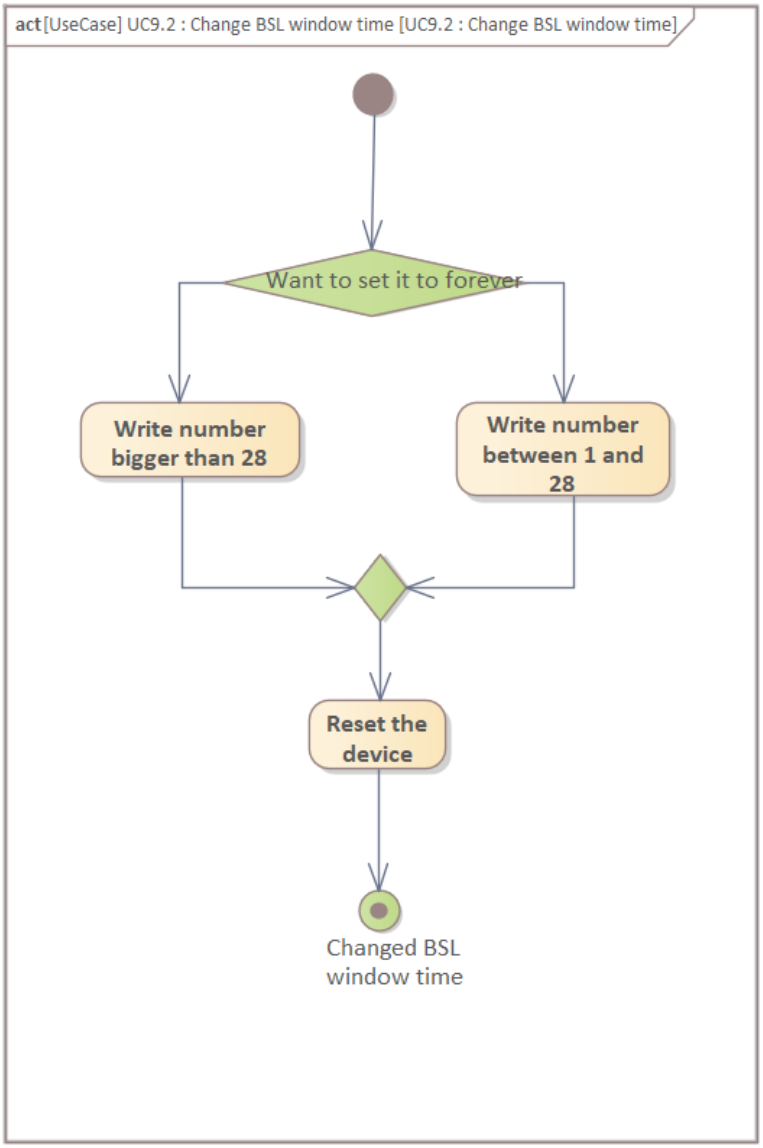


Figure 21: Activity diagram for the UC9.2 - Change BSL window time

UC9.3 - Disable BSL

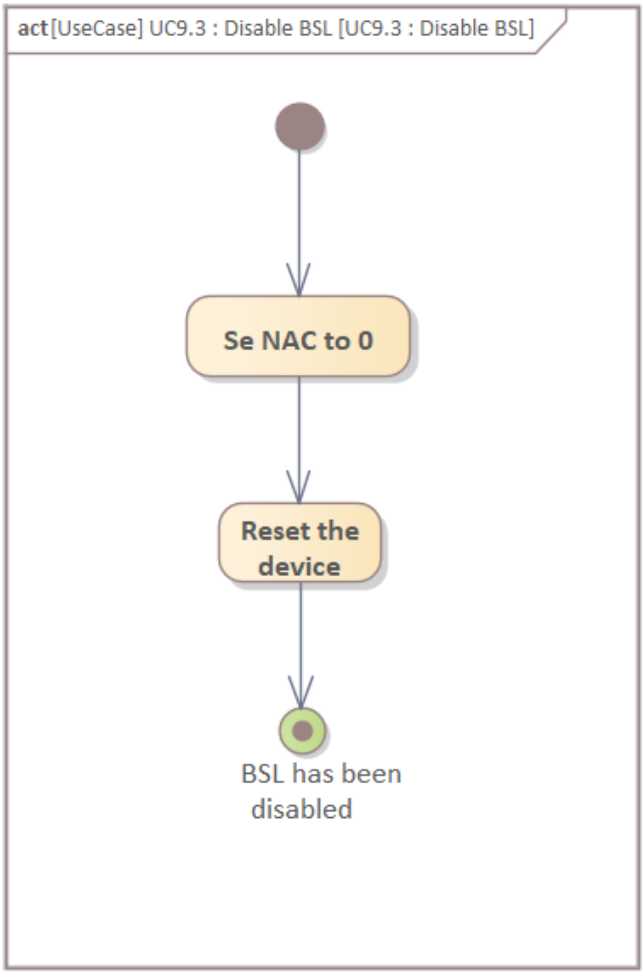


Figure 22: Activity diagram for the UC9.3 - Disable BSL

UC11 - Interacts with the product via startup service routine

It can happen that the user wants to interact with the product via BSL, but both the interfaces are deactivated and so he can not use it.
In this particular case the device provides a way to manage the interfaces, and it is a feature called "Startup service routine" (Figure 23).
There are only a few others operations the user can perform in this mode more than the interface management, for example reading or writing a specific sector of the memory.

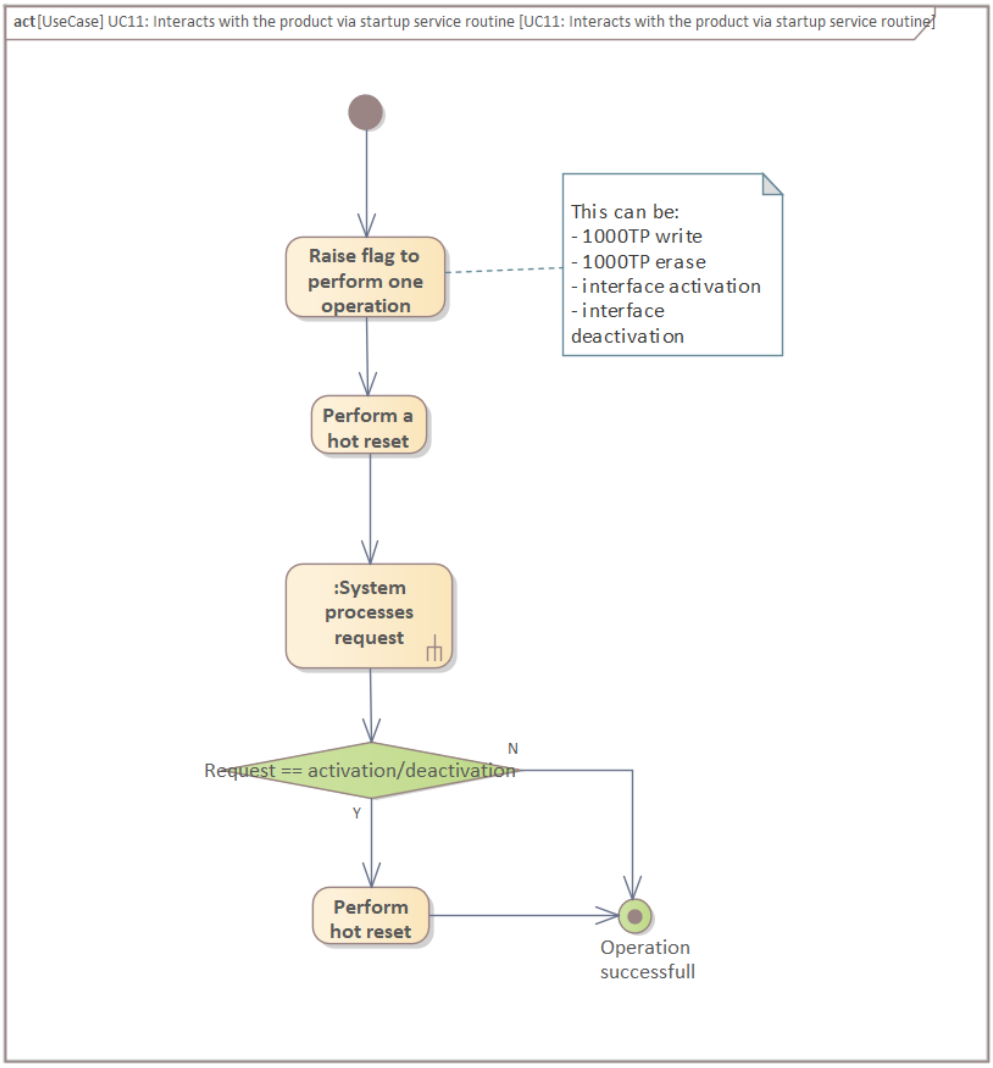


Figure 23: Activity diagram for the UC11 - Interacts with the product via startup service routine

For this use case I also created another diagram (Figure 24 at page 28) that explains how different requests are handled by the device.

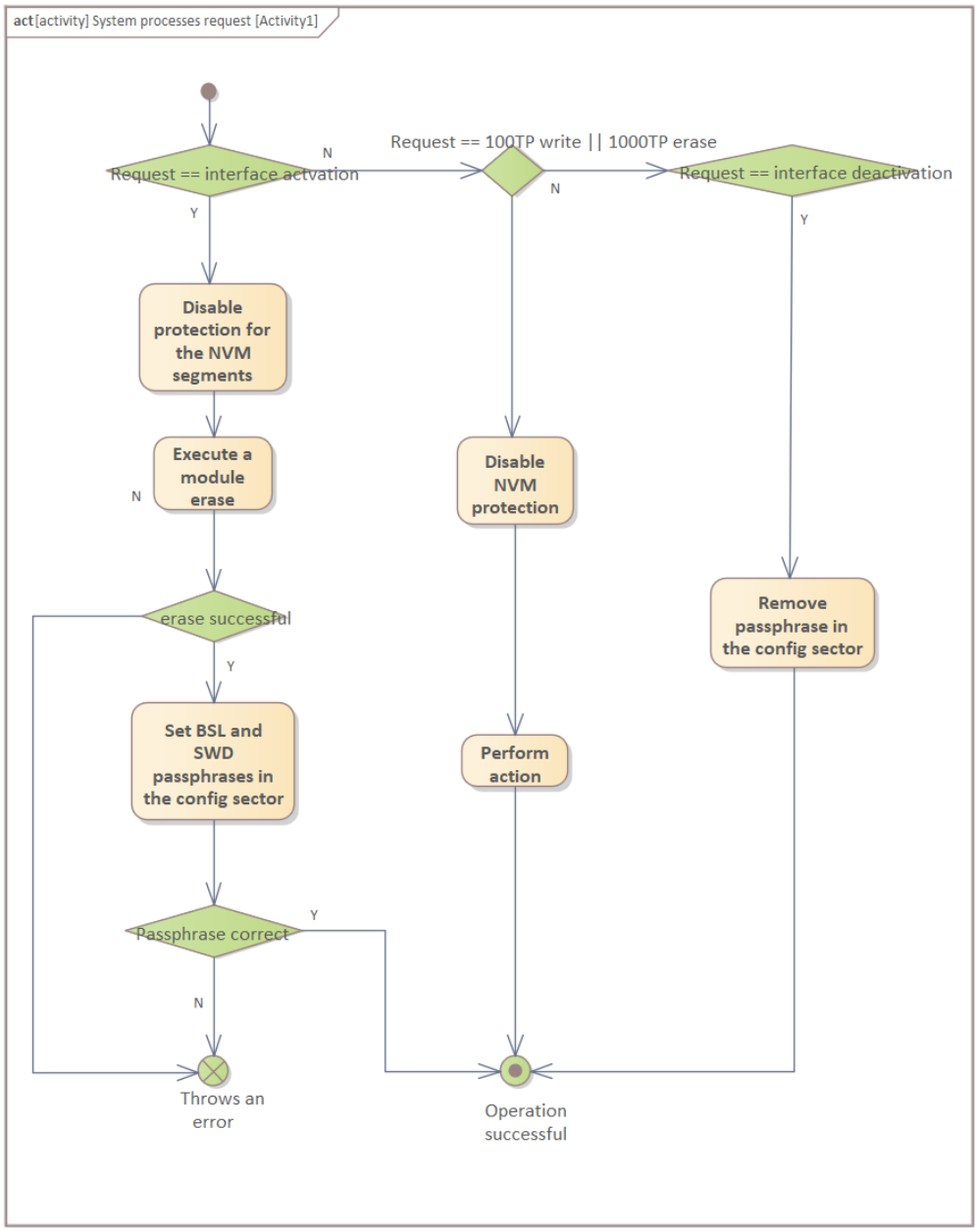


Figure 24: Request processing

3.4 Practical application

In the second part of the internship experience, I focused on the BSL interaction with the product and so I recreated some of the identified use cases.
The goal was to better understand both BSL protocol and LIN protocol, and to see the device response for a given command.

3.4.1 BSL via LIN protocol

LIN BSL is a LIN-like protocol [10]. The LIN protocol layer handles incoming frames and proceeds to forward the given command to the BSL protocol layer who is responsible for response message handling.
The frame format can be seen in the following picture (Figure 25).

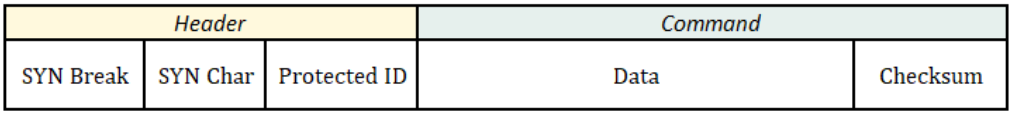


Figure 25: Frame format of a LIN message

- **SYN Break**: specified as a dominant condition at least 13 bit times;
- **SYN Char**: byte sent to enable the slave to determine the transmission rate the master uses. The bit pattern is `0b01010101` (see Figure 26) and so it has a fixed hex value of `0x55`;
- **Protected ID**: contains information about sender and receiver and the number of bytes which is expected in the response.
  It can assume the following values depending on the message type:
  - `0x3C` in the master request header;
  - `0x7D` in the slave response header.
- **Data**: consists of 8 bytes. For BSL over LIN the first byte represents the NAD_G while the remaining seven bytes contains information about the command to be performed;
- **Checksum**: checksum calculated with data bytes.

In particular I used FastLIN that is a LIN enhancement supporting higher baud rates.
In order to communicate with the device, user must send the passphrase to activate the interface. A passphrase consists of two consecutive frames sent by the host (the PC) and for FastLIN communication the frames are extended with the checksum byte.
As you can see in Figure 27, the BSL communication is unlocked if and only if the passphrase frames are correctly sent to the device without any another messages in

between.
The content of the passphrase frames is described in Figure 28.

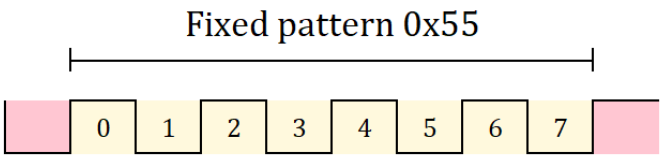


Figure 26: Sync field byte

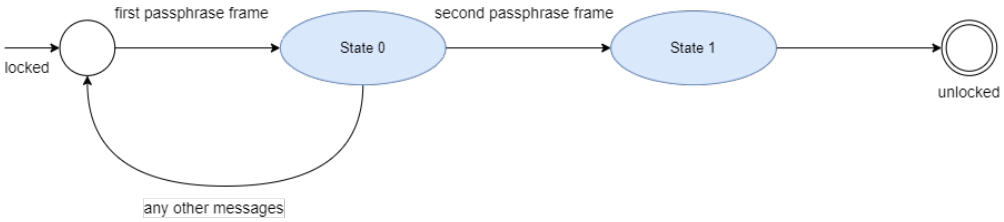


Figure 27: Finite state machine describing the correct unlock sequence

		0	1	2	3	4	5	6
Frame 0	ASCII	F	L	I	N	P	A	S
	Hexadecimal	0x46	0x4c	0x49	0x4E	0x50	0x41	0x53

		0	1	2	3	4	5	6
Frame 1	ASCII	S	P	H	R	A	S	E
	Hexadecimal	0x53	0x50	0x48	0x52	0x41	0x53	0x45

Figure 28: Passphrase content

To implement some features, for example sending LIN frames via serial, I wrote a script in Python (see Section 3.5).
Moreover I used a logic analyzer [5] to actually see the frames I was sending and how the device was answering. A logic analyzer is an electronic instrument that is able to capture and display multiple signals from a digital system. By setting the right baud rate and with the correct pin configuration, I could see in real time how the PC (the host) and the device (the slave) were communicating.
In Figures 29 and 30 you can see the actual passphrase sent via serial to the device.

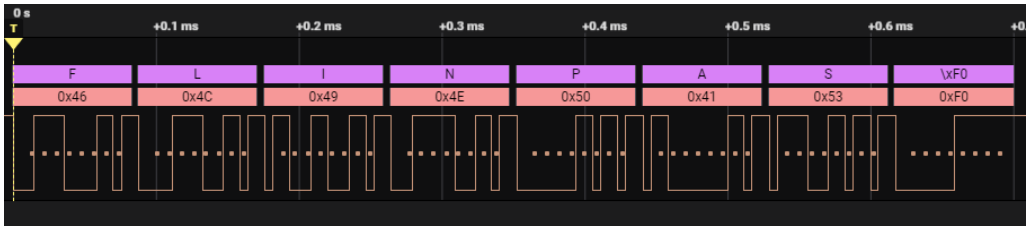


Figure 29: Capture with the Saleae tool [6] of the first frame of the passphrase

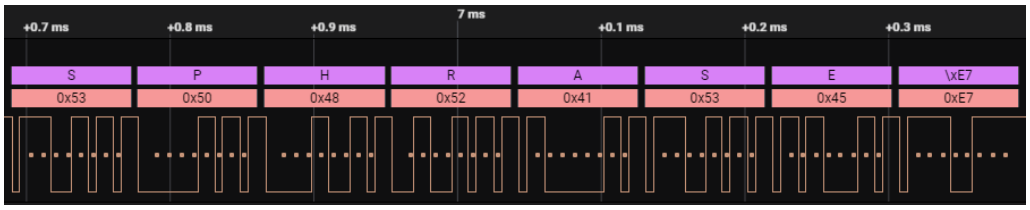


Figure 30: Capture with the Saleae tool [6] of the second frame of the passphrase

The last byte is the checksum byte: it contains the inverted eight-bits sum with a carry over all data bytes. An eight-bits sum with carry is equivalent to the sum of all values, substracted by 255 every time the sum is ≥ 256.
With the passphrase set, I was finally able to send specific commands and wait for the device response. For example, I tried to ask for the NAD address by sending the corresponding BSL command.
The capture of the digital analyzer is shown in Figure 31.

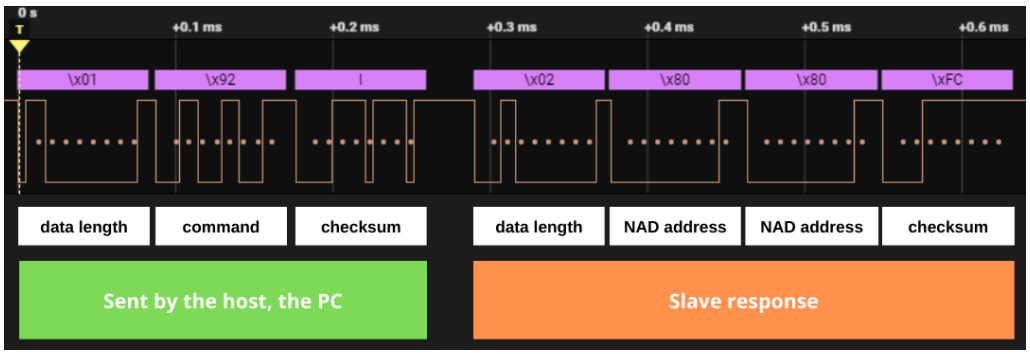


Figure 31: Get NAD address command request and response

3.4.2 NVM Write command

Then I tried to write in the NVM some data in order to verify *UC6 - Download program in the NVM* (§3.3). In fact, to download user code in the NVM means to write zeros and ones in the memory. The corresponding BSL command consists in two different frames. The communication between the host and the slave for the write command is described in Figure 32.

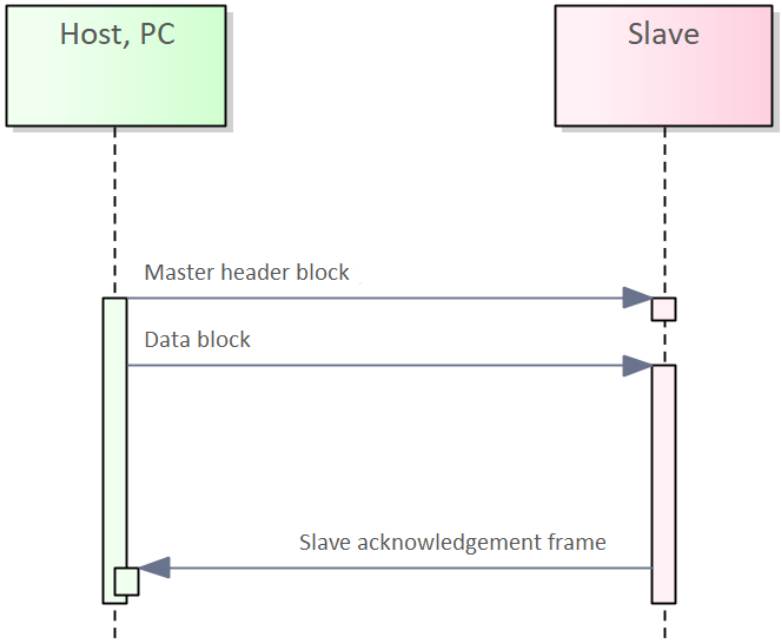


Figure 32: Sequence diagram for the NVM write command

The master header block, that consists of seven bytes [1], is structured as shown in Figure 33. In particular:

- **Data length**: set to `0x06`;
- **Command identifier**: set to `0x05` for NVM write;
- **Address**: Address offset where to store data, maximum of 24 bits (3 bytes);
- **Reserved byte**: set to `0x00`;
- **Length of following data**: number of data bytes to write.

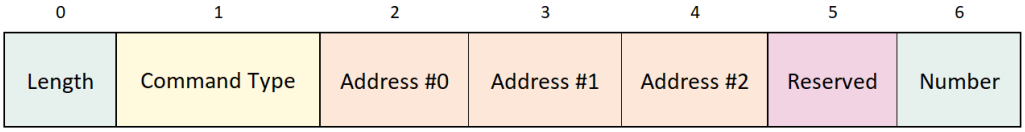


Figure 33: Master header block for NVM write command

The data block is described in Figure 34.

[1] Not considering the checksum byte that is always attached to the frame and calculated as described at page 31

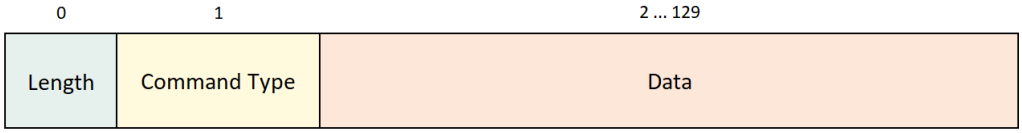


Figure 34: Data block

Then I sent the BSL command via Python script, and you can see the actual header frame sent by the host in Figure 35 as well as the data block containing the data to write and the slave acknowledge frame in Figure 36.

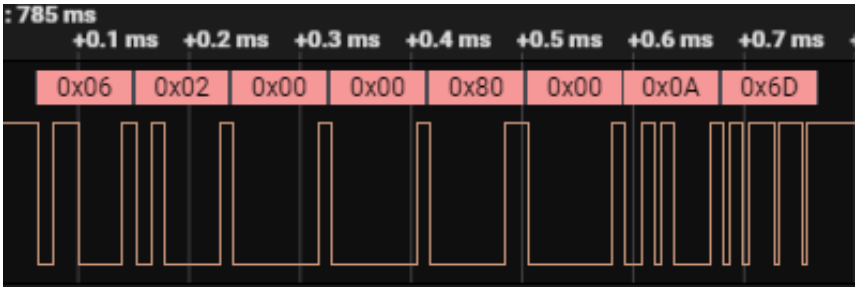


Figure 35: NVM write command header block

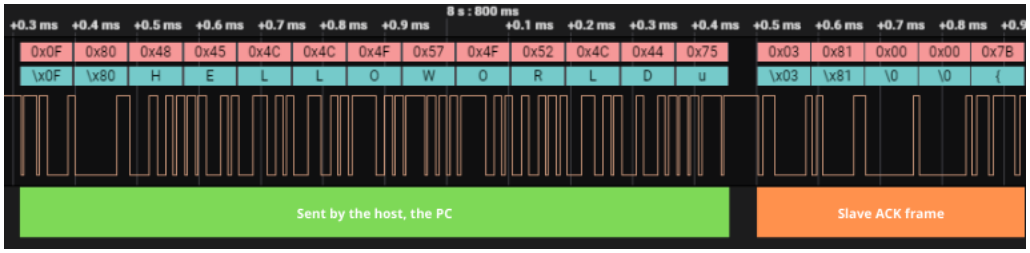


Figure 36: Data block and slave acknowledge

3.4.3 NVM Read command

In order to check that the NVM was correctly written with the data I sent, I also tried to read the memory (I used the same address offset of the write command) by sending the relative BSL command.
The frame consists of seven bytes [2]:

- **Data length**: set to `0x06`;
- **Command identifier**: set to `0x84` for NVM read;
- **Address**: Address offset where to start reading;

[2]Not considering the checksum byte that is always attached to the frame and calculated as described at page 31

- **Reserved byte**: set to `0x00`;
- **Length of following data**: number of data bytes to read.

A picture of the frame is shown in Figure 37. In this way I was able to demonstrate *UC5 - Verify NVM code* as well.

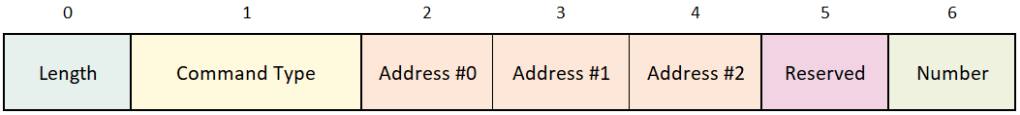


Figure 37: Master header block for NVM read command

The actual frame sent and the slave response with the required data has been captured by the logic analyzer and is shown in Figure 38.

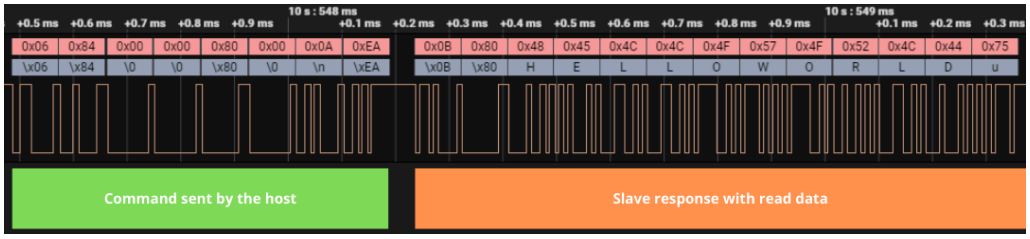


Figure 38: Capture of read command and response

3.5 Python Script

First of all I had to import in the Python script the libraries I needed, as shown in Listing 1.

```
import serial
import time
import serial.tools.list_ports
from serial.serialutil import SerialException
from struct import *
import binascii
```

Listing 1: Libraries import

Then I created a class `LinMaster` and implemented some basic features to communicate over serial, as shown in Listing 2. The baudrate is set to 115200bps since we used the FastLIN.

```
class LinMaster:
    def __init__(self, comPort=None):
        self.baudRate = 115200 #set baudrate to 115200
        self.syncBaudRate = 9600 #set slave synchronization
    baudrate to 9600
        self.ser = serial.Serial(baudrate=self.baudRate, stopbits=
    serial.STOPBITS_TWO)
```

```
        self.ser.timeout = 1
        if comPort is not None: #set COM port
            self.setPort(comPort)

    """closes the master communication"""
    def close(self):
        self.ser.close()

    """gets all the available ports"""
    def getPorts(self):
        ports = serial.tools.list_ports.comports()
        ports.sort(key=lambda x: x.device)
        return ports

    """sets the selected port"""
    def setPort(self, port):
        print(f"Port {port} opened")
        self.ser.port = port
        if (not self.ser.is_open):
            self.ser.open()
```

Listing 2: Serial configuration

In order to send frames via serial I wrote a function that takes the data to send as an argument, it calculates the checksum and send the data

```
    """Send BSL frame"""
    def send(self, data: bytes) -> None:
        # Calculate classic checksum and convert to byte
        checksum = 0xFF ^ reduce(
            lambda a, b: (a + b) if (a + b) < 256 else (a + b - 255), data
        )
        #send frame
        self._sendData(data + checksum.to_bytes(1, "big"))
```

Listing 3: Send function

The checksum is calculated with a lambda function [4]. A lambda function is just like any other Python function, except that it has no name when defining it, and it is contained in one line of code. That allows to create small, single-use functions that can save time and space in the program.
The keyword `reduce` is used to apply a particular function passed as its argument to all of the list elements mentioned in the sequences passed along. In this case the function is the lambda function while the elements are the single bytes of the `data` passed as an argument.
To perform the three operations described in Section 3.4 I wrote three different functions, one for each command. The code for getting the NAD is shown in Listing 4, the function for writing in the NVM a simple `HELLOWORLD` is shown in Listing 5 and finally the one for the NVM read is reported in Listing 6 below.

```
    def getNad(self):
        self.send(b"\x01\x92")
```

Listing 4: Get NAD function

```
    def NVMwrite(self):
        self.send(b"\x06\x02\x00\x00\x80\x00\x0A")
        time.sleep(0.00001)
        self.send(b"\x0B\x80" + b"HELLOWORLD")
```

Listing 5: NVM Write function

```
    def NVMread(self):
        self.send(b"\x06\x84\x00\x00\x80\x00\x0A")
```

Listing 6: NVM Read function

4 Conclusions

4.1 Generic overview

During my internship experience I worked as an Application Engineer at *Infineon Technologies*.
Infineon is a German semiconductor manufacturer and I worked there for two months. My main task was to identify the use cases of the interaction of the customer with *Infineon* products and subsequently to create the use case and activity diagrams using SysML to describe all the operation a user can perform with the product.
Finally, I focused on the Bootstrap Loader interaction and implemented via Python a script that allowed the communication with the device.

4.2 Final balance

I reported in Table 3 a comparison between the estimated time for each activity in the preliminary schedule and the actual activities completed during the internship experience.

Activity	Spent time	Estimated time
Preliminary studies	5 days	8 days
Analysis and modeling	14 days	20 days
Firmware optimization	14 days	-
Verification with a demonstrator	7 days	12 days

Table 3: Report of the completed activities

4.2.1 Preliminary studies

In the first weeks of the internship I had to collect all the information about interior lighting and the current solutions in the automotive field, as well as some innovative trends.

4.2.2 Analysis and Modeling

In the second part of the experience I finally started to gather all the information. I found it really helpful to talk with other departments inside *Infineon* in order to cover as many points of view as possible.
Due to all the meetings I had to make with some colleagues, this phase required almost six extra days since I had to add more and more details as I talked with different people.

4.2.3 Firmware optimization

The initial idea of writing color mixing code was replaced with a task focused on the general usage of the device. A core functionality and feature is to be able to download application software and calibration data onto a device. As a result, the practical part was changed from firmware development to implementing the bootstrap loader protocol and writing data to a device.

4.2.4 Verification with a demonstrator

In the last part of my internship experience I focused on both the BSL and the LIN protocols. This activity required some extra days because I also had to study the protocols, before implementing the commands via Python script.
After having collected all the important information, I could finally connect to a device and use the BSL interface.

4.3 Personal feedback

At first, the thought of working in such a big company like *Infineon* both scared me and excited me.
On the one hand I was fascinated by the idea of understanding what it was like to work in such a complex environment; on the other hand I was afraid I could not meet the company requirements with me being only a bachelor student with no experience in the field.
With *Infineon* being such a big company, I really thought I would have found a strict and severe environment: instead I was astonished by the friendly and warm atmosphere in the office.
Behind the high quality products *Infineon* provides, there is the company welfare for its employees: I think it is extremely important to allow one to feel confident and appreciated so that one can be more productive and work more efficiently.
Nowadays work is considered one of the main causes of stress, and working for a company with a billion dollars turnover, obviously represents a huge responsibility. It was amazing for me to see the attention *Infineon* gives to all its workers and I honestly think this is the key behind every successful company.
Concluding, I can say I learnt a lot, both from a professional and a personal point of view and I am sure I will make use of my experience, whatever the future will hold for me.

5 Glossary

BSL$_G$ BootStrap Loader. 13

Formula SAE$_G$ Formula SAE is a student international engineering design competition organized by the Society of Automotive Engineers in 1980.
A Formula SAE team must be made up entirely of active university students, and creates the challenge if we consider the restrictions on available work hours, skills set and experience. There are different categories, such as Formula Combustion, Formula Hybrid, Formula Electric and Formula Driverless. 8

LIN$_G$ Local Interconnect Network (LIN) is a serial network protocol that allows master-slave communication between devices. It is largely used in the automotive field. 16

NAC$_G$ No Activity Counter. Its value defines the time window after reset release within the firmware is able to receive BSL connection messages. 13

NAD$_G$ Node Address for Diagnostics. It specifies the address of the active slave node. 29

NVM$_G$ Non-Volatile Memory. 17

OMG$_G$ The Object Management Group (OMG) is a computer industry standards consortium.The goal of the OMG was a common portable and interoperable object model with methods and data that work using all types of development environments on all types of platforms. 11

SWD$_G$ The Serial Wire Debug (SWD) debug interface is used for programming firmware and accessing registers. 14

UML$_G$ The Unified Modeling Language (UML) is a general-purpose modeling language intended to provide a standard way to visualize the design of a system. 11

References

[1] *Benefits of Model-Based Systems Engineering*. URL: https://bit.ly/3A1xDdw.

[2] *Infineon - Company Presentation*. URL: https://bit.ly/3u68CtJ.

[3] *Infineon Technologies - Wikipedia*. URL: https://bit.ly/3NlhlPz.

[4] *Lamda Functions with Practical example*. URL: https://bit.ly/3Rf8HFu.

[5] *Logic Analyzer - Wikipedia*. URL: https://bit.ly/3apY9me.

[6] *Logic Analyzers from Saleae*. URL: https://bit.ly/3Awp4aO.

[7] Driving Vision News. *Interior lighting report*. 2012. URL: https://bit.ly/3bsxbe0.

[8] Nataliya Shevchenko. *An Introduction to Model-Based Systems Engineering (MBSE)*. 2020. URL: https://bit.ly/3bkl3vs.

[9] OMG SysML. *What is SysML? | OMG SysML*. URL: https://bit.ly/3bklfea.

[10] *TLE984x Firmware User Manual*. URL: https://bit.ly/3yjnZS9.

[11] *What is Python? Executive Summary*. URL: https://bit.ly/39U6oqH.