Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Master Thesis

Light sensor development for Ara platform

Author:
Alexis DUQUE

Supervisor:
Pr. Josep Paradells Aspas

Wireless Network Group
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

June 2015

Abstract

INSA de Lyon - Télécommunications, Services et Usages
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Master's degree in Telecommunications Engineering

Light sensor development for Ara platform

by Alexis DUQUE

During the last years, Visible Light Communication (VLC), a novel technology that enables standard Light-Emitting-Diodes (LEDs) to transmit data, is gaining significant attention. In the near future, this technology could enable devices containing LEDs – such as car lights, city lights, screens and home appliances – to carry information or data to the end-users, using their smartphone. However, VLC is currently limited by the end-point receiver, such as a the mobile camera, or a peripheral connected through the jack input and to unleash the full potential of VLC, more advanced receiver are required.

On other, few year ago, Google ATAP - the Google innovation department - announced the Ara initiative. This consist on a modular phone where parts of the phone, like cameras, sensors or networks can be changed. So when a new feature appears or required by the user it is not needed to change the mobile phone, just to buy the modules with the functionality.

This Master Thesis presents the design and development of a simple module that will support communication by light (VLC) using the Ara Module Developer Kit provided by Google. It consists on building a front-end circuit, connecting a photodiode that receives the level of light and use it as data carrier, in order to receive and display data inside a custom Android application on the Ara smartphone.

Acknowledgements

To my supervisor, Josep Paradells which trust me and give me the opportunity to work on this project.
To the Wireless Network Group team for there guidance and everyday help and support.
To Miguel and Cleo, my hosts during these five past months in Barcelona.

Contents

List of Figures

List of Tables

Abbreviations

ADC	**A**nalogic to **D**igital **C**onverter
BPS	**B**it **P**er **S**econd
DMA	**D**irect **M**emory **A**ccess
I^{2}C	**I**nter**I**ntegrated **C**ircuit
MDK	**M**odule **D**evelopement **K**it
MCU	**M**icro **C**ontroller **U**nit
Op-Amp	**Op**erational **Amp**lifier
OS	**O**perating **S**ystem
OOK	**O**n **O**ff **K**eying
PTM	**P**ulse **T**ime **M**odulation
PAM	**P**ulse **A**mplitude **M**odulation
PWM	**P**ulse **W**idth **M**odulation
RAM	**R**andom **A**ccess **M**emory
SDK	**S**oftware **D**evelopement **K**it
TIA	**T**rans **I**mpedance **A**mplifier
UART	**U**niversal **A**synchronous **R**eceiver **T**ransmitter
VLC	**V**isible **L**ight **C**ommunication

Chapter 1

Introduction

1.1 Motivations

For the last century, radio frequency (RF) has dominated wireless communications.But RF has been a victim of its own success. The number of mobile devices and embedded systems has increased tremendously as technology becomes a primary need in peoples lives. This explosion of mobile devices comes at a cost:the frequency spectrum is a scarce resource. The massive increase in RF communicating devices is leading to a saturation of the available bandwidth, which will result in a drop of quality-of-service.

To ameliorate the bandwidth saturation problem, the research community has been exploring other wireless technologies. One of the most promising alternatives is visible light communication. Light is electromagnetic radiation just like RF, but the difference lies in frequency. Because of this frequency, the interaction between light and matter is different on a fundamental level which gives light unique properties.

With the advent of Visible Light Communication (VLC), the widespread exploitation of the visible light spectrum is becoming a reality. VLC enables standard Light Emitting Diodes (LEDs) to transmit data wirelessly, and this is an important step because LEDs are permeating our daily environments at a very fast pace.

After intensive research into energy consumption, in 2009 the European Union and other countries started measures to phase out incandescent light bulbs in favor of high efficient LEDs. But it is not only residential and commercial lighting that is being replaced with

LEDs, a number of other objects such as car lights, city lights, billboards, smartphone and laptop screens, price tags, toys and home appliances, are also using LEDs to reduce their energy consumption.

A nice VLC example is indoor positioning. At the 2014 Mobile Wolrd Congress (MWC 14'), the i2CAT foundation demonstrate its indoor location positioning system, using VLC and the phone's ambient light sensor.

Other applications, such navigation assistance for visually impaired, propose an other approach using an external peripheral as VLC receiver, plug-in it to the jack mobile phone or through Bluetooth [1].

Thus, considering that VLC can potentially transform any LED device into a wireless transmitter there may be a new generation of objects waiting to be connected to people smartphone.

1.2 Objectives

However, actual research and VLC implementation on mobile phone are highly limited by receiver hardware. In fact, embedded sensor on smartphone, such as camera are not appropriate for sensing modulated light. External peripheral, adding an other channel reduce considerably the throughput. That why, the recent concept of modular smartphone, called Phonebloks (2013) and industrialized as Project Ara y Google, could be the solution.

Project Ara is an effort in Google's Advanced Technology Projects (ATAP) organization to create a modular smartphone platform, with the twin aims of delivering a deep customization experience to users and enabling significantly lowered barrier to entry into the mobile hardware ecosystem.

This project aims to take advantage of this new technology developing a Visible Light Communication receiver for the Ara plartform. Major objectives are listed below :

- Design and develop a module that support visible light communication.
- Determine the Ara smartphone possibility for future work on wireless communications.

- Develop a convenient LED driver suited for our need.
- Experiment a communication between a LED emitter and the modular smartphone.

1.3 Report Organization

Chapter 1 of this report serves to provide an introduction of the basic concepts and techniques and also shows several designs that are required for the implementation of VLC.
Chapter 2 provides the background needed for the VLC designs.
Chapter 3 provides the literature review of the VLC technology.
Chapters 4 and 5 provide our proposal description and the experimental setup and implementation of the models.
Chapter 6 presents our results and recommendations for improving the designs as well as the conclusion with suggestions for further improvements in the work.

Chapter 2

Technical background

2.1 VLC Standard

VLC as we know it today would not be possible without LEDs. Before we introduce our proposal, we provide background information about LEDs as wireless transmitter, then we describe the many ways in which light intensity can be modulated and received, and the line coding scheme used in indoor VLC.

2.2 VLC Technology

Modulation of light has been used for centuries. In 1792 the French inventor Claude Chappe invented the optical telegraph, a system of towers with signaling devices, which allowed Napoleon to pass messages throughout his empire. Although simple and practical, these systems are the ancestors of VLC. For some decades after these events, light communication in the form of morse was used, but further development of light communication stood still because with incandescent lamps the data rates pale in comparison to what could be achieved with radio. In the same way, Infrared(IR) has been introduced for a long time, but they are limited to Line-Of-Sight (LOS), with a very short range use case, and are very sensitive to the weather or obstacles.

It is only until recently that we have a pervasive infrastructure that can modulate light to achieve data rates that are meaningful for today's communication needs. Due to recent advancements in LED technology we can transform light bulbs into high speed wireless transmitters. Data can be modulated in every light source using changes in intensity, but only certain light sources possess the necessary properties to transmit high data rates.

Fundamentally, modulating light requires changes of light intensity. For the last century incandescent lamps have been the primary source of light, but incandescent light cannot comply with high speed modulation because of the mechanism it uses to generate light. Incandescence is the effect of emitting thermal radiation from matter as a result of its temperature. In incandescent light bulbs a wire is heated by running a current through it, and the resistance of this wire forms kinetic energy which is released in the form of light. This means that intensity control of incandescent lamps takes place through two steps, resulting in indirect control of the signal. This would not be a problem if the thermal inertia would not make the system too slow for high speed modulation, but it does.

In the case of LEDs, the direct relation between intensity and electricity permit high speed modulation. LEDs consist of a semiconductor material that contains excited electrons. A fundamental property of electrons is that when they are forced into a lower energy state they release their energy in the form of the emission of photons. This effect is called electroluminescense and gives direct control over light intensity through the control of voltage and current. A simple circuit with transistors can deliver the necessary control of current, and this makes the changes in light intensity fast enough to transfer information at a high data rate.

An alternative to LEDs is laser. Lasers can also be controlled at high speed, and have the additional capability of strengthening the electroluminescense effect by amplifying and focusing the generated light. This makes laser a good candidate for long range communication, but for short distances its transmission angle is too narrow (and hence it can be easily obstructed) without using lenses. Another important disadvantage of laser is that poses health hazards to human beings.

2.2.1 Modulation and transmitter

The light emitted by LEDs can be modulated in different forms, and there are also several types of receivers that can be used to decode the modulated light. Next, we describe all these options.

As mentioned before, the ability of LEDs to modulate light at high speeds make them the obvious choice for VLC transmitters in terms of speed and light intensity, but potentially good transmitters are not the only thing that make a communication system work. Data needs to be encoded (or modulated) into a signal before transmission. In radio communication, two common parameters used in signal modulation are amplitude and frequency. Although radio and light are both electromagnetic waves, the effect of modulation is not the same. We will briefly describe this point to make the reader aware of the fundamental difference in modulation between radio and VLC.

In radio, frequency modulation changes the frequency of the waves in the electromagnetic field, and amplitude modulation changes the height of this waves. In VLC, amplitude modulation works the same way as in radio, and it is often called intensity modulation. Frequency modulation however, is very hard to apply with visible light. It can be done but requires advanced laser setups to change the signal's properties, like phase and frequency, through interference.

There are many different modulation schemes used for VLC, and we will now discuss the most commonly used.

1. On-Off-Keying (OOK), or amplitude shift keying, uses keying (switching) to turn a carrier signal on and off. OOK has a low processing burden but is proven to be very sensitive to noise. Enhanced schemes like On-Off-Keying Non-Return-to-Zero have shown data rates of 1.5 Gigabits per seconds, using a Integrated Circuit with LEDs bonded into the chip [2].

2. Pulse Time Modulation (PTM) is a technique in which data is modulated in the ratio between the on and off time of the carrier signal. Pulse Position Modulation (PPM) and Pulse Width Modulation (PWM) fall into this category. The advantage of PTM is that it does not require digital-to-analog converters to generate a smooth output signal, and does not require an analog-to-digital converter either but only a comparator circuit. On the other hand, PTM requires accurate timing for both the receiver and transmitter because the ratio between on and off periods needs to be well synchronized. Another advantage of PTM is that it provides flickering

and dimming support without additional techniques, which is good for use in illumination devices, but speeds are lower than with other modulation systems [3].

3. Pulse Amplitude Modulation (PAM) uses brightness levels to realize multilevel signals to encode symbols. This modulation is sensitive to external light sources as they influence the intensity, and has relative low speeds compared to other techniques. Dimming control can easily be implemented in PAM by changing the probability of the constellation points [4]

4. Frequency Shift Keying (FSK) looks a lot like On-Off-Keying, but instead of switching a carrier on and off, the system switches between two frequencies. This switch in frequency can be in terms of color i.e. a one is red and a zero is blue, or in pulse frequency.

5. Phase Shift Keying (PSK) encodes symbols by changing the phase of the light intensity that has been shaped into a sinusoidal form. Many variants of PSK exist, depending on the number of constellation points used (Binary PSK uses 2 points, 0°and 180°, Quadrature PSK uses 4 points). PSK can be used as base modulation, but can also be used in combination with other techniques like OFDM to improve certain weaknesses like intersymbol interference [5].

6. Orthogonal Frequency Division Multiplexing (OFDM) is a technique that uses a large number of modulated carriers with sufficient frequency spacing so that they are orthogonal. Again, this frequency is modulated through intensity. The strongest advantage of OFDM is that it provides resistance to multipath effects, which result in long distances and high speed data transfers. Data rates beyond 3 Gigabit per seconds have been reported at short distances $\approx$ 10cm) [6]. Unfortunately such high speeds demand require high-speed processing units.

7. Quadrature Amplitude Modulation (QAM) is a modulation scheme that conveys bit streams by changing the amplitudes of two or more independent carrier signals, which results in a spectral efficient scheme. A form of QAM is the Carrier-less Amplitude and Phase (CAP) modulation, a promising high speed VLC modulation scheme capable of transmitting at 3.22 gigabits per second using an RGB-type led. QAM and CAP can both be combined with OFDM to improve its performance even further [7].

To select a good modulation scheme, people have designed models that can be used to compare different schemes by simulating environmental conditions. The Matlab-based platform of De Lausnay et al. evaluates different modulation techniques and can also help to select the right modulation type [8].

Based on the information above, we decided to use On-Off-Keying modulation for our platform because the use of this technique keeps hardware component count low as well as the modulation/demodulation programming stays simple.

2.2.2 Receivers

While on the transmitter side the most widely used element is the LED, on the receiver side we have several options: cameras, photodiodes and phototransistors, and LEDs themselves. The common name for these types of sensors is photodetectors and they use the same fundamental principle, the photoelectric effect: many metals release electrons when light shines upon it. Although these sensors share the same effect, they have subtle differences that result in unique characteristics.

1. Cameras can detect light, its intensity and its color, using an array of semiconductor junctions or capacitors. As the miniaturization of cameras continues they can be found in embedded systems like mobile phones, allowing them to receive vlc signals without additional hardware [9]. Cameras have the advantage of focusing on the transmission source by looking at specific pixels, so it becomes possible to select a specific VLC source or to ignore a noise source. As a receiver for VLC, cameras have the disadvantage of requiring more processing to retrieve the signal from the image sensor. Secondly, a bigger disadvantage is the frame rate which is limited by the camera's (electronic or mechanic) shutter speed, which limits the signal sample rate and thus the data transfer rate. There are high speed cameras which offer a solution to this problem, but currently they are too big and power hungry to use in embedded devices.

2. Photodiodes and phototransistors work based on the same principle but their assembly is different: photodiodes have only one metal junction and transistors have

two. This semiconductor junction is exposed through a transparent casing, so photons can reach the junction and, when having the right frequency, excite electron-hole pairs which results in a current and voltage. Because of the big surface of the semiconductor junction, photodiodes and phototransistors are very sensitive to light. When comparing photodiodes to phototransistors, diodes are faster due to the single junction giving it a fast response time. On the other hand, phototransistors have a bigger signal gain which results in a stronger electric signal. A disadvantage of photodiodes is the dedicated circuitry needed for amplification, filtering and sampling to be able to receive a clear electronic signal. Advances in technology are capable of integrating these into a single chip which can alleviate this disadvantage.

3. LEDs can be used as light sensors. An attentive reader may have noticed that photodiodes and LEDs exploit an oppose effect: while in photodiodes photons that strike the material generate a flow of electrons, in LEDs a flow of electrons release photons (light). This reversed and complementary effect can be used to make an LED both a receiver and a transmitter of VLC [10].

   Considering the various options for receivers, we opted for photodiodes because they have the higher potential to achieve high data rates, are more sensitive than other sensors.

2.2.3 Run-length limited coding

Line coding technique are used to send arbitrary data over a communications channel with bandwidth limits. It bounds the length of stretches (runs) of repeated bits during which the signal does not change If the runs are too long clock recovery is difficult, while if they are too short, the high frequencies components that are more attenuated by the communications channel.

IEEE 802.15.7 standard suggests different RLL codes for VLC PHY I, that can be used with OOK or VPPM modulation.

2.2.3.1 Manchester

Manchester coding (also known as phase encoding, or PE) is a line code in which the encoding of each data bit has at least one transition and occupies the same time. It therefore has no DC component, and is self-clocking, which means that it may be inductively or capacitively coupled, and that a clock signal can be recovered from the encoded data.

It has a code rate of 1/2, maximum run-length of 2. It is commonly used at 10Base-T Ethernet or RFID and has been proposed in the VLC standard to avoid the flickering issue.

bit	symbol
0	01
1	10

TABLE 2.1: Manchester RLL code

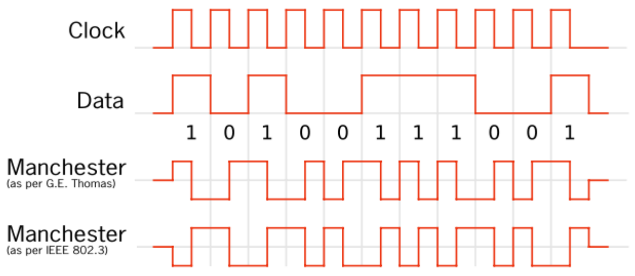


FIGURE 2.1: Manchester encoding

2.2.3.2 4B6B

4B6B expands 4-bit codes to 6-bit symbols with same ratio of 1 and 0 (3:3) and 50 percent duty cycle It supports clock recovery and DC-balanced waveform.

4B6B main advantages are :

- Free from intra-frame flickering
- Error detection

- Easier clock recovery
- DC balancing
- Short run length

4 bits	6 bits
0000	001110
0001	001101
0010	010011
0011	010110
0100	010101
0101	100011
0110	100110
0111	100101
1000	011001
1001	011010
1010	011100
1011	110001
1100	110010
1101	101001
1110	101010
1111	101100

TABLE 2.2: 4B6B RLL code

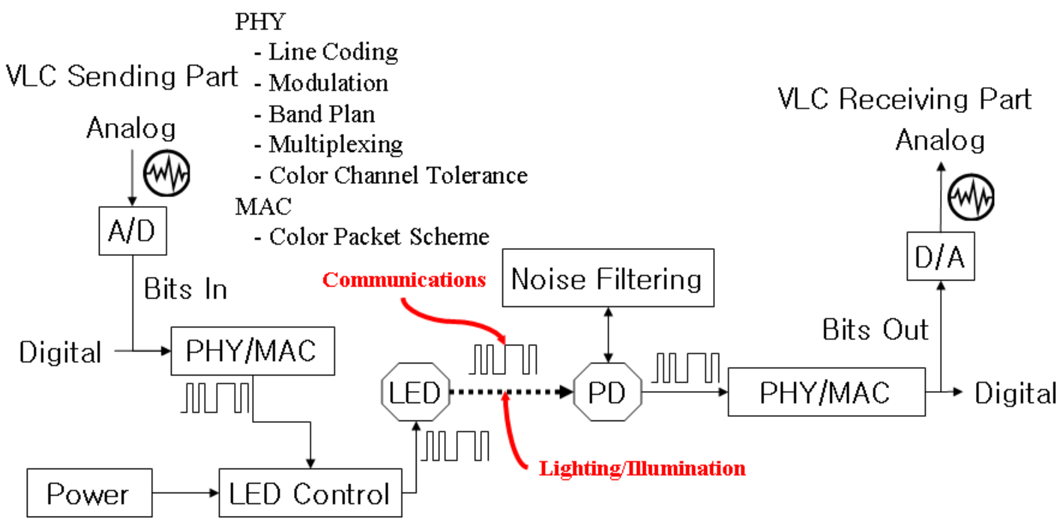


FIGURE 2.2: Visible Light Communications system

2.3 Ara Platform

2.3.1 Introduction to the Ara project

Project Ara is the codename for an initiative that aims to develop an open hardware platform for creating highly modular smartphones. The platform will include a structural frame or endoskeleton that holds smartphone modules of the owner's choice, such as a display, camera or an extra battery. It would allow users to swap out malfunctioning modules or upgrade individual modules as innovations emerge, providing longer lifetime cycles for the handset, and potentially reducing electronic waste. Project Ara smartphone will begin pilot testing in Puerto Rico later 2015 with a target bill of materials cost of $50 for a basic grey phone. The project was originally headed by the Advanced Technologies and Projects team within Motorola Mobility while it was a subsidiary of Google. Although Google had sold Motorola to Lenovo, it is retaining the project team who will work under the direction of the Android division.

2.3.2 Project goals

Google says the device is designed to be utilized by "6 billion people"; including 1 billion current smartphone users, 5 billion feature phone users, and 1 billion future user not currently connected. Google intends to sell a starter kit where the bill of materials is $50 and includes a frame, display, battery, low-end CPU and WiFi.

Google wants Project Ara to lower the entry barrier for phone hardware manufacturers so there could be "hundreds of thousands of developers" instead of the current handful of big manufacturers. This would be similar to how the Google Play Store is structured. Lowering the barrier for entry allows many more people to develop modules. Anyone will be able to build a module without requiring a license or paying a fee.

2.3.3 Structure and features

Ara Smartphones are built using modules inserted into metal endoskeletal frames known as "endos". The frame will be the only component in an Ara Smartphone made by Google. It acts as the switch to the on-device network linking all the modules together.

Two frame sizes will be available at first: ”mini”, a frame about the size of a Nokia 3310 and ”medium”, about the size of a LG Nexus 5. In the future, a ”large” frame about the size of a Samsung Galaxy Note 3 will be available. Frames have slots on the front for the display and other modules. On the back are additional slots for modules. The data from the modules can be transferred at up to 10 Gbps connection. The 2x2 modules have two connections and will allow up to 20 Gbps. However, this stack isn't yet included in the Ara Development Kit.

Modules can provide common smartphone features, such as cameras and speakers, but can also provide more specialized features, such as medical devices, receipt printers, laser pointers, pico projectors, night vision sensors, or game controller buttons. Each slot on the frame will accept any module of the correct size. The front slots are of various heights and take up the whole width of the frame. The rear slots come in standard sizes of 1x1, 1x2 and 2x2. Modules can be hot-swapped without turning the phone off. The frame also includes a small backup battery so the main battery can be hot-swapped.Modules are secured with electropermanent magnets. The enclosures of the modules were planned to be 3D-printed, but due to the lack of development in the technology Google opted instead for a customizable molded case.

Modules will be available both at an official Google store and at third-party stores. Ara Smartphones will only accept official modules by default, but users can change a software setting to enable unofficial modules. This is similar to how Android handles app installations.

2.3.4 Project team

Project Ara was developed and is led by Paul Eremenko from Google ATAP. The core Project Ara team at Google consists of three people with most of the work being done by outside contractors. One of the main contractors is NK Labs, a Massachusetts-based engineering firm, whose co-founder is Ara Knaian after whom the project was named. Another contractor is 3D System.

2.3.5 History and development process

The first version of the developers' kit (MDK v0.1x) relies on a prototype implementation of the Ara on-device network using the MIPI UniPro protocol implemented on FPGA and running over an LVDS physical layer with modules connecting via retractable pins. The second version (MDK v0.2) is built around a much more efficient and higher performance ASIC implementation of UniPro, running over a capacitive M-PHY physical layer.

2.3.6 Module Development Kit Architecture

The Module Developper Kit (MDK) we will consider is the v0.11, even if a new one - v0.2 - has been released in January, just after the Google Ara Developer Conference. The kit contains three dev boards and a bunch of SMA cables. The dev boards are an Application Processor (AP) board, a Generic Endpoint Dev Board and an Endoskeleton Switch Dev Board.

2.3.6.1 Android AP board

The Application Processor Board is a development kit for the Android operating system giving convenient way to modify or update the OS. It's part of the core of the Ara smartphone.

Board processor is an ARMv7 and the operating system base is on Android Tegra 3.10, and provide the SDK API 18, for the Android application developers.

It should be wired to the Endoskeleton Switch Board with 8 SMA coaxial cables

2.3.6.2 Endoskeleton Switch Dev Board

This board is the development kit for the Endoskeleton on which each module will be connected. It bridges the application processor and the modules using an UniPro high-speed interface. The Ara team also introduced a new software protocol called Greybus, which handles the communications between the endoskeleton and the module at an

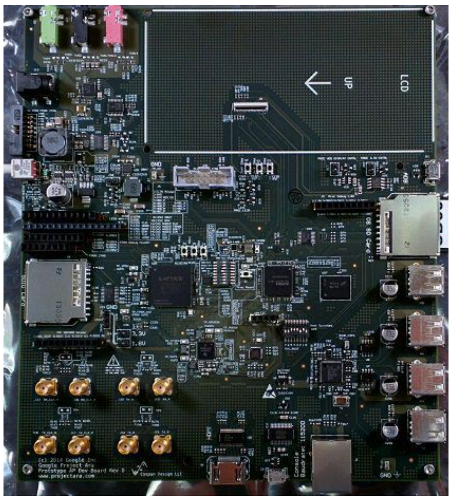


Figure 2.3: Ara Application Processor Board

higher level, making a convenient way for module developers to manipulate the UniPro stack.

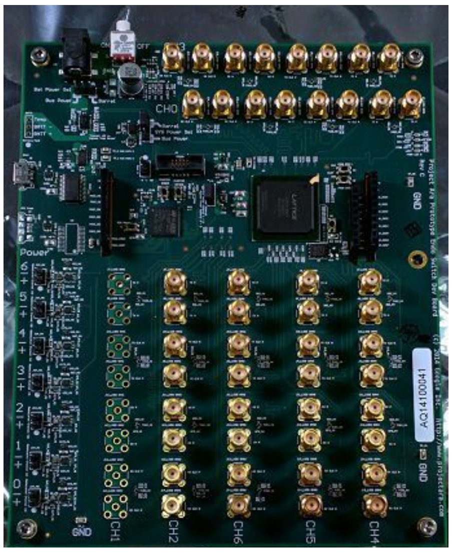


Figure 2.4: Ara Endoskeleton Switch Dev Board

2.3.6.3 GP Endpoint board

The Generic Endpoint is a development board to quickly prototype module. In fact, it's a kind of "base" module on which we would be able wiring our hardware by using standard connectors and bus. The board provides a set of Input/Output such as GPIO, I^{2}C or 5v and 3.3V output.

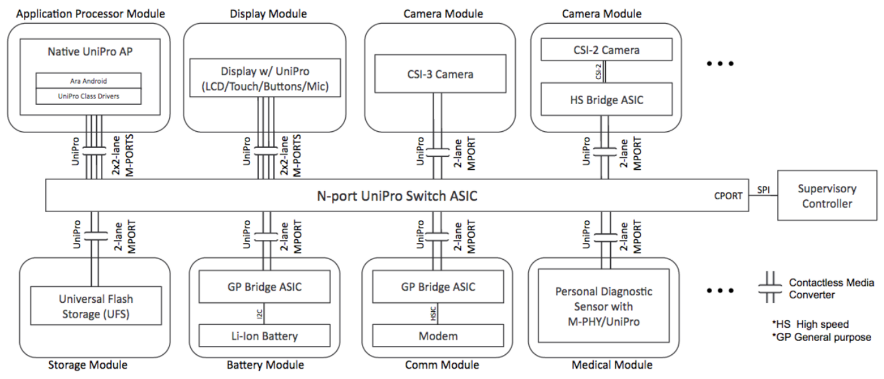


FIGURE 2.5: Module to Module Communication (with Native UniPro Support and Bridge ASICs)

Like the AP Board, it should be connected to the Endoskeleton with 8 SMA coaxial cables.

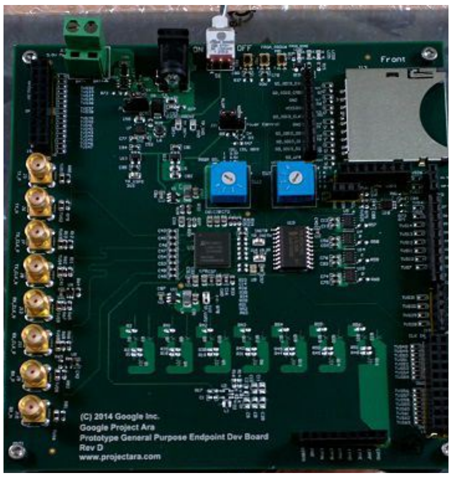


FIGURE 2.6: Ara Generic Endpoint Board

2.3.6.4 MDK Configuration and Setup

See the document in Appendix **??**

Chapter 3

State of the Art

This chapter provides an overview of the topics that supplied the ideas for this thesis. The following sections examine the previous works which have been done on implementing Visible Light Communication technology.

3.1 VLC Implementations

3.1.1 Visible Light Road-to-vehicle Communication Using High-Speed Camera

LEDs are already being used in traffic lights, and they can be used as the communication medium. Road-to vehicle communication using the LEDs in the traffic signal lights and on-vehicle high-speed camera as the receiver was proposed [11].

The bellow figure 3.1 shows the basic usage of LED as a transmitter and CAMERA as a receiver. In this model, they mounted a camera before the front end of the car. The camera is used as the information receiver from traffic signal lights. The advantage of using the camera is that multiple data can be transmitted by the LEDs and received by High-speed cameras.

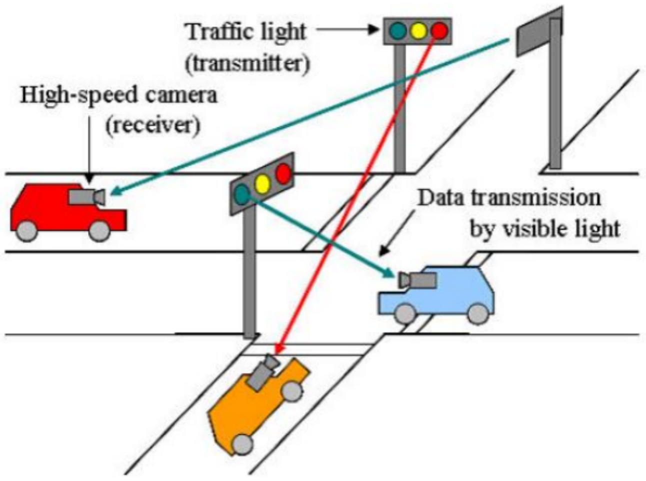


FIGURE 3.1: Road-to-vehicle visible light communication

3.1.2 Integrated System of White LED Visible-Light Communication and Power-Line Communication

In [2], optical communication using the existing power-line in a household is proposed as shown in 3.2 The power-line is used for communication between white LEDs and other fixed networks. The already installed power-lines and outlets behave as data networks and ports.

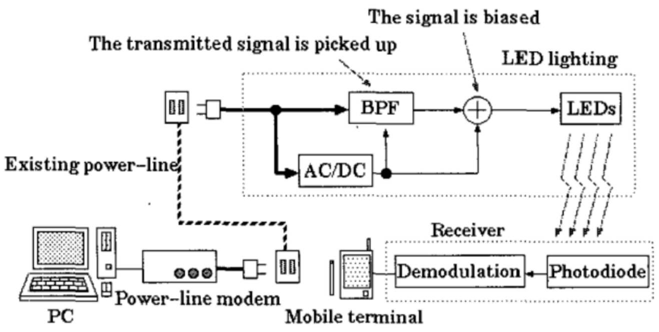


FIGURE 3.2: Waveform on power-line

As in optical intensity modulation, the transmitted signals are added to the cyclic waveform of the alternating current (AC). The transmitter signal from the PC is picked by BPF through the power-line, and biased before sending to the LED lights. The electrical signal is then converted into an optical signal by LEDs and sends it to the photodiode, where it converts the captured optical signal to an electrical signal. The signal is demodulated according to the received level of light and then is passed to the mobile terminal.

3.1.3 Visible Light Communication for Advanced Driver Assistant Systems

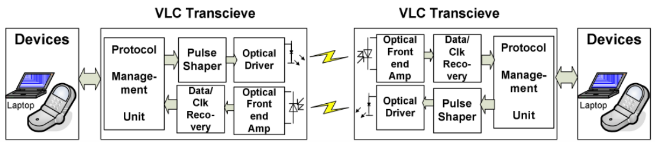


Figure 3.3: General architecture for a full duplex VLC system

Optical communications for outdoor communication has been discussed and elaborated upon [4]. Devices such as laptops an mobile phones can be used for transmitting and receiving information, using transceivers, as shown in 3.3. Transceiver systems use both LEDs and photodiodes. Intensity modulation was implemented to reach the most viable modulation. Various important design parameters were optimized by using intensive investigation based on gain variation over 100m of transmission range [4].

3.1.4 Study of Visible Light Communication System Using RGB LED Lights

Disney research group in Zurih [12] demonstrates a half-duplex VLC LED-to-LED communication with various applications with new generation toys. First example is a short-range directional communication with a number of toys, where front and backlights are used to exchange messages when pointed towards each other. On other application is a LED light bulb in a desk lamp receiving messages from a mobile device and broadcasts it back to the desk with higher light intensity and larger coverage. The static lamp acts as a repeater so that more devices can receive the original messages and the resulting network connectivity increases.

3.2 Smartphone solutions

Recently, the IEEE released a new task group called IEEE 802.15.7r1 OWC TG to write a revision to IEEE 802.15.7-2011 that accommodates infrared and near ultraviolet wavelengths. In addition to visible light, it also adds options such as optical camera communications, LED-ID (which is wireless light identification system using LED), Li-Fi (which is high-speed, bidirectional, networked mobile wireless communication using light). Some of their researches intend to bring VLC possibility to mobile phones. as visible light communication

3.2.1 Visible Light Communication using smartphone camera and rolling shutter effect

Cameras embedded in smartphones can be used as VLC receivers. As a result, common lights and smartphone cameras has the potential to enable a great number of applications with low cost. In [13], a prototype VLC system that utilizes undersampled frequency shift ON-OFF keying (UFSOOK) modulation is proposed. The system utilizes rolling shutter cameras as the receiver and takes advantages of its characteristics to improve the receiving performance. An LED is used as the transmitter. Information is transmitted in the continuous state (ON-OFF) changes of LEDs which are invisible to human eyes. The performance evaluation results demonstrate that the communication prototype is robust and can resist common optical interferences and noises within the image.

3.2.2 Visible Light Communication using smartphone ambient light sensor

Other approach as been proposed using the smartphone ambient light sensor as receiver. A light sensor is one of the most common sensors in smartphones, and is located on it's surface above the screen. Since the screen of a smartphone is a major factor in draining its battery, an ambient light sensor is used to recognize the brightness of its surroundings and adapt the screen backlight to save battery power while optimizing the visibility. Some researches leverage this sensor for indoor localization or secure authentication purpose. The It's the case of FIRE system presented in [14]. FIRE takes advantage of a smartphone's ambient light sensor and uses a challenge-based programmable light-emitting token generator. They have designed and prototyped an inexpensive passcode encoder and LED light-emitting hardware. Their experiments validated that FIRE can authenticate a user on a smartphone building a bi-directional communication channel based VLC.

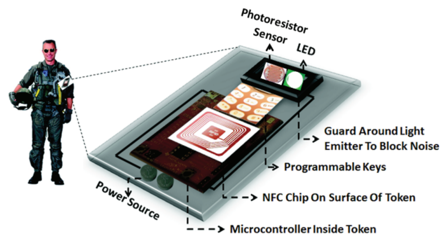

Photoresistor Sensor
LED
Guard Around Light Emitter To Block Noise
Programmable Keys
NFC Chip On Surface Of Token
Microcontroller Inside Token
Power Source

Figure 3.4: FIRE use case

Chapter 4

Proposal

In this chapter, we would introduce our proposal and contribution to current VLC researches.

4.1 System Description

Considering previous results and solutions, we propose an innovative solution, taking advantage of the Ara modular smartphone, to design a well suited front-end receiver for visible light communication.

As the Ara platform let use choose our own hardware and receiver, we would avoid problems and sampling limitations related to camera, or ambient light sensor. On other hand, the modular platform, through its UniPro interface on the endoskeleton board, give us the possibility to connect our module with the same performance than if it has been directly plugged in the main application processor board. In this way, we don't need to use a smartphone peripheral, such as USB, or Bluetooth controller, that would add a huge overhead in the transmission and processing.

The system, that we develop as part of this project consist in a commercial LED emitter, driven by a STM32L0 micro-controller unit, in order to transmit the OOK modulated signal. Before that, we propose to encode data using 4B6B or Manchester, in order to perform different experimentation and compare the result.

Considering the receiver module, our light sensor consist in a photodiode followed by several electronic circuit : current to tension converter, low-pass filter and positive gain amplifier.

The analog to digital conversion has been realized at 1,1 MSPS by the micro-controller unit. To fit the platform requirement and optimize the throughput, it performs several operations such as thresholding, decoding and buffering.

The transmission between the MCU and the Ara Development Kit is achieved through I2C bus in "slave-transmit" mode, with the MCU as slave-node and the Ara as master-node.

Finally, we develop a basic Android application for the platform, that implement I2C bus initialization, data polling, and logging for later computation and analysis.

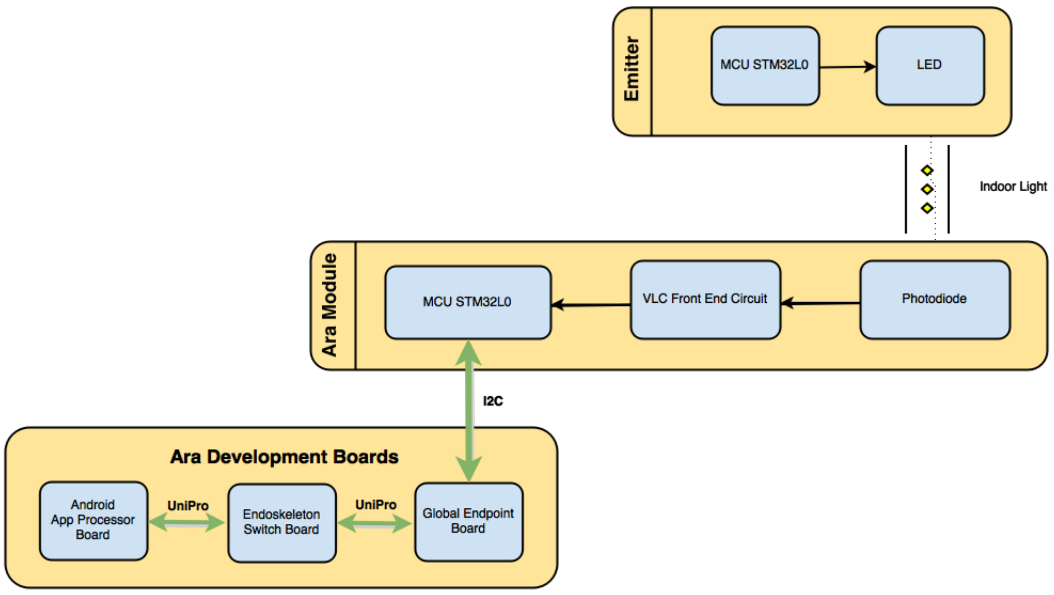


FIGURE 4.1: Proposal hardware implementation

4.2 VLC Contribution

Our approach would demonstrate a new VLC usage and possibility carrying an improved among of data the the user smartphone. Even if we follow IEEE 802.15.7 standard recommendation for PHY mode, we propose to combine OOK modulation with 4B6B coding, in order to improve emitted light power, avoid flickering, and improve the number of bit per symbol. In addition, we increase the emitter clock rate, up to 560kHz.

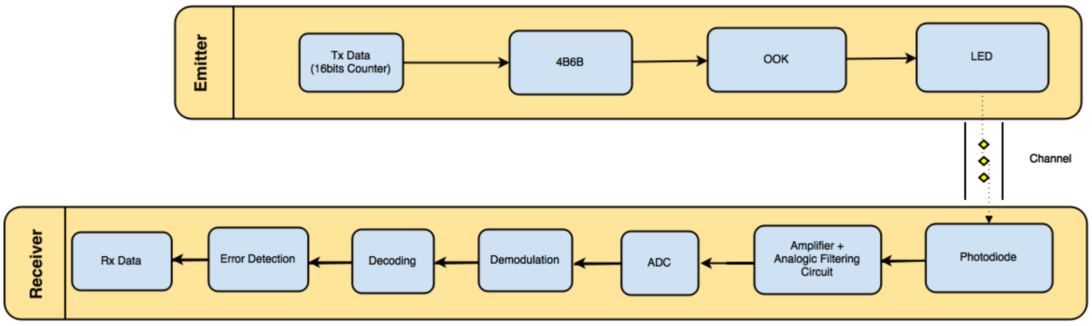


FIGURE 4.2: VLC system proposal

Chapter 5

Project Description

5.1 Ara platform study

In this section, we will describe different studies and preliminary experimentation we've done before designing the module.

In fact, because of the lake of literature or past experience with the Ara platform, we first need to study the Ara Module Development Kit send by Google ATAP. Our point of interest was understanding the role of each board, and there I/O - regarding protocols and speed. Then we determined the reliable rate on which one the operating system can execute a single task - such as performing an I^2C operation. Finally we did some benchmark and CPU load tests.

5.1.1 MDK and boards study

5.1.2 Android for Ara reliable I/O operation rate

5.1.2.1 Problematic

In this part, we would determine the I/O performance between the Ara main board and modules in order to fit its characteristics: maximum ADC sampling rate, resolution, buffering needs.

We focused on the minimum stable thread execution speed, which fix the number of Android I/O operation per second, then the maximum number of bytes we can read each time.

5.1.2.2 Experiments with the Oxymeter module and an Arduino

The Ara module development kit is shipped with a demo module. It consists in an oxymeter sensor that use infrared to determine the heart rate and blood oxygen. The application displays two graphs corresponding to the measured LED reflection at two different frequencies.

We modify the oxymeter application changing the thread execution interval from 1ms to 100ms and record the effective execution period using time-stamps. To have a consistent dataset, we record 1 million of samples for each measure. As consequence, we would be able to determine the minimum thread execution period, and its stability. This point is particularly important for designing the buffer on the VLC receiver : a bigger I/O operation period than expected will cause overflow, and data loss.

To determine the bus speed and the max effective transferred without error for each operation, we replaced the oxymeter module by an Arduino that will write different among of data on the bus. In addition, we use a logic analyzer to record what's happening on the bus.

5.1.2.3 Results

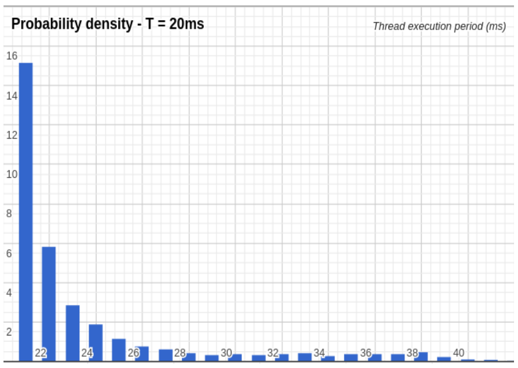


FIGURE 5.1: Probability density of Android thread execution interval with 20ms as defined period

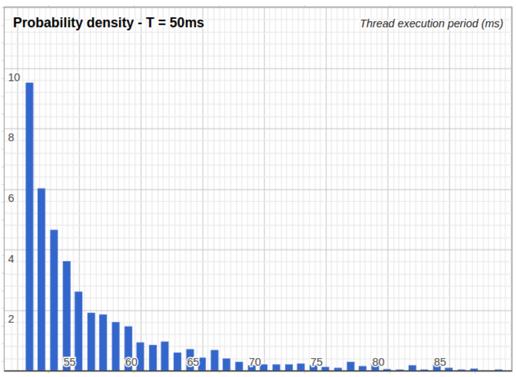


FIGURE 5.2: Probability density of Android thread execution interval with 50ms as defined period

Remark : as value lower than expected have no consequence, we don't plot them on the chart, but are taken in account in the probability repartitions.

- Android threads can't reach lower period than 8ms but with an elevate variance.
- Performing statistical analysis, for each execution period and plotting the effective period distribution and variance, we can consider 30ms as safe.
- Even if the I^{2}C bus speed clock rate should be 400kHz - according to the Ara MDK documentation - the effective clock rate is 133kHz. On other hand, even if the kernel driver limits I/O operations to 512 bytes, the bus gets corrupted trying to transfer more than 350 bytes per operation.

5.1.2.4 Conclusion

According to 5.1.2, we should design our VLC receiver module to taking account an effective bit-rate of 92,4 kbps and will developed the Android application with a module polling thread period to 30ms and an I2C transaction .

5.1.3 Ara benchmark

In addition to 5.1.2, we should be sure that a normal to extensive modular smartphone won't affect VLC module performances, even if the application run in background mode. In this case, the Android CPU governor will reduce the priority of our application, giving it less resources or stopping it if necessary.

In order to do that, we use a benchmark application that would simulate typical smartphone use-case : web browsing, short-message writing, video playing or games.

During the stress tests, both CPU load, CPU allocation, and RAM usage are monitored.

For each case, except the video game that made freeze and crash the Ara platform, results where exactly the same than with a single activity - ie. just our application - and we achieve the same bit-rate.

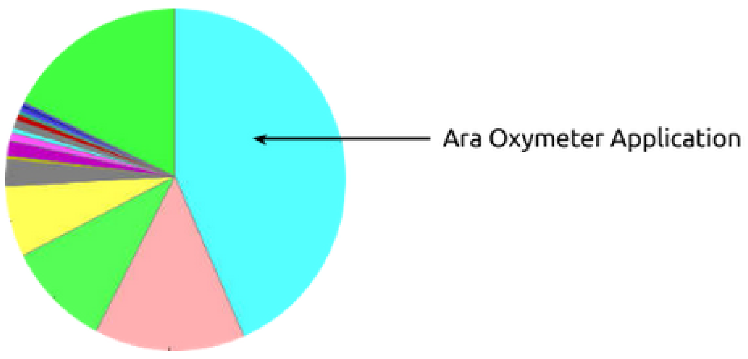


FIGURE 5.3: CPU load and allocation with only Oxymeter application running. Process on the left are system thread.

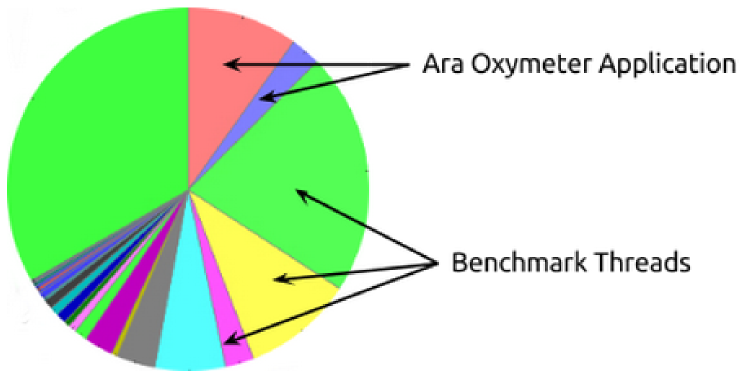


FIGURE 5.4: CPU load and allocation during the web browsing simulation

5.1.3.1 Conclusion

Following conclusion in5.1.2, we are now able to determine the analogical to digital converter characteristic. In order to get the better ratio as possible between the resolution, the ADC sampling rate and the LED driver clock rate, we chose values summarized in the next table.

Sampling Rate	$\approx$ 1 MSPS
Resolution	8-12 bits
Buffer Size	200 bytes

TABLE 5.1: DAC needs summary

5.2 Receiver circuit design

In this section, we will focus on the VLC module hardware design, that will have a major impact in the quality and efficiency of our system. Main difficulty we have to face here, is to select appropriate circuit pattern, filters type and cute-off frequency, according to the channel characteristic. The method we applied was in a first to study the channel principal characteristic, to design a first circuit, then compute theoretical values, and finally adjust them after some practical results.

5.2.1 Needs description

The VLC receiver front-end circuit must detect with high frequency precision the modulated light within a range starting from 1kHz to MHz. Before digitalization, the current generated by the photodiode has to be converted into current. As a major issue in VLC are interferences - from the ambient light or UV from the sun - the signal should be properly filtered and amplified before digitalization.

Our solution is constituted with a photodiode, witch convert the modulate light into current. It's followed by a trans-impedance amplifier in order to convert the photodiode output current into tension. Then, an analogical low-pass filter and gain are applied, before digitalization by the a micro-controller unit.

5.2.2 1st Op-Amp : Trans-Impedance Amplifier

The first part of the circuit is the Trans-Impedance Amplifier, presented in 5.5. TIA is a standard electronic pattern made from an operational amplifier and a resistor to convert the current to a tension.

We choose the MCP6292 Op-Amp for its 10 MHz wide band pass, and 3.3V input voltage. Indeed, the interesting input frequency from modulated light won't be greater than 1 MHz and we will be able supply it with the MCU. The DC output voltage due to the photodiode is given by 5.1

$$V_{OUT} = I_{DI} * R_1 \tag{5.1}$$

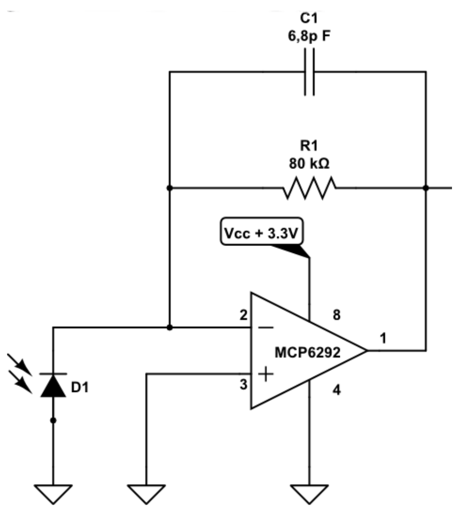


FIGURE 5.5: Trans-Impedance Amplifier

The capacitance in the feed-back loop is determined to compensate the photodiode capacitance in order to assure the operational amplifier stability which depends on C_1, R_1 and C_{D1}.

5.2.3 2nd: High Pass filter

After the Trans-Impedance Amplifier circuit part, we add a high-pass (figure 5.6 filter to attenuate the noise from the ambient lights such as sunlight and 50-120 kHz indoor fluorescent lights.

To perform, the filter was designed with a cut-off frequency of 200 Hz. We chose $C3$ and $R2$ value. According to figure 5.7 and equation 5.2, we use 10 nF capacitance and 48 kHΩ resistor

$$f_{cf} = \frac{1}{2\pi RC} \tag{5.2}$$

5.2.4 Gain

The output gain in the second part of the circuit aims to increase the signal amplitude, without increasing the noise. As a result, we will get better SNR, and digital signal processing and thresholding will be more accurate.

As for the TIA, a capacitance C_3 of $6.8pF$ has been added to assure the Op-Amp stability.

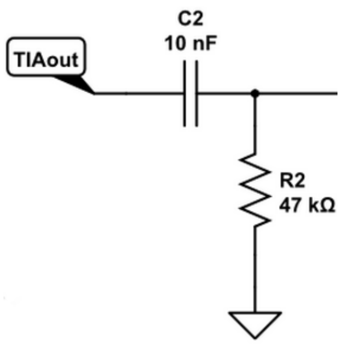


FIGURE 5.6: High-Pass Filter

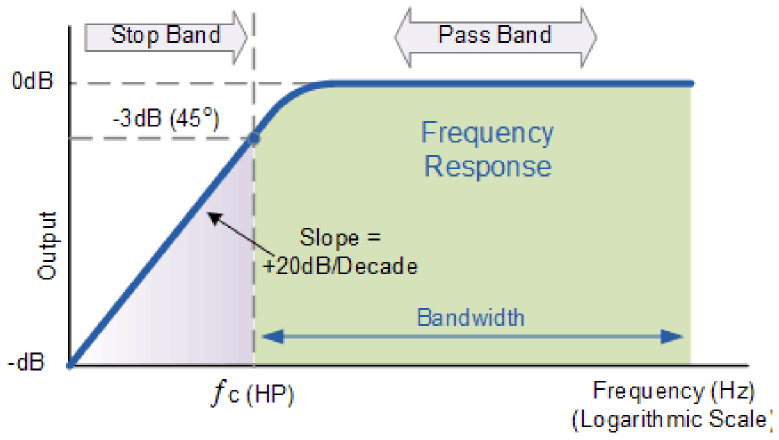


FIGURE 5.7: High-Pass Filter Bode Plot

The gain G_{OUT} has been adjusted during our final experiments with the modulated LED and its value is given by 5.3

$$G_{OUT} = \frac{R_4}{R_5} \tag{5.3}$$

5.3 Emitter driver

In this section, we will discuss the LED emitter and driver we developed for our system.

A commercial LED has been used as a modulated light source with an on-off signal generated by a micro-controller unit, that carry transmitted data encoded using a modified 4B6B run-length limited coding.

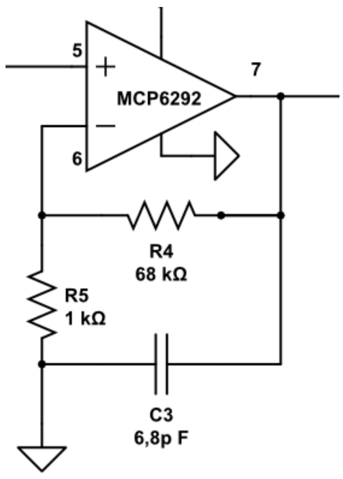


FIGURE 5.8: Gain Operational Amplifier

5.3.1 Micro-Controller Unit

The MCU used for the LED driver is an STM32L0, exactly the same as for the receiver, because its development kit is low cost, and toolchain with software development environment can be shared. We exploited its 3.3V digital output to blink the emitter white LED.

5.3.2 OOK modulation

Following IEEE 802.15.7 recommendations, we propose to use an On-Off Keying modulated signal as described in section 2.2.1 meaning that each bit 1 is mapped onto an high output level, and bit 0 mapped onto null output on the digital micro-controller output.

5.3.3 4B6B Line coding

The line code used is a modified version of the standard 4B6B described in 2.2.3 to increase the transmitted power. 4 bits are mapped into 6 bits always using 4 bit 1 and 2 bit 0. In that way, duty cycle is 2/3 instead of 1/2 for the standard version. This modified version would be used as error detection too.

To compare the performance between different RLL coding, Manchester has been implemented to, and both coding schema can be easily switched programmatically using a C macro.

4 bits	6 bits
0000	001111
0001	010111
0010	011011
0011	011110
0100	011110
0101	100111
0110	101011
0111	101101
1000	101110
1001	101111
1010	110011
1011	110101
1100	110110
1101	111001
1110	111010
1111	111100

Table 5.2: Modified 4B6B

5.3.4 Transmission Pattern

We propose a transmission pattern that would easily allow clock synchronization on the receiver. In addition, it has been designed taking account of the flickering and dimming issue.

Three transmissions state are defined :

- Idle : no data are transmitted, but we keep transmitting continuously **111000** and **000111** 4B6B symbols.
- Preamble : just before sending the data, the emitter should send a preamble, to let the receiver synchronizing. Symbols used should be different than these used to encode the data. We propose a combination of **110100** and **001011**.
- Data : 4B6B encoded data.

  The table 5.3 shows the transmission of the number 2, 16 bits encoded : 0000 0000 0000 0010.

TABLE 5.3: A transmission example : 2 16 bits

Idle	Preamble	Data	Idle
000111 111000 000111	110100 001011	001111 001111 001111 011011	000111 111000 000111

5.3.5 Algorithm

To evaluate our system, the micro-controller has been programmed as a 16bits counter, sending periodically - or using the development board button to random timing - the increased value, as defined in algorithm 1.

Algorithm 1 VLC Emitter main loop

```
$time \Leftarrow O$
$counter \Leftarrow O$
while 1 do
    if $time \geq 25$ then
        send preamble;
        send 4B6B counter value;
        $counter \Leftarrow counter + 1$
    else
        send idle;
    end if
    $time \Leftarrow time + 1$
end while
```

5.4 Receiver ADC and Buffer

In this section, we will describe the digitalization and digital processing performed by the VLC receiver module, just after the front-end circuit.

5.4.1 Hardware choice

According to 5.1, we chose to use the STM32L0 micro-controller based on an ARM Cortex M0+ processor with these characteristics :

- Processor Frequency up to 32 MHz: it will let us embedded some signal processing in order to reduce the computation on the Android.

- Low power consumption : 87 μA/MHz: it's a key feature for the mobile battery powered.
- Analogical Input with 1.1 MSPS ADC : it fits our requirements.
- Different I/O, including I^{2}C : it's a requirement to connect the MCU to the MDK endpoint board.
- 8kB RAM : this feature will let us implements the buffering using volatile memory (RAM).

5.4.2 Software implementation

We developed the MCU software in C, using the Arm GCC tool-chain and ST-Link debugger interface included in the Em::Blocks IDE [15].

The MCU is initialized setting the HSI as source clock with 32MHZ as processor frequency. Only I^{2}C, UART, ADC are enabled.

The main program works in interrupt mode.

5.4.2.1 Data Acquisition

The ADC sampling rate is set to 1,1 MSPS, in order to get the higher frequency precision, and the resolution to 8 bits, enough for OOK modulation.

The acquisition is done using buffering and Direct Memory Access (DMA) mode. When the DMA buffer is full, an interruption is thrown starting the demodulation and RLL decoding process.

5.4.2.2 Demodulation and RLL Decoding

First step is the the OOK demodulation that consisting in applying a threshold to the value.

```c
uint8_t ARA_ADC_Threashold(uint32_t voltage)
{
    if (voltage > 400) return 1;
    return (voltage > (minVal + DETECTION_THRESHOLD)) ? 1 : 0;
}
```

Listing 5.1: Thresholding implementation

As explain in 5.3.4, we should detect the preamble pattern - **110100001011** - that would help synchronize both emitter and receiver. Then as we know exactly the data frame and structure, we would be able to read the 4B6B encoded data.

5.4.2.3 Buffering

After being decoded , bits are pushed into a FIFO Buffer implemented in D

```c
typedef struct
{
    uint8_t *bufptr;    // pointer to the buffer array
    size_t bufsize;    // size of buffer
    size_t rdidx;    // points to next address to be read
    size_t wridx;    // points to next address to be written
#if(FIFO_LOG_MAX_USAGE == 1)
    size_t max;
#endif
} FIFO_t;
```

Listing 5.2: fifo.h

5.4.2.4 Input/output

1. UART peripheral is enabled only in debugging case or to record measures to the laptop via Serial-to-USB port.

2. I^{2}C is enabled in slave mode and connected to the Ara module development board. An interruption is thrown when a read request coming from the master peripheral driven by the Android application. is detected

   The function called in the interruption callback flush the last 350 bytes, to send them to the I^{2}c.

```c
void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef *I2CxHandle)
{
    Flush_Buffer((uint8_t*)aTxBuffer, aTxSize);
    RES_t res = FIFO_read(&AdcFIFO, &aTxBuffer, aTxSize);
}
```

LISTING 5.3: I^{2}c interrupt Callback

5.5 Ara Android App

In this part, we describe the Android application that we have developed, to support our VLC receiver module.

5.5.1 Application structure and operations

Our application as been developed using Android Studio IDE [16] with the Gradle build toolchain. The target SDK API version is number 18, to be compliant with the operating system version which has been installed on the AP Development Board.

The application package has been called respecting Java name convention :

```
package edu.upc.entel.wng.vlcAraModule;
```

This package includes 3 classes :

- VLCAraActivity : initialize the application and user interface. It surcharges the Activity class , as requested by the Android API.
- Sensor : define and perform operations to communicate with our module such as I^{2}C bus initialization, data polling as well as handling and message with the User Interface (UI).
- VlcLogger : this class realize logging operation saving received data on the board internal storage or an external SD Card.

Application workflow is quiet simple :

1. Initialize and start the application main Activity.

2. Setup the I^{2}C Bus.

3. Initialize the logger and create a log file.

4. Start the **Sensor** thread that poll I^{2}C bus at defined interval and send the result to the Android activity.

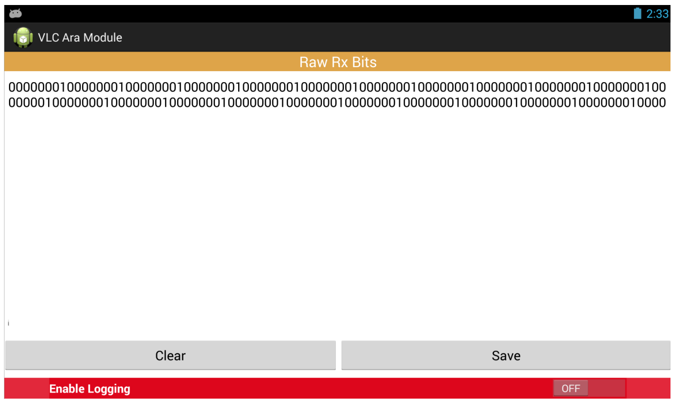
5. Handle Sensor thread messaging and user interaction.

5.5.2 User Interface

The application user interface as few components as defined in the XML activity layout description file :

- EditText field: used to display received bits.
- "Clear" and "Save" Button :

Each component are placed in a horizontal layout container.



```
0000000100000000100000000100000000100000000100000000100000000100000000100000000100000000100000000100
0000010000000010000000010000000010000000010000000010000000010000000010000000010000000010000000010000
```

Figure 5.9: Android application receiving data via VLC

5.5.3 I^{2}C JNI interface

As the Android Java API for the Ara Module Development Kit is the same as the standard API, we need to implement additional components in order to access the low level hardware such as the GPIO or the I^{2}C bus. The Ara MDK is provided with an operating system level driver, developed in C, that can be used to configure, read and write date, in a simple way. However, this C interface, can't be directly used through the Java API. In order to give the possibility to perform I^{2}C operations in our Android application, we develop a JNI interface that would wrap the C driver into a Java package. So we define 2 Java classes :

- I2CManager : it configures the I^{2}C bus, by using UNIX I^{2}C kernel driver and execute operations defined in I2CTransaction.
- I2CTransaction : it represents read or write operation on the bus.

Class methods are detailed in the appendix F.

Chapter 6

Results

In this chapter, we will describe and discuss our experimental results in order to validate our system and bring it to face to previous studies.

6.1 Experimental setup

We propose to evaluate different aspects and characteristics of our system considering an indoor communication situation avoiding sunlight perturbations.

In addition, we consider different cases: with or without ambient light interferences, Manchester or 4B6B coding, changing the clock rate and the emitter distance from the receiver.

6.2 Protocol

The receiver was placed in a fix position, while the LED light source keep mobile. In that way, the distance between the emitter and the receiver can be changed. To keep the measure consistent, we place a luxmeter, just near the receiver to approximate the effective illumination.

During the probes, both emitter and receiver micro-controller were connected with their debugging interface to a laptop, on a side to change the LED driver parameters, and on other side to visualize and record received data at different place of our system.

In addition, we used on oscilloscope to check the circuit output and control the signal generated by the emitter micro-controller.

The emitter light was alimented with a 28 Volts power supply and the LED Driver has been programmed to send periodically a 16-bits counter value, with the pattern defined in 5.3.4.

About the distance/illumination, probes have been realized in a range of 75 to 300 centimeters to match an indoor possible usage. On other side, the clock frequency has been limited to 560kHz, considering the ADC sampling rate - 1,1 MSPS - and the Nyquist-Shannon sampling theorem :

$$f_{Nyquist} = \frac{f_{sampling}}{2} \tag{6.1}$$

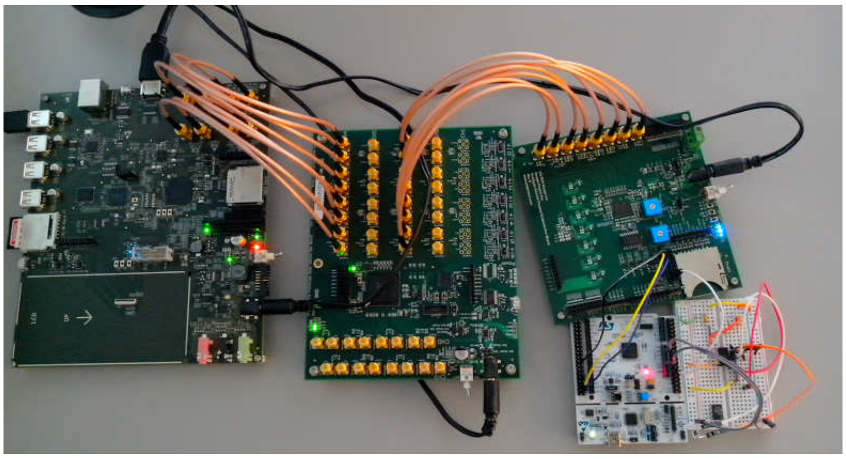


FIGURE 6.1: The Ara MDK and its VLC receiver module

6.3 Circuit Evaluation

The received signal was measured at different steps of the analogical processing, using the 2 oscilloscope voices at different points of our circuit. The high-pass filter and adjustments made on the receiver front-end circuit are satisfactory, and improve the signal quality before sampling.

sectionDigitalization and demodulation

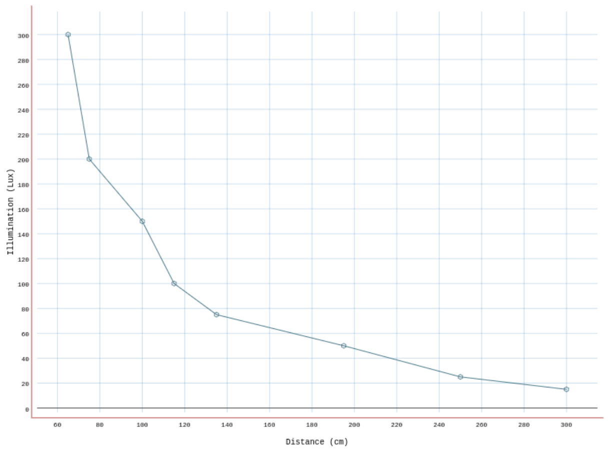
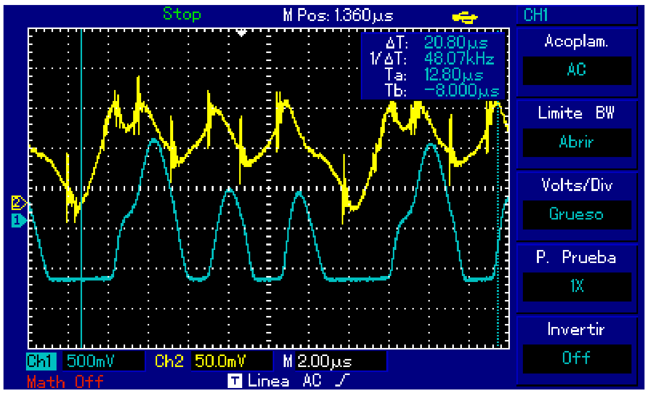


FIGURE 6.2: Receiver illumination over the distance


Stop M Pos: 1.360µs CH1

ΔT: 20.80µs
1/ΔT: 48.07kHz
Ta: 12.80µs
Tb: -8.000µs

Acoplam. AC
Limite BW Abrir
Volts/Div Grueso
P. Prueba 1X
Invertir Off

Ch1 500mV Ch2 50.0mV M 2.00µs
Math Off T Linea AC

FIGURE 6.3: In yellow the signal just after the TIA. In blue, the signal after analogical filtering - 2,5 meter distance

In order evaluate our demodulation and digital processing algorithms, we plot the sampled signal and the bits obtained after computation. We can see in the figure 6.4 that it is correct and we can get rid of signal attenuation due to circuit latency.

6.4 Bit Error Rate

In digital transmission, the number of bit errors is the number of received bits of a data stream over a communication channel that have been altered due to noise, interference, distortion or bit synchronization errors.

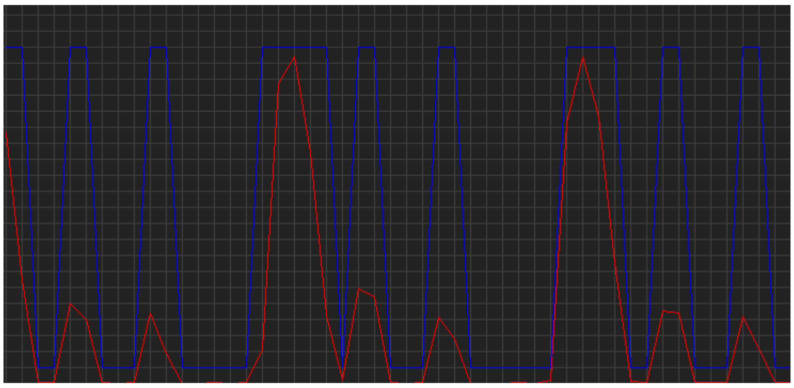


FIGURE 6.4: In red the signal sampled. In blue, the bits computed after digital processing and demodulation

As we know exactly the sequence of bit that have been emitted and the decoded sequence, we were able to compute it, for different illumination level and clock-rate, using Manchester or 4B6B coding.

6.4.1 4B6B

clock rate (kHz)	bitrate (kbps)	BER
100	67	$< 10^-4$
160	107	$< 10^-4$
240	160	3.10^-4
380	253	6.10^-4
560	373	2.10^-2

TABLE 6.1: Bit Error Rate for different clock rate at 2,5 meters, using 4B6B coding

Considering table 6.1 2,5 meters distance and 10^-3 as admissible BER, we can assume that using a 380kHz clock rate can be used for a correct transmission using 4B6B.

In that case, with 4B6B coding, we obtain a raw throughput of 253 kbps.

6.4.2 Manchester

Using Manchester coding, we obtained a surge of the bit error rate when the illumination is lower than 100 Lux. Analyzing the signal at the ADC input, we put in evidence that this is due to wave attenuation : isolated bit, just after long sequences are not detected, as visible in the figure 6.5.

clock rate (kHz)	bitrate (kbps)	BER
100	50	$< 10^{-}4$
160	80	$< 10^{-}4$
240	120	$8.10^{-}4$
380	190	$4.10^{-}2$
560	280	$> 10^{-}1$

Table 6.2: Bit Error Rate for different clock rate at 2,5 meters, using Manchester coding

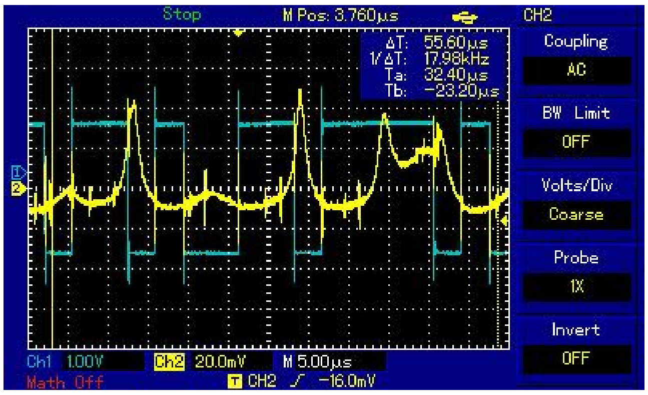

Stop M Pos: 3.760µs CH2
ΔT: 55.60µs
1/ΔT: 17.98kHz
Ta: 32.40µs
Tb: −23.20µs
Coupling AC
BW Limit OFF
Volts/Div Coarse
Probe 1X
Invert OFF
Ch1 1.00V Ch2 20.0mV M 5.00µs
Math Off T CH2 −16.0mV

Figure 6.5: In blue, the signal generated by the LED driver. In yellow, the signal after our circuit - 2,5 meter distance - 360kHz

6.5 Discussion

Regarding previous results, we were able to validate our proposal. We can achieve a reliable transmission between a commercial LED and the modular smartphone at a suitable distance for indoor usage.

Comparing our module for Ara platform with previous system, such [17], or other proposal using the smartphone camera and rolling shutter effect [13], we obtained and recorded an higher bitrate, up to 253 kbps. [17]

Chapter 7

Conclusion

7.1 Achieved work

In this project, we demonstrate the new opportunity and usage that the recent concept of modular smartphone and Ara platform give to the smartphone, extending its hardware to support visible light communication.

We finally develop a VLC receiver module for the Ara smartphone using the development kit provided by Google, with the full communication chain - including the emitter LED driver - in order to test it.

In this way, we were able to give our contribution to the VLC research, probing a modified version of 4B6B runlength-limited coding associated with the On-Off Keying modulation.

Bringing a suitable and dedicated receiver to the smartphone, we overcome past results in term of bitrate, in an indoor transmission scenario, between a lightening LED and a smartphone.

7.2 Future works

On the one hand, regarding our work and results, we might improve them in different ways. First the circuit need a better expertise to increase it's stability, or interferences tolerance, and obtain a better SNR ratio. Then, the release of the MDK v0.2 and the new Spiral 2 development board by Google ATAP and the Ara project team, make our work on this board quiet obsolete as it's totally different from the commercialized Ara smartphone.

On other hand, as we didn't develops any protocol at the Medium-Access-Control (MAC) or application layer for our VLC system, a lot of work keep underachieved and need to be investigate in further research projects.

Finally, this proof of concept of a novel wireless communication module for the Ara smartphone, open the door to further modules development to bring M2M and IoT protocols such 6LoWPAN or Zigbee to the mobile phones.

Bibliography

[1] Xiaohan Liu, H. Makino, and Y. Maeda. Basic study on indoor location estimation using visible light communication platform. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, pages 2377–2380, Aug 2008. doi: 10.1109/IEMBS.2008.4649677.

[2] Honglei Li, Xiongbin Chen, Junqing Guo, Zongyu Gao, and Hongda Chen. An analog modulator for 460 mb/s visible light data transmission based on ook-nrs modulation. *Wireless Communications, IEEE*, 22(2):68–73, April 2015. ISSN 1536-1284. doi: 10.1109/MWC.2015.7096287.

[3] Xiaoxue Ma, Kyujin Lee, and Kyesan Lee. Appropriate modulation scheme for visible light communication systems considering illumination. *Electronics Letters*, 48(18):1137–1139, August 2012. ISSN 0013-5194. doi: 10.1049/el.2012.2195.

[4] Jin-Yuan Wang, Jun-Bo Wang, Ming Chen, and Xiaoyu Song. Dimming scheme analysis for pulse amplitude modulated visible light communications. In *Wireless Communications Signal Processing (WCSP), 2013 International Conference on*, pages 1–6, Oct 2013. doi: 10.1109/WCSP.2013.6677037.

[5] S.E. Alavi, H. Rezaie, and A.S.M. Supa'at. Application of ofdm on integrated system of visible free space optic with plc. In *Applied Electromagnetics (APACE), 2010 IEEE Asia-Pacific Conference on*, pages 1–5, Nov 2010. doi: 10.1109/APACE.2010.5720114.

[6] S. Rajagopal, R.D. Roberts, and Sang-Kyu Lim. Ieee 802.15.7 visible light communication: modulation schemes and dimming support. *Communications Magazine, IEEE*, 50(3):72–82, March 2012. ISSN 0163-6804. doi: 10.1109/MCOM.2012.6163585.

[7] F.M. Wu, C.T. Lin, C.C. Wei, C.W. Chen, Z.Y. Chen, H.T. Huang, and Sien Chi. Performance comparison of ofdm signal and cap signal over high capacity rgb-led-based wdm visible light communication. *Photonics Journal, IEEE*, 5(4): 7901507–7901507, Aug 2013. ISSN 1943-0655. doi: 10.1109/JPHOT.2013.2271637.

[8] S. De Lausnay, L. De Strycker, J.-P. Goemaere, N. Stevens, and B. Nauwelaers. Matlab based platform for the evaluation of modulation techniques used in vlc. In *Development and Application Systems (DAS), 2014 International Conference on*, pages 57–61, May 2014. doi: 10.1109/DAAS.2014.6842427.

[9] Nam-Tuan Le, Trang Nguyen, and Yeong Min Jang. Frequency shift on-off keying for optical camera communication. In *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on*, pages 22–25, July 2014. doi: 10.1109/ICUFN.2014.6876741.

[10] S. Schmid, G. Corbellini, S. Mangold, and T.R. Gross. An led-to-led visible light communication system with software-based synchronization. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 1264–1268, Dec 2012. doi: 10.1109/GLOCOMW.2012.6477763.

[11] M. Wada, T. Yendo, T. Fujii, and M. Tanimoto. Road-to-vehicle communication using led traffic light. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 601–606, June 2005. doi: 10.1109/IVS.2005.1505169.

[12] S. Schmid, G. Corbellini, S. Mangold, and T.R. Gross. An led-to-led visible light communication system with software-based synchronization. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 1264–1268, Dec 2012. doi: 10.1109/GLOCOMW.2012.6477763.

[13] Peng Ji, Hsin-Mu Tsai, Chao Wang, and Fuqiang Liu. Vehicular visible light communications with led taillight and rolling shutter camera. In *Vehicular Technology Conference (VTC Spring), 2014 IEEE 79th*, pages 1–6, May 2014. doi: 10.1109/VTCSpring.2014.7023142.

[14] K. Dhondge, Baek-Young Choi, Sejun Song, and Hyungbae Park. Optical wireless authentication for smart devices using an onboard ambient light sensor. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8, Aug 2014. doi: 10.1109/ICCCN.2014.6911803.

[15] Em::Blocks. Em::blocks ide, March 2015. URL `http://www.emblocks.org`.

[16] Google Android. Android studio ide, March 2015.

[17] E. Sarbazi and M. Uysal. Phy layer performance evaluation of the ieee 802.15.7 visible light communication standard. In *Optical Wireless Communications (IWOW), 2013 2nd International Workshop on*, pages 35–39, Oct 2013. doi: 10.1109/IWOW.2013.6777772.

Appendix A

Ara development kit documentation

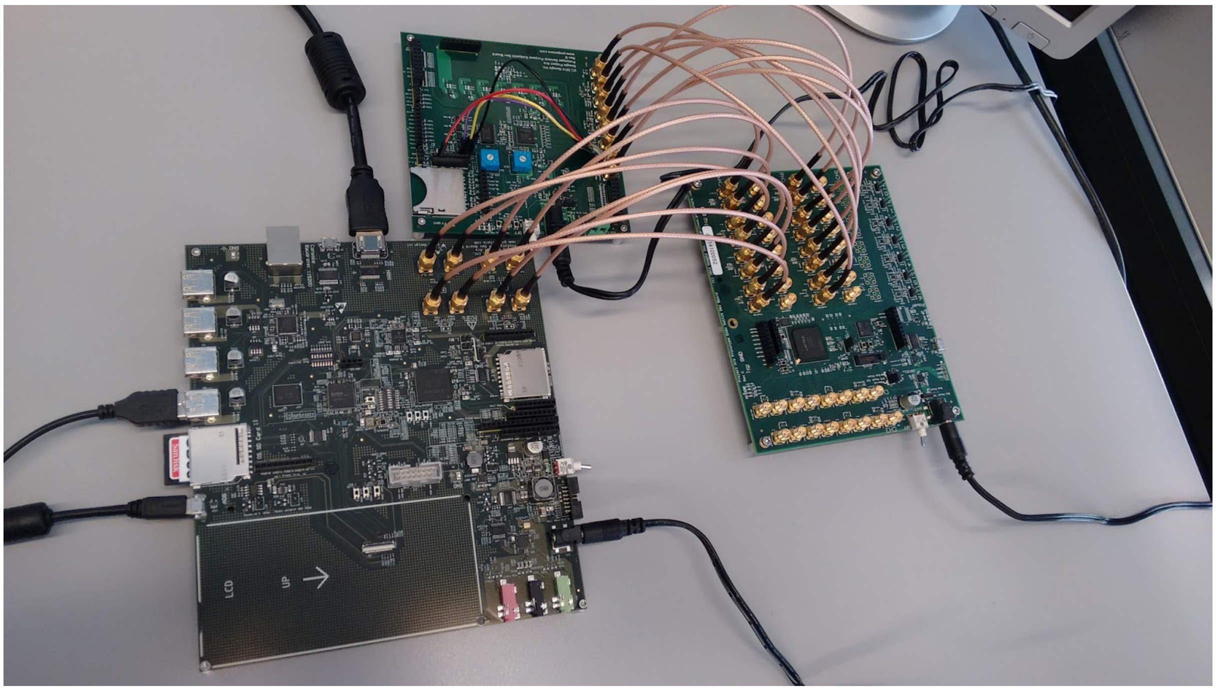
The ARA development kit is shipped with 3 components :

- Ara Development Board (rev. D) : that's the main board, embedding an ARMv7l processor, SD Card with a custom Android Operating System (v4.3)
- Generic Endpoint : it provides GPIO connectivity
- Endo Dev Board

Dev Board assembly and wiring

Endo Switch CH6 -> GPIO Extension Board
Endo Switch CH4 -> AP Android Board

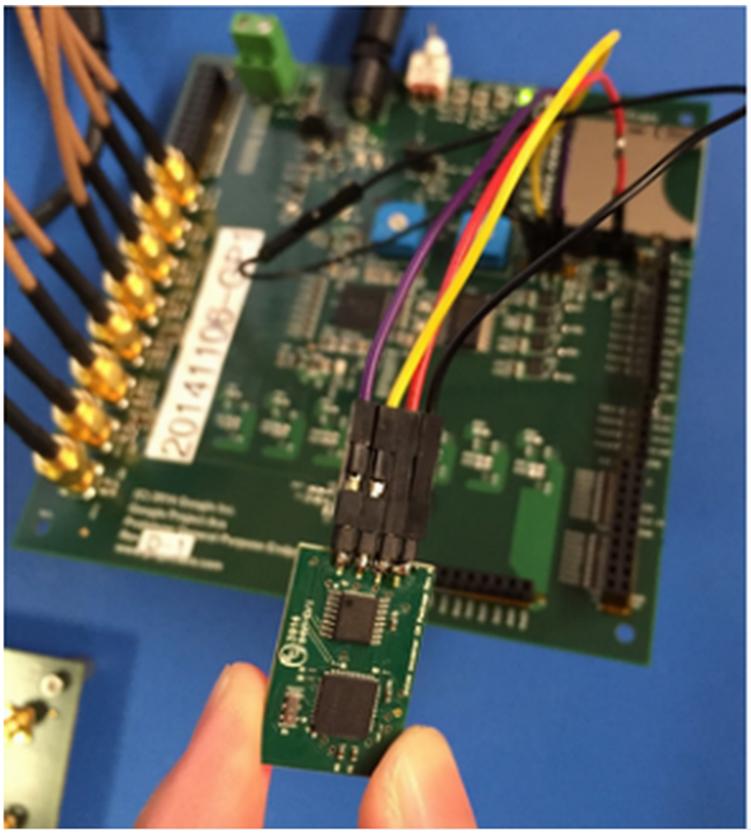
Tx -> Rx
N -> N and CLK_N -> CLK_N
P -> P and CLK_P -> CLK_P

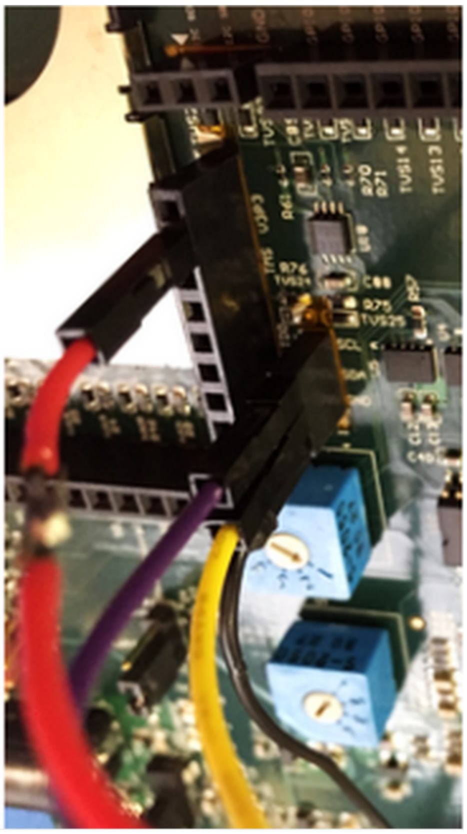


Pulse Oxymeter

From left (purple) to right (black), the connections : SCL, SDA, Power (3V3) and Ground.

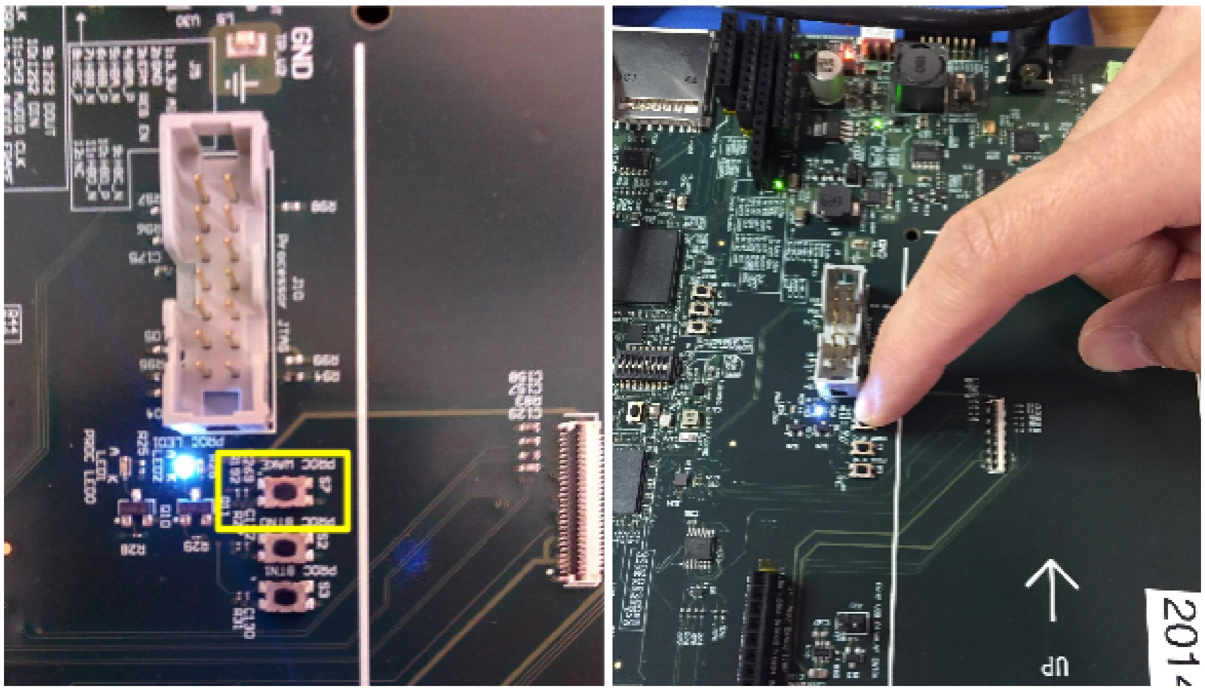
J15 Header : GND, SDA, and SCL
J5 header : 3V

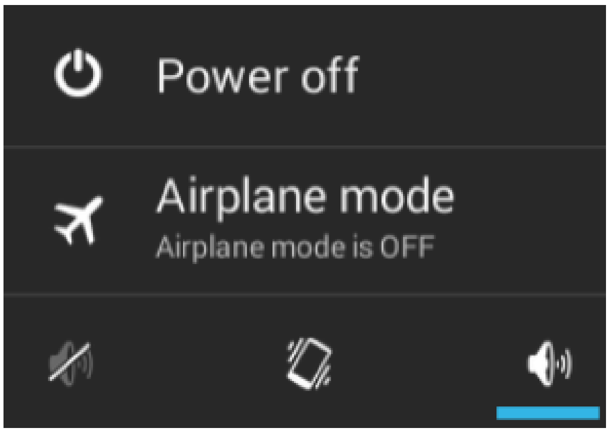




<u>Power Sequence</u>
1 Switch on the Endo Switch Board
2 Wait at least 15 seconds (to allow the switch to boot)
3 Switch on the GP Endpoint Board
4 Wait a few seconds for UniPro linkup
5 Switch on the AP Board

<u>Shutdown</u>
1 Power Off Android. Press during 3-4 seconds.
2 Once Android is down, no order importance.





Additional LCD Screen

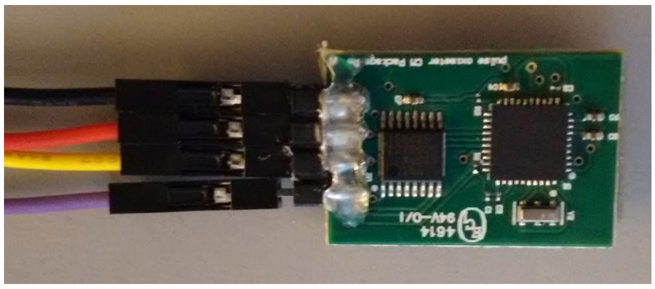
Moto X on the DSI port.

Speed Modulation, Byte Rate and Framing

- **system rate:** 20000 micro seconds = (https://ara-mdk.googlesource.com/platform/frameworks/base/+/master/core/java/android/hardware/SensorManager.java#624)
- **on the app** : 20 ms => 50hz

Used in the demo application provided with the DevKit that use infrared to get the heart rate.

Pulse Oximeter Sensor



- TI AFE4400 : Analogic front-end for PulseOx

http://www.ti.com/product/afe4400

➔ 4 Mhz clock
➔ PRP Count - Pulse repetition period count - Can be set from 800 to 64000. Set by the Android app to 7999.
➔ Max sampling rate = 1,3 KSPS
➔ Default sampling rate = 500 SPS
➔ Max résolution = 22 Bits

- **IS602B : I2C to SPI**

http://www.nxp.com/products/interface_and_connectivity/bridges/i2c_slave_to_spi_master_gpio_bridges/SC18IS602BIPW.html

➔ supports I²C data transfers up to 400 kHz.
➔ SPI master operating up to 1.8 Mbit/s
➔ 200-byte data buffer

Appendix B

Circuit Schematics

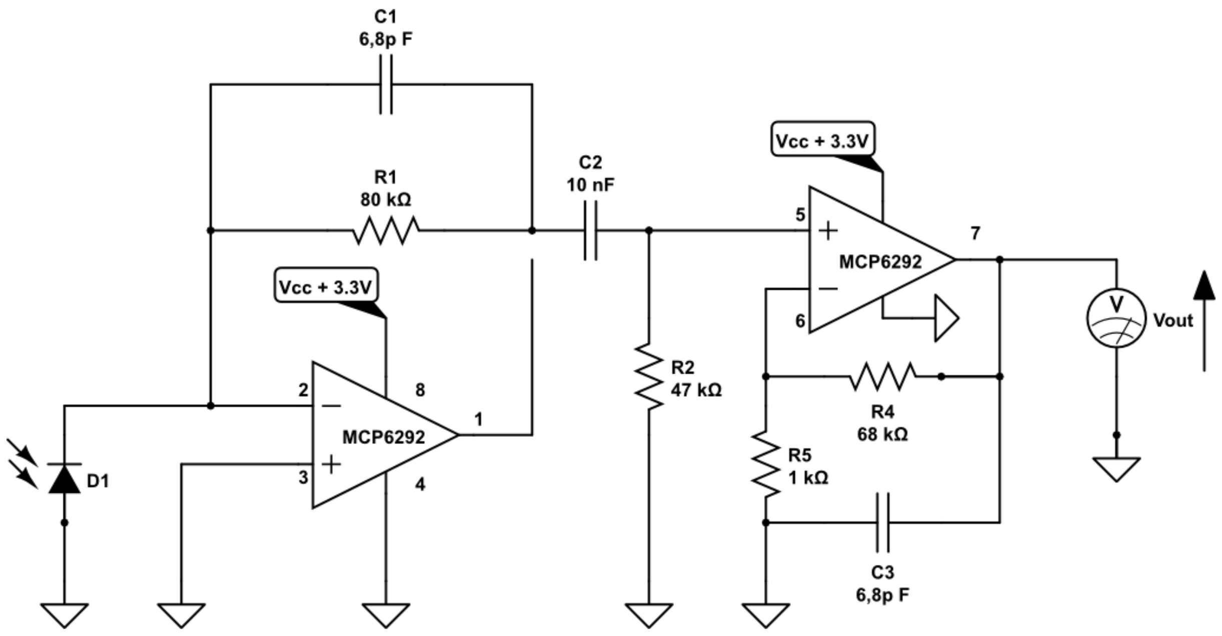


Figure B.1: VLC Receiver front-end circuit

Appendix C

Emitter software sources

#*

```
/**
    *************************************************************************
  * @file    Src/main.c
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-Avril-2015
  * @brief   VLC Emitter Driver
    *************************************************************************
  */

/* Includes
    -----------------------------------------------------------------*/
#include "main.h"
#include "4b6b.h"
#include "stm32l0xx_it.h"

/* Private typedef
    -----------------------------------------------------------*/
/* Private define
    ------------------------------------------------------------*/
#define BIT_PERIOD      1
//#define CODE_4B6B
```

```
/* Private macro
    -------------------------------------------------------------*/
/* Private variables
    ---------------------------------------------------------*/
/* Private variables
    ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

uint8_t aRxBuffer[RXBUFFERSIZE];
uint8_t message[] = "11011101";
uint16_t counter = 0;
int i = 0;
/* Private function prototypes
    -----------------------------------------------*/
void SystemClock_Config(void);
void GPIO_Init();
void sendManchesterMessage(void);
void sendManchesterCounter(void);
void send4B6BCounter();
void send4B6BEncoded6Bits(uint8_t bits);
void sendManchesterDefaultMessage(void);
void sendPreambule(void);
void sendStartBit(void);
void sendStopBit(void);
void sendNoData(void);
void delay (int a);
void send0(void);
void send1(void);
long time = 0;
//void USART2_UART_Init();

GPIO_InitTypeDef GPIO_InitStruct;

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    SysTick_Init();
    GPIO_Init();
    //USART2_UART_Init();
    BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);
    while (1)
    {
        time++;
        if (time > 10)
        {
            sendPreambule();
            //sendStartBit();
            send4B6BCounter();
            sendStopBit();
            counter++;
            time = 0;
        }
        else
        {
            sendNoData();
        }

    }
}


/**
  * @brief EXTI line detection callback.
  * @param GPIO_Pin: Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)


{
    if(GPIO_Pin == KEY_BUTTON_PIN)
    {
        sendPreambule();
        sendStartBit();
#ifndef CODE_4B6B
        sendManchesterDefaultMessage();
        //sendManchesterCounter();
#else
        send4B6BCounter();
#endif
        sendStopBit();
        counter++;
    }

}

void delay (int a)
{
    volatile int i,j;

    for (i=0 ; i < a ; i++)
    {
        j++;
    }

    return;
}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
void GPIO_Init(void)
{

    GPIO_InitTypeDef GPIO_InitStruct;

    /* GPIO Ports Clock Enable */
    __GPIOC_CLK_ENABLE();
    __GPIOH_CLK_ENABLE();
    __GPIOA_CLK_ENABLE();

    /*Configure GPIO pin : PC13 */
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    /* -2- Configure PA.5 IO in output push-pull mode to
```

```
           drive external LEDs */
    GPIO_InitStruct.Pin = (GPIO_PIN_5);
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FAST;

    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

void sendPreambule(void)
{
    send1();
    send1();
    send0();
    send1();
    send0();
    send0();

    send0();
    send0();
    send1();
    send0();
    send1();
    send1();
}

void sendStartBit(void)
{
    send0();
}

void sendStopBit(void)
{
    send0();
    send0();
    send1();
    send0();
    send1();
    send1();
}

```

```
void sendManchesterDefaultMessage(void)
{
    send1();
    send0();
    send1();
    send0();
    send0();
    send1();
    send1();
    send0();
    send1();
    send0();
    send1();
    send0();
    send0();
    send1();
    send1();
    send0();
}

void sendManchesterMessage()
{
    for(i=0; i<sizeof(message); i++)
    {
        switch(message[i])
        {
        case 0:
            send0();
            send1();
            break;

        case 1:
            send1();
            send0();
            break;

        default:
            break;
        }
    }
}
```

```
void sendManchesterCounter()
{
    for(i=16; i>0; i--)
    {
        if (counter & (1 << (i-1)))
        {
            send1();
            send0();
        }
        else
        {
            send0();
            send1();
        }
    }
}

void send4B6BCounter()
{
    int i = 0;
    uint8_t * splitedCounter = convertFrom16To4(counter);
    for(i=0;i<4;i++)
    {
        send4B6BEncoded6Bits(encode4b6b(splitedCounter[i]));
    }
    /*
    uint8_t first4b6b = encode4b6b(splitedCounter[0]);
    uint8_t sec4b6b = encode4b6b(splitedCounter[1]);
    uint8_t sec4b6b = encode4b6b(splitedCounter[2]);
    uint8_t sec4b6b = encode4b6b(splitedCounter[3]);
    send4B6BEncoded6Bits(first4b6b);
    send4B6BEncoded6Bits(sec4b6b);
    */
}

void send4B6BEncoded6Bits(uint8_t bits)
{
    int i = 0;
    for (i=6; i>0; i--)
    {
        if (bits & (1 << (i-1)))
        {
            send1();
        }
        else
        {
            send0();
        }
    }
}

void sendNoData()
{
    send1();
    send1();
    send1();
    send0();
    send0();
    send0();

    send0();
    send0();
    send0();
    send1();
    send1();
    send1();
}

void send1()
{
    GPIOA->BSRR = GPIO_PIN_5;
    delay(BIT_PERIOD);
}

void send0()
{
    GPIOA->BRR = GPIO_PIN_5 ;
    delay(BIT_PERIOD);
}

/** System Clock Configuration
*/
```

```
void SystemClock_Config(void)
{

    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;

    __PWR_CLK_ENABLE();

    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = 16;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLLMUL_4;
    RCC_OscInitStruct.PLL.PLLDIV = RCC_PLLDIV_2;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1);

    __SYSCFG_CLK_ENABLE();
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
}

#ifdef USE_FULL_ASSERT

/**
```

```
   * @brief Reports the name of the source file and the source line
    number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
    printf("Wrong parameters value: file %s on line %d\r\n", file, line);
}

#endif

#ifdef UART_ENABLED

/* USART2 init function */
void USART2_UART_Init(void)
{

    huart2.Instance = USART2;
    huart2.Init.BaudRate = 9600;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_RTS_CTS;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    huart2.Init.OneBitSampling = UART_ONEBIT_SAMPLING_DISABLED;
    huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    HAL_UART_Init(&huart2);

}

void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart2)
{
    while(1)
    {
    }
}

```

```
/**
  * @brief  Rx Transfer completed callback
  * @param  UartHandle: UART handle
  * @note   This example shows a simple way to report end of IT Rx
    transfer, and
  *         you can add your own implementation.
  * @retval None
  */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart2)
{
    /* Set transmission flag: trasfer complete*/
    HAL_UART_Transmit(huart2, (uint8_t*)&"OK\r\n", 6, 10);
}

#endif
/**
  * @}
  */

/**
  * @}
*/

/*****************************END OF FILE
    ************************************/
```

LISTING C.1: main.c

#*

```
/**

    ******************************************************************************

  * @file    Inc/main.h
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-Avril-2015
  * @brief   VLC Emitter Driver

    ******************************************************************************
  */

/* Define to prevent recursive inclusion
    -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

/* Includes
    ------------------------------------------------------------------*/
#include <stdint.h>
#include <stdio.h>
#include "stm32l0xx_hal.h"
#include "stm32l0xx_nucleo.h"
#include "stm32l0xx_it.h"

/* Exported types
    ------------------------------------------------------------*/
/* Exported constants
    --------------------------------------------------------*/
/* Definition for USARTx clock resources */
#define USARTx                          USART2
#define USARTx_CLK_ENABLE()             __USART2_CLK_ENABLE();
#define USARTx_RX_GPIO_CLK_ENABLE()     __GPIOA_CLK_ENABLE()
#define USARTx_TX_GPIO_CLK_ENABLE()     __GPIOA_CLK_ENABLE()

#define USARTx_FORCE_RESET()            __USART2_FORCE_RESET()
#define USARTx_RELEASE_RESET()          __USART2_RELEASE_RESET()

/* Definition for USARTx Pins */
#define USARTx_TX_PIN                   GPIO_PIN_9
#define USARTx_TX_GPIO_PORT             GPIOA
#define USARTx_TX_AF                    GPIO_AF4_USART1
#define USARTx_RX_PIN                   GPIO_PIN_10
#define USARTx_RX_GPIO_PORT             GPIOA
#define USARTx_RX_AF                    GPIO_AF4_USART1

/* Definition for USARTx's NVIC */
#define USARTx_IRQn                     USART2_IRQn
#define USARTx_IRQHandler               USART2s_IRQHandler

#define RXBUFFERSIZE                    10
```

```
/* Exported macro
    ------------------------------------------------------------*/
/* Exported functions
    ------------------------------------------------------- */
void Error_Handler(void);
extern UART_HandleTypeDef huart2;

#endif /* __MAIN_H */

/****************************END OF FILE
    **************************************/
```

Listing C.2: main.h

#*

```
/**
  
    ******************************************************************************

  * @file    Src/4b6b.c
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    16-May-2015
  * @brief   4B6B conversion
  
    ******************************************************************************

  */

/* Includes
    ------------------------------------------------------------------*/
#include "4b6b.h"
/* Private typedef
    -----------------------------------------------------------*/
/* Private define
    ------------------------------------------------------------*/
/* Private macro
    -------------------------------------------------------------*/
/* Private variables
    ---------------------------------------------------------*/
```

```
/* Private function prototypes
    -----------------------------------------------*/

uint8_t encode4b6b(uint8_t value)
{
    switch (value)
    {
    case 1:
        return CODE_4B6B_1;
        break;
    case 2:
        return CODE_4B6B_2;
        break;
    case 3:
        return CODE_4B6B_3;
        break;
    case 4:
        return CODE_4B6B_4;
        break;
    case 5:
        return CODE_4B6B_5;
        break;
    case 6:
        return CODE_4B6B_6;
        break;
    case 7:
        return CODE_4B6B_7;
        break;
    case 8:
        return CODE_4B6B_8;
        break;
    case 9:
        return CODE_4B6B_9;
        break;
    case 10:
        return CODE_4B6B_10;
        break;
    case 11:
        return CODE_4B6B_11;
        break;
    case 12:
        return CODE_4B6B_12;
        break;
    case 13:
        return CODE_4B6B_13;
        break;
    case 14:
        return CODE_4B6B_14;
        break;
    case 15:
        return CODE_4B6B_15;
        break;
    case 0:
        return CODE_4B6B_0;
        break;
    default:
        return CODE_4B6B_0;
    }
}

uint16_t convertFrom8To16(uint8_t dataFirst, uint8_t dataSecond) {
    uint16_t dataBoth = 0x0000;

    dataBoth = dataFirst;
    dataBoth = dataBoth << 8;
    dataBoth |= dataSecond;
    return dataBoth;
}

uint8_t *convertFrom16To4(uint16_t dataAll) {
    static uint8_t arrayData[4] = { 0x00, 0x00, 0x00, 0x00 };

    *(arrayData) = (dataAll >> 12) & 0x000F;
    arrayData[1] = (dataAll >> 8) & 0x000F;
    arrayData[2] = (dataAll >> 4) & 0x000F;
    arrayData[3] = (dataAll & 0x000F);
    return arrayData;
}
/**
  * @}
  */

```

```
/**
  * @}
*/

/*****************************END OF FILE
    ************************************/
```

Listing C.3: 4b6b.c

#*

```
/**
    ****************************************************************************
  * @file    Inc/4b6b.h
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    16-May-2015
  * @brief   4B6B conversion
    ****************************************************************************
  */

/* Define to prevent recursive inclusion
    -------------------------------------*/
#ifndef __4B6B_H
#define __4B6B_H

/* Includes
    ------------------------------------------------------------------*/
#include <stdint.h>
/* Exported types
    ------------------------------------------------------------*/
/* Exported constants
    --------------------------------------------------------*/
#define CODE_4B6B_0         (uint8_t)15
#define CODE_4B6B_1         (uint8_t)23
#define CODE_4B6B_2         (uint8_t)27
#define CODE_4B6B_3         (uint8_t)29
#define CODE_4B6B_4         (uint8_t)30
```

```
#define CODE_4B6B_5         (uint8_t)39
#define CODE_4B6B_6         (uint8_t)43
#define CODE_4B6B_7         (uint8_t)45
#define CODE_4B6B_8         (uint8_t)46
#define CODE_4B6B_9         (uint8_t)47
#define CODE_4B6B_10        (uint8_t)51
#define CODE_4B6B_11        (uint8_t)53
#define CODE_4B6B_12        (uint8_t)54
#define CODE_4B6B_13        (uint8_t)57
#define CODE_4B6B_14        (uint8_t)58
#define CODE_4B6B_15        (uint8_t)60

/* Exported macro
    ------------------------------------------------------------*/
/* Exported functions
    ------------------------------------------------------- */
uint16_t convertFrom8To16(uint8_t dataFirst, uint8_t dataSecond);
uint8_t *convertFrom16To4(uint16_t dataAll);
uint8_t encode4b6b(uint8_t value);
#endif

/*****************************END OF FILE
    **************************************/
```

Listing C.4: 4b6b.h

#*

```
/**

    ****************************************************************************

  * @file    Inc/stm32l0xx_hal_conf.h
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-Avril-2015
  * @brief   HAL configuration file.

    ****************************************************************************

  */

```

```

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32L0xx_HAL_CONF_H
#define __STM32L0xx_HAL_CONF_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/

/* ########################## Module Selection ############################## */
/**
  * @brief This is the list of modules to be used in the HAL driver
  */
#define HAL_MODULE_ENABLED
//#define HAL_ADC_MODULE_ENABLED
//#define HAL_COMP_MODULE_ENABLED
//#define HAL_CRC_MODULE_ENABLED
//#define HAL_CRYP_MODULE_ENABLED
//#define HAL_DAC_MODULE_ENABLED
//#define HAL_I2C_MODULE_ENABLED
//#define HAL_I2S_MODULE_ENABLED
//#define HAL_IWDG_MODULE_ENABLED
//#define HAL_LCD_MODULE_ENABLED
//#define HAL_LPTIM_MODULE_ENABLED
//#define HAL_RNG_MODULE_ENABLED
//#define HAL_RTC_MODULE_ENABLED
//#define HAL_SPI_MODULE_ENABLED
#define HAL_TIM_MODULE_ENABLED
//#define HAL_TSC_MODULE_ENABLED
#define HAL_UART_MODULE_ENABLED
#define HAL_USART_MODULE_ENABLED
//#define HAL_IRDA_MODULE_ENABLED
//#define HAL_SMARTCARD_MODULE_ENABLED
//#define HAL_SMBUS_MODULE_ENABLED
```

```
//#define HAL_WWDG_MODULE_ENABLED
//#define HAL_PCD_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED

/* ########################## Oscillator Values adaptation ####################*/
/**
  * @brief Adjust the value of External High Speed oscillator (HSE) used in your application.
  *        This value is used by the RCC HAL module to compute the system frequency
  *        (when HSE is used as system clock source, directly or through the PLL).
  */
#if !defined  (HSE_VALUE)
  #define HSE_VALUE    ((uint32_t)8000000) /*!< Value of the External oscillator in Hz */
#endif /* HSE_VALUE */

#if !defined  (HSE_STARTUP_TIMEOUT)
  #define HSE_STARTUP_TIMEOUT    ((uint32_t)5000)   /*!< Time out for HSE start up, in ms */
#endif /* HSE_STARTUP_TIMEOUT */

/**
  * @brief Internal Multiple Speed oscillator (MSI) default value.
  *        This value is the default MSI range value after Reset.
  */
#if !defined  (MSI_VALUE)
  #define MSI_VALUE    ((uint32_t)4194000) /*!< Value of the Internal oscillator in Hz*/
#endif /* MSI_VALUE */

/**
  * @brief Internal High Speed oscillator (HSI) value.
  *         This value is used by the RCC HAL module to compute the system frequency
  *         (when HSI is used as system clock source, directly or through the PLL).
  */
#if !defined  (HSI_VALUE)
  #define HSI_VALUE    ((uint32_t)16000000) /*!< Value of the Internal oscillator in Hz*/
#endif /* HSI_VALUE */

/**
  * @brief External Low Speed oscillator (LSE) value.
  *         This value is used by the UART, RTC HAL module to compute the system frequency
  */
#if !defined  (LSE_VALUE)
  #define LSE_VALUE    ((uint32_t)32000) /*!< Value of the External oscillator in Hz*/
#endif /* LSE_VALUE */

#if !defined  (LSE_STARTUP_TIMEOUT)
  #define LSE_STARTUP_TIMEOUT  ((uint32_t)5000)   /*!< Time out for LSE start up, in ms */
#endif /* LSE_STARTUP_TIMEOUT */

/* Tip: To avoid modifying this file each time you need to use different HSE,
   ===  you can define the HSE value in your toolchain compiler preprocessor. */

/* ########################### System Configuration ######################### */
/**
  * @brief This is the HAL system configuration section
  */
#define  VDD_VALUE                    ((uint32_t)3300) /*!< Value of VDD in mv */
#define  TICK_INT_PRIORITY            ((uint32_t)0)    /*!< tick interrupt priority */
#define  USE_RTOS                     0
#define  PREFETCH_ENABLE              1
```

```
#define  PREREAD_ENABLE               1
#define  BUFFER_CACHE_DISABLE         1

/* ########################## Assert Selection
   ############################## */
/**
  * @brief Uncomment the line below to expanse the "assert_param" macro
    in the
  *        HAL drivers code
  */
/* #define USE_FULL_ASSERT    1 */

/* Includes
   ------------------------------------------------------------------*/
/**
  * @brief Include module's header file
  */

#ifdef HAL_RCC_MODULE_ENABLED
  #include "stm32l0xx_hal_rcc.h"
#endif /* HAL_RCC_MODULE_ENABLED */

#ifdef HAL_GPIO_MODULE_ENABLED
  #include "stm32l0xx_hal_gpio.h"
#endif /* HAL_GPIO_MODULE_ENABLED */

#ifdef HAL_DMA_MODULE_ENABLED
  #include "stm32l0xx_hal_dma.h"
#endif /* HAL_DMA_MODULE_ENABLED */

#ifdef HAL_CORTEX_MODULE_ENABLED
  #include "stm32l0xx_hal_cortex.h"
#endif /* HAL_CORTEX_MODULE_ENABLED */

#ifdef HAL_ADC_MODULE_ENABLED
  #include "stm32l0xx_hal_adc.h"
#endif /* HAL_ADC_MODULE_ENABLED */

#ifdef HAL_COMP_MODULE_ENABLED
  #include "stm32l0xx_hal_comp.h"
#endif /* HAL_COMP_MODULE_ENABLED */
```

```

#ifdef HAL_CRC_MODULE_ENABLED
  #include "stm32l0xx_hal_crc.h"
#endif /* HAL_CRC_MODULE_ENABLED */

#ifdef HAL_CRYP_MODULE_ENABLED
  #include "stm32l0xx_hal_cryp.h"
#endif /* HAL_CRYP_MODULE_ENABLED */

#ifdef HAL_DAC_MODULE_ENABLED
  #include "stm32l0xx_hal_dac.h"
#endif /* HAL_DAC_MODULE_ENABLED */

#ifdef HAL_FLASH_MODULE_ENABLED
  #include "stm32l0xx_hal_flash.h"
#endif /* HAL_FLASH_MODULE_ENABLED */

#ifdef HAL_I2C_MODULE_ENABLED
 #include "stm32l0xx_hal_i2c.h"
#endif /* HAL_I2C_MODULE_ENABLED */

#ifdef HAL_I2S_MODULE_ENABLED
 #include "stm32l0xx_hal_i2s.h"
#endif /* HAL_I2S_MODULE_ENABLED */

#ifdef HAL_IWDG_MODULE_ENABLED
 #include "stm32l0xx_hal_iwdg.h"
#endif /* HAL_IWDG_MODULE_ENABLED */

#ifdef HAL_LCD_MODULE_ENABLED
 #include "stm32l0xx_hal_lcd.h"
#endif /* HAL_LCD_MODULE_ENABLED */

#ifdef HAL_LPTIM_MODULE_ENABLED
#include "stm32l0xx_hal_lptim.h"
#endif /* HAL_LPTIM_MODULE_ENABLED */

#ifdef HAL_PWR_MODULE_ENABLED
 #include "stm32l0xx_hal_pwr.h"
#endif /* HAL_PWR_MODULE_ENABLED */

```

```
#ifdef HAL_RNG_MODULE_ENABLED
 #include "stm32l0xx_hal_rng.h"
#endif /* HAL_RNG_MODULE_ENABLED */

#ifdef HAL_RTC_MODULE_ENABLED
 #include "stm32l0xx_hal_rtc.h"

#endif /* HAL_RTC_MODULE_ENABLED */

#ifdef HAL_SPI_MODULE_ENABLED
 #include "stm32l0xx_hal_spi.h"
#endif /* HAL_SPI_MODULE_ENABLED */

#ifdef HAL_TIM_MODULE_ENABLED
 #include "stm32l0xx_hal_tim.h"
#endif /* HAL_TIM_MODULE_ENABLED */

#ifdef HAL_TSC_MODULE_ENABLED
 #include "stm32l0xx_hal_tsc.h"
#endif /* HAL_TSC_MODULE_ENABLED */

#ifdef HAL_UART_MODULE_ENABLED
 #include "stm32l0xx_hal_uart.h"
#endif /* HAL_UART_MODULE_ENABLED */

#ifdef HAL_USART_MODULE_ENABLED
 #include "stm32l0xx_hal_usart.h"
#endif /* HAL_USART_MODULE_ENABLED */

#ifdef HAL_IRDA_MODULE_ENABLED
 #include "stm32l0xx_hal_irda.h"
#endif /* HAL_IRDA_MODULE_ENABLED */

#ifdef HAL_SMARTCARD_MODULE_ENABLED
 #include "stm32l0xx_hal_smartcard.h"
#endif /* HAL_SMARTCARD_MODULE_ENABLED */

#ifdef HAL_SMBUS_MODULE_ENABLED
 #include "stm32l0xx_hal_smbus.h"
#endif /* HAL_SMBUS_MODULE_ENABLED */

```

```
#ifdef HAL_WWDG_MODULE_ENABLED
 #include "stm32l0xx_hal_wwdg.h"
#endif /* HAL_WWDG_MODULE_ENABLED */

#ifdef HAL_PCD_MODULE_ENABLED
 #include "stm32l0xx_hal_pcd.h"
#endif /* HAL_PCD_MODULE_ENABLED */

/* Exported macro
   ------------------------------------------------------------*/
#ifdef  USE_FULL_ASSERT
/**
  * @brief  The assert_param macro is used for function's parameters
    check.
  * @param  expr: If expr is false, it calls assert_failed function
  *         which reports the name of the source file and the source
  *         line number of the call that failed.
  *         If expr is true, it returns no value.
  * @retval None
  */
  #define assert_param(expr) ((expr) ? (void)0 : assert_failed((uint8_t
    *)__FILE__, __LINE__))
/* Exported functions
   ------------------------------------------------------- */
  void assert_failed(uint8_t* file, uint32_t line);
#else
  #define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */

#ifdef __cplusplus
}
#endif

#endif /* __STM32L0xx_HAL_CONF_H */


/*****************************************************************/
```

Listing C.5: *stm32l0xx$_h$al$_c$onf.h*

#*

```c
/**
  ******************************************************************************
  * @file    Src/stm32l0xx_it.c
  * @date    05/04/2015 10:49:21
  * @author  Alexis Duque
  * @version V0.0.1
  * @brief   Interrupt Service routine
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32l0xx_hal.h"
#include "stm32l0xx.h"
#include "stm32l0xx_it.h"
#include "main.h"

#define BITRATE_570KHZ 32000000
#define BITRATE_285KHZ 32000000

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/

static __IO uint32_t sysTickCounter;
void EXTI4_15_IRQHandler(void);
extern UART_HandleTypeDef huart2;

/******************************************************************************/
/*            Cortex-M0+ Processor Interruption and Exception Handlers        */
/* ***************************************************************************** */

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
    TimeTick_Decrement();
    HAL_IncTick();
}

void SysTick_Init(void)
{
    /****************************************
     *SystemFrequency/1000      1ms         *
     *SystemFrequency/100000    10us        *
     *SystemFrequency/1000000   1us         *
     *****************************************/
    while (SysTick_Config(SystemCoreClock / 32000000) != 0)

    {
    } // One SysTick interrupt now equals 1us

}


/* ***************************************************************************** */
/*                 STM32L0xx Peripherals Interrupt Handlers                     */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32l0xx.s).                                                */
/* ****************************************************************************/
/**
  * @brief  This function handles UART interrupt request.
  * @param  None
  * @retval None
  * @Note   This function is redefined in "main.h" and related to DMA stream
  *         used for USART data transmission
  */
void USART2_IRQHandler(void)
{
    /* USER CODE BEGIN USART2_IRQn 0 */

    /* USER CODE END USART2_IRQn 0 */
    HAL_UART_IRQHandler(&huart2);
    /* USER CODE BEGIN USART2_IRQn 1 */

    /* USER CODE END USART2_IRQn 1 */
}

void EXTI4_15_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(KEY_BUTTON_PIN);
}

/**
 * This method needs to be called in the SysTick_Handler
 */
void TimeTick_Decrement(void)
{
    if (sysTickCounter != 0x00)
    {
        sysTickCounter--;
    }
}

void delay_nus(uint32_t n)
{
    sysTickCounter = n;
```

```c
    while (sysTickCounter != 0)
    {
    }
}

void delay_1ms(void)
{
    sysTickCounter = 1000;
    while (sysTickCounter != 0)
    {
    }
}

void delay_nms(uint32_t n)
{
    while (n--)
    {
        delay_1ms();
    }
}

/* ****************************************************************************** */
/* STM32L0xx Peripheral Interrupt Handlers */
/* Add here the Interrupt Handlers for the used peripherals. */
/* For the available peripheral interrupt handler names, */
/* please refer to the startup file (startup_stm32l0xx.s). */
/* ****************************************************************************** */

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/****************************************************************END OF FILE****/
```

Listing C.6: *stm32l0xx$_i$t.c*

#*

```
/**
  ******************************************************************************
  * @file    Inc/stm32l0xx_it.h
  * @date    05/04/2015 10:49:21
  * @author  Alexis Duque
  * @version V0.0.1
  * @brief   This file provides code for the MSP Initialization
  *                      and de-Initialization codes.
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32L0xx_IT_H
#define __STM32L0xx_IT_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void USART2_IRQHandler(void);
void SysTick_Handler(void);

void SysTick_Init(void);
void TimeTick_Decrement(void);
void delay_nus(uint32_t n);
void delay_1ms(void);
void delay_nms(uint32_t n);


#ifdef __cplusplus
}
#endif

#endif /* __STM32L0xx_IT_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
    FILE****/
```

Listing C.7: *stm32l0xx$_i$t.h*

Appendix D

Receiver software sources

#*

```c
/**
  ******************************************************************************
  * @file    Src/main.c
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-March-2015
  * @brief   Main ADC conversion with buffering
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "i2c.h"
#include "gpio.h"
#include "fifo.h"
#include "uart.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
extern UART_HandleTypeDef huart2;
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);

int main(void)
{

    /* MCU Configuration----------------------------------------------------------*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    FIFO_init(&AdcFIFO, AdcBuffer, BUFFER_SIZE);

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_ADC_Init();
    MX_I2C1_Init();
    #ifdef UART
    MX_UART2_UART_Init();
    #endif
    ARA_I2C_Listen();

    /* Infinite loop */
    while(1)
    {
        printf("hello");
    }

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_PeriphCLKInitTypeDef PeriphClkInit;

    __PWR_CLK_ENABLE();

    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = 16;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLLMUL_4;
    RCC_OscInitStruct.PLL.PLLDIV = RCC_PLLDIV_2;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1);

    PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2|
   RCC_PERIPHCLK_I2C1;
    PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
    PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
    HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);

    __SYSCFG_CLK_ENABLE();
}

/**
```

```
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line
    number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
    printf("Wrong parameters value: file %s on line %d\r\n", file, line);
}

#endif

/**
  * @}
  */

/**
  * @}
*/

/*****************************END OF FILE
    ***************************************/
```

Listing D.1: main.c

#*

```
/**
```

```
  ******************************************************************************
 * File Name          : Src/i2c.c
 * Date               : 05/04/2015 10:49:20
 * Description        : This file provides code for the configuration
 *                      of the I2C instances.
  ******************************************************************************
 */

/* Includes ------------------------------------------------------------------*/
#include "i2c.h"
#include "adc.h"
#include "main.h"

/* Private macro -------------------------------------------------------------*/
#define I2C_ADDRESS             0x50
#define MASTER_REQ_READ         0x01
#define MASTER_REQ_WRITE        0x02
#define MASTER_REQ_STOP         0x03
#define I2C_TIMING_100KHZ       0x00A03D53
#define I2C_TIMING_400KHZ       0x00500615
#define I2C_TIMING_32_400KHZ    0x00B0122A
#define I2C_TIMING_32_SFAST     0x0040060D
#define SLAVE_TX_ENABLED        0x01
#define SLAVE_TX_DISABLED       0x00

/* Private variables ---------------------------------------------------------*/
uint8_t slaveTxEnabled = SLAVE_TX_DISABLED;
/* I2C handler declaration */
I2C_HandleTypeDef I2CxHandle;

/* Buffer used for transmission */
uint8_t aTxBuffer[TXBUFFERSIZE];
uint16_t aTxSize = TXBUFFERSIZE;
/* Buffer used for reception */
```

```
uint8_t aRxBuffer[RXBUFFERSIZE];
uint16_t aRxSize = RXBUFFERSIZE;
uint16_t fifo_size = 0;
uint8_t bTransferRequest = 0;
HAL_StatusTypeDef read = 0;

/* Private function prototypes
    -----------------------------------------------*/
static void Flush_Buffer(uint8_t* pBuffer, uint16_t BufferLength);

/* Private functions
    ---------------------------------------------------------*/

/* I2C1 init function */
void MX_I2C1_Init(void)
{

    I2Cx_CLK_ENABLE();

    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin       = I2Cx_SCL_PIN;
    GPIO_InitStruct.Mode      = GPIO_MODE_AF_OD;
    GPIO_InitStruct.Pull      = GPIO_PULLUP;
    GPIO_InitStruct.Speed     = GPIO_SPEED_FAST;
    GPIO_InitStruct.Alternate = I2Cx_SCL_AF;

    HAL_GPIO_Init(I2Cx_SCL_GPIO_PORT, &GPIO_InitStruct);

    /* I2C RX GPIO pin configuration  */
    GPIO_InitStruct.Pin = I2Cx_SDA_PIN;
    GPIO_InitStruct.Alternate = I2Cx_SDA_AF;

    HAL_GPIO_Init(I2Cx_SDA_GPIO_PORT, &GPIO_InitStruct);

    /* NVIC for I2C1 */
    HAL_NVIC_SetPriority(I2Cx_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(I2Cx_IRQn);

    I2CxHandle.Instance = I2C1;
    I2CxHandle.Init.Timing =  0x00500615;
    I2CxHandle.Init.OwnAddress1 = I2C_ADDRESS;
    I2CxHandle.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    I2CxHandle.Init.DualAddressMode = I2C_DUALADDRESS_DISABLED;
    I2CxHandle.Init.OwnAddress2 = 0;
    I2CxHandle.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
    I2CxHandle.Init.GeneralCallMode = I2C_GENERALCALL_DISABLED;
    I2CxHandle.Init.NoStretchMode = I2C_NOSTRETCH_DISABLED;
    if(HAL_I2C_Init(&I2CxHandle) != HAL_OK)
    {
        Error_Handler();
    }

    /**Configure Analogue filter
    */
    HAL_I2CEx_AnalogFilter_Config(&I2CxHandle, I2C_ANALOGFILTER_ENABLED);
    Flush_Buffer((uint8_t*)aTxBuffer, aTxSize);
}

void HAL_I2C_MspInit(I2C_HandleTypeDef *I2CxHandle)
{

}

/**
  * @brief I2C MSP De-Initialization
  *        This function frees the hardware resources used in this
     example:
  *          - Disable the Peripheral's clock
  *          - Revert GPIO, DMA and NVIC configuration to their default
     state
  * @param hi2c: I2C handle pointer
  * @retval None
  */
void HAL_I2C_MspDeInit(I2C_HandleTypeDef *I2CxHandle)
{
    I2Cx_FORCE_RESET();
    I2Cx_RELEASE_RESET();

    HAL_GPIO_DeInit(I2Cx_SCL_GPIO_PORT, I2Cx_SCL_PIN);
    HAL_GPIO_DeInit(I2Cx_SDA_GPIO_PORT, I2Cx_SDA_PIN);

    /* Peripheral interrupt Deinit*/
    HAL_NVIC_DisableIRQ(I2C1_IRQn);
}

void ARA_I2C_Listen(void)
{
    HAL_I2C_Slave_Transmit_IT(&I2CxHandle, (uint8_t*)aTxBuffer, aTxSize);
}

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef *I2CxHandle)
{
    Flush_Buffer((uint8_t*)aTxBuffer, aTxSize);
    RES_t res = FIFO_read(&AdcFIFO, &aTxBuffer, aTxSize);
}

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef *I2CxHandle)
{
    if(aRxBuffer[0] == MASTER_REQ_READ)
    {
        slaveTxEnabled = SLAVE_TX_ENABLED;
    }
    else if(aRxBuffer[0] == MASTER_REQ_STOP)
    {
        slaveTxEnabled = SLAVE_TX_DISABLED;
    }
    Flush_Buffer((uint8_t*)aRxBuffer, aRxSize);
}

/**
  * @brief  Flushes the buffer
  * @param  pBuffer: buffers to be flushed.
  * @param  BufferLength: buffer's length
  * @retval None
  */
static void Flush_Buffer(uint8_t* pBuffer, uint16_t BufferLength)
{
    while(BufferLength--)
    {
        *pBuffer = 1;
        pBuffer++;
    }
}
```

```

/**
  * @brief  I2C error callbacks
  * @param  I2cHandle: I2C handle
  * @note   This example shows a simple way to report transfer error, and
     you can
  *         add your own implementation.
  * @retval None
  */
void HAL_I2C_ErrorCallback(I2C_HandleTypeDef *I2CxHandle)
{
    /* Turn On LED2 */
    Error_Handler();
}

/**
  * @}
  */

/**
  * @}
  */

/*****************************END OF FILE
    *************************************/
```

Listing D.2: i2c.c

#*

```
/**
  ******************************************************************************
  * File Name          : Src/fifo.c
  * Date               : 05/04/2015 10:49:20
  * Description        : This file provides code for the configuration
  *                      of the FIFO ABD Buffer.
  ******************************************************************************
  */

#include <stdint.h>
#include <stddef.h>
#include <string.h>
```

```c
#include "fifo.h"


void FIFO_init(FIFO_t *fifo, void *bufptr, size_t bufsize)
{
    fifo->bufptr = bufptr;
    fifo->bufsize = bufsize;
    fifo->rdidx = 0;
    fifo->wridx = 0;
#if(FIFO_LOG_MAX_USAGE == 1)
    fifo->max = 0;
#endif
}

RES_t FIFO_write(FIFO_t *fifo, void *src, size_t size)
{
    size_t wrcount;

    if(size > FIFO_wrcount(fifo))
    {
        return(RES_FULL);
    }

    if((wrcount = fifo->bufsize - fifo->wridx) <= size)
    {
        // write operation will wrap around in fifo
        // write first half of fifo
        memcpy(fifo->bufptr+fifo->wridx,src,wrcount);

        //wrap around and continue
        fifo->wridx = 0;
        size -= wrcount;
        src = (uint8_t*)src + wrcount;
    }

    if(size > 0)
    {
        memcpy(fifo->bufptr+fifo->wridx,src,size);
        fifo->wridx += size;
    }

```

```
#if(FIFO_LOG_MAX_USAGE == 1)
    wrcount = FIFO_rdcount(fifo);
    if(wrcount > fifo->max)
    {
        fifo->max = wrcount;
    }
#endif

    return(RES_OK);
}

void FIFO_write_trample(FIFO_t *fifo, void *src, size_t size)
{

    if(size >= fifo->bufsize-1)
    {
        // if writing more than can ever fit in the buffer,
        // only write the latter portion of src buf.

        fifo->wridx = fifo->bufsize-1;
        fifo->rdidx = 0;
        memcpy(fifo->bufptr, src + (size-(fifo->bufsize-1)), fifo->
    bufsize-1);
    }
    else
    {
        size_t wrcount;
        int overflow = 0;
        if(size > FIFO_wrcount(fifo))
        {
            overflow = 1;
        }

        wrcount = fifo->bufsize - fifo->wridx;
        if(wrcount <= size)
        {
            // write operation will wrap around in fifo
            // write first half of fifo
            memcpy(fifo->bufptr+fifo->wridx, src, wrcount);

            //wrap around and continue
            fifo->wridx = 0;
            size -= wrcount;
            src = (uint8_t*)src + wrcount;
        }

        if(size > 0)
        {
            memcpy(fifo->bufptr+fifo->wridx, src, size);
            fifo->wridx += size;
        }

        // if overflow happens, move read pointer.
        if(overflow)
        {
            if(fifo->wridx == fifo->bufsize-1)
            {
                fifo->rdidx = 0;
            }
            else
            {
                fifo->rdidx = fifo->wridx + 1;
            }
        }
    }
}

RES_t FIFO_read(FIFO_t *fifo, void *dst, size_t size)
{
    size_t rdcount;

    if(size > FIFO_rdcount(fifo))
    {
        return(RES_PARAMERR);
    }

    rdcount = fifo->bufsize - fifo->rdidx;
    if(rdcount <= size)
    {
        // read operation will wrap around in fifo
        // read first half of fifo
        if(dst != NULL)
        {
            memcpy(dst, fifo->bufptr + fifo->rdidx, rdcount);
        }
        //wrap around and continue
        fifo->rdidx = 0;
        size -= rdcount;
        dst = (uint8_t*)dst + rdcount;
    }

    if(size > 0)
    {
        if(dst != NULL)
        {
            memcpy(dst, fifo->bufptr + fifo->rdidx, size);
        }
        fifo->rdidx += size;
    }

    return(RES_OK);
}

size_t FIFO_read_max(FIFO_t *fifo, void *dst, size_t max_size)
{
    size_t rdcount, nbytes;

    rdcount = FIFO_rdcount(fifo);
    if(max_size > rdcount)
    {
        max_size = rdcount;
    }
    nbytes = max_size;

    rdcount = fifo->bufsize - fifo->rdidx;
    if(rdcount <= nbytes)
    {
        // read operation will wrap around in fifo
        // read first half of fifo
        if(dst != NULL)
        {
            memcpy(dst, fifo->bufptr + fifo->rdidx, rdcount);
        }
        //wrap around and continue
        fifo->rdidx = 0;
        nbytes -= rdcount;
        dst = (uint8_t*)dst + rdcount;
    }

    if(nbytes > 0)
    {
        if(dst != NULL)
        {
            memcpy(dst, fifo->bufptr + fifo->rdidx, nbytes);
        }
        fifo->rdidx += nbytes;
    }

    return(max_size);
}

RES_t FIFO_peek(FIFO_t *fifo, void *dst, size_t size)
{
    size_t rdcount;
    size_t rdidx;

    if(size > FIFO_rdcount(fifo))
    {
        return(RES_PARAMERR);
    }

    rdidx = fifo->rdidx;

    if((rdcount = fifo->bufsize - rdidx) <= size)
    {
        // read operation will wrap around in fifo
        // read first half of fifo
        memcpy(dst,fifo->bufptr+rdidx,rdcount);
        //wrap around and continue
        rdidx = 0;
        size -= rdcount;
        dst = (uint8_t*)dst + rdcount;
    }

    if(size > 0)
    {
        memcpy(dst,fifo->bufptr+rdidx,size);
        rdidx += size;
    }

    return(RES_OK);
}

void FIFO_clear(FIFO_t *fifo)
{
    fifo->rdidx = 0;
    fifo->wridx = 0;
}

size_t FIFO_rdcount(FIFO_t *fifo)
{
    size_t wridx,rdidx;
    wridx = fifo->wridx;
    rdidx = fifo->rdidx;

    if(wridx >= rdidx)
    {
        return(wridx-rdidx);
    }
    else
    {
        return((fifo->bufsize-rdidx)+wridx);
    }
}


size_t FIFO_wrcount(FIFO_t *fifo)
{
    size_t wridx,rdidx;
    wridx = fifo->wridx;
    rdidx = fifo->rdidx;

    if(rdidx >= wridx+1)
    {
        return(rdidx-wridx-1);
    }
    else
    {
        return((fifo->bufsize-wridx)+rdidx-1);
    }
}

///\}

/****************************END OF FILE*********************************
    */
```

Listing D.3: fifo.c

#*

```
/**
    ****************************************************************************

  * @file    Src/dma.c
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-March-2015
  * @brief   Main ADC conversion with buffering
    ****************************************************************************

  */

/* Includes
    ------------------------------------------------------------------*/
#include "dma.h"

/* Private typedef
    -----------------------------------------------------------*/
/* Private define
    ------------------------------------------------------------*/
/* Private macro
    -------------------------------------------------------------*/
/* Private variables
    ---------------------------------------------------------*/
```

```
/* Private function prototypes
    -----------------------------------------------*/

/**
  * Enable DMA controller clock
  */
void MX_DMA_Init(void)
{
    //Nothing do to there ...
}

/**
  * @}
  */

/**
  * @}
*/

/*****************************END OF FILE
    *************************************/
```

Listing D.4: dma.c

#*

```
/**

    ******************************************************************************

  * File Name          : Src/adc.c
  * Date               : 05/04/2015 10:49:20
  * Description        : This file provides code for the configuration
  *                        of the ADC instances.

    ******************************************************************************

  */

/* Includes
    ------------------------------------------------------------------*/
#include <stdlib.h>
```

```c
#include <string.h>
#include "adc.h"
#include "math.h"
#include "4b6b.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
#define THRESHOLD 0
#define DETECTION_THRESHOLD 300
#define BIT_LENGTH 3
#define DMA_BUFFER_SIZE 512
/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef AdcHandle;
DMA_HandleTypeDef DmaHandle;
UART_HandleTypeDef huart2;

FIFO_t AdcFIFO;
uint8_t AdcBuffer[BUFFER_SIZE];

uint32_t resultDMA[DMA_BUFFER_SIZE];
uint32_t tmp_resultDMA[DMA_BUFFER_SIZE];
uint8_t sampledDMA[DMA_BUFFER_SIZE/2];
uint32_t adcResultDMA;

uint8_t bitBuffer = 0x00;
int counter = 0;
long offset = 0;
unsigned char bit = 0;
float average, sum, sumTemp, variance, minVal, maxVal, noise;

uint32_t ADCVoltageValue = 0;
uint8_t ADCVoltageValue8 = 0;

char * val;
char * valBit;
char * valFiltered;
```

```

uint8_t buff = 0x00;
uint8_t * buff_ptr = &buff;

/* Private function prototypes
    -----------------------------------------------*/
double meanArray(uint32_t * adcArray, long arraySize);
long sumArray(uint32_t * adcArray, long arraySize);

/* ADC init function */
void MX_ADC_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    /* Peripheral clock enable */
    __ADC1_CLK_ENABLE();
    /* DMA controller clock enable */
    __DMA1_CLK_ENABLE();

    /**ADC GPIO Configuration
    PA0-WKUP     ------> ADC_IN0
    */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* Peripheral DMA init*/

    DmaHandle.Instance = DMA1_Channel1;
    DmaHandle.Init.Request = DMA_REQUEST_0;
    DmaHandle.Init.Direction = DMA_PERIPH_TO_MEMORY;
    DmaHandle.Init.PeriphInc = DMA_PINC_DISABLE;
    DmaHandle.Init.MemInc = DMA_MINC_ENABLE;
    DmaHandle.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
    DmaHandle.Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
    DmaHandle.Init.Mode = DMA_CIRCULAR;
    DmaHandle.Init.Priority = DMA_PRIORITY_MEDIUM;
    HAL_DMA_DeInit(&DmaHandle);
    HAL_DMA_Init(&DmaHandle);

    __HAL_LINKDMA(&AdcHandle, DMA_Handle, DmaHandle);

    /* DMA interrupt init */
    HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

    ADC_ChannelConfTypeDef sConfig;
    ADC_AnalogWDGConfTypeDef AnalogWDGConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data
   Alignment and number of conversion)
    */
    AdcHandle.Instance = ADC1;
    AdcHandle.Init.OversamplingMode = DISABLE;
    AdcHandle.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV1;
    AdcHandle.Init.Resolution = ADC_RESOLUTION8b;
    AdcHandle.Init.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
    AdcHandle.Init.ScanDirection = ADC_SCAN_DIRECTION_UPWARD;
    AdcHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    AdcHandle.Init.ContinuousConvMode = ENABLE;
    AdcHandle.Init.DiscontinuousConvMode = DISABLE;
    AdcHandle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIG_EDGE_NONE;
    AdcHandle.Init.DMAContinuousRequests = ENABLE;
    AdcHandle.Init.EOCSelection = EOC_SINGLE_CONV;
    AdcHandle.Init.Overrun = OVR_DATA_PRESERVED;
    AdcHandle.Init.LowPowerAutoWait = DISABLE;
    AdcHandle.Init.LowPowerFrequencyMode = DISABLE;
    AdcHandle.Init.LowPowerAutoOff = DISABLE;

    HAL_ADC_Init(&AdcHandle);
    HAL_ADCEx_Calibration_Start(&AdcHandle, ADC_SINGLE_ENDED);

    sConfig.Channel = ADC_CHANNEL_0;
    HAL_ADC_ConfigChannel(&AdcHandle, &sConfig);

    /**Configure the analog watchdog
    */
    AnalogWDGConfig.WatchdogMode = ADC_ANALOGWATCHDOG_SINGLE_REG;
    AnalogWDGConfig.Channel = ADC_CHANNEL_0;
    AnalogWDGConfig.ITMode = ENABLE;
    AnalogWDGConfig.HighThreshold = 0;
    AnalogWDGConfig.LowThreshold = 0;
    HAL_ADC_AnalogWDGConfig(&AdcHandle, &AnalogWDGConfig);

    HAL_ADC_Start_DMA(&AdcHandle, &resultDMA, DMA_BUFFER_SIZE);

}

void HAL_ADC_MspInit(ADC_HandleTypeDef* AdcHandle)
{
    // Nothing to do there;
}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* AdcHandle)
{

    if(AdcHandle->Instance==ADC1)
    {
        /* Peripheral clock disable */
        __ADC1_CLK_DISABLE();

        /**ADC GPIO Configuration
        PA0-WKUP     ------> ADC_IN0
        */
        HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0);

        /* Peripheral DMA DeInit*/
        HAL_DMA_DeInit(AdcHandle->DMA_Handle);

    }
}

uint8_t ARA_ADC_Threashold(uint32_t voltage)
{
    if (voltage > 400) return 1;
    return (voltage > (minVal + DETECTION_THRESHOLD)) ? 1 : 0;
}

void compute_stat(uint32_t * buffer, int bufferSize)
{
    int i = 0;
    sum = 0;
    average =0;
```

```c
    variance = 0;
    minVal = 0;
    maxVal = 0;
    sumTemp = 0;
    for (i = 0; i < bufferSize; i++)
    {
        sum += buffer[i];
        minVal = buffer[i] < minVal ? buffer[i] : minVal;
        maxVal = buffer[i] > maxVal ? buffer[i] : maxVal;
    }
    average = sum / (float)bufferSize;
    /*  Compute  variance  and standard deviation  */
    for (i = 0; i < bufferSize; i++)
    {
        sumTemp += powf((float)(buffer[i] - average), (float)2);
    }
    variance = sumTemp / (float)bufferSize;

}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
{
    int i,y, preamb = 0;
    int succsiveOne = 0;
    int sample = 0;
    uint8_t filtered = 0;
    memcpy(&tmp_resultDMA, &resultDMA, DMA_BUFFER_SIZE * 4);

    compute_stat(tmp_resultDMA, DMA_BUFFER_SIZE);
    noise = variance < 1000 ? average : noise;
    for (i = 0; i < DMA_BUFFER_SIZE; i++)
    {
        sample++;
        ADCVoltageValue = tmp_resultDMA[i] / 16;

        int ADCVoltageValueUART =  tmp_resultDMA[i];
        val = itoa(ADCVoltageValueUART);
        #ifdef UART
        HAL_UART_Transmit(&huart2, (uint8_t*)val, strlen(val), 10);
        HAL_UART_Transmit(&huart2, (uint8_t*)&",", 2, 10);
        #endif
        uint8_t bit = 0;
        bit = ARA_ADC_Threashold(ADCVoltageValueUART);
        ADCVoltageValue8 = (uint8_t) ADCVoltageValue;

        /* USELESS
        uint8_t filtered = 0;
        filtered = ((ADCVoltageValueUART - average) > 0) ? (
    ADCVoltageValueUART - average) : 0;
        valFiltered = itoa(filtered);
        HAL_UART_Transmit(&huart2, (uint8_t*)valFiltered, strlen(
    valFiltered), 10);
        HAL_UART_Transmit(&huart2, (uint8_t*)&",", 2, 10);
        */

        valFiltered = bit == 0 ? itoa(1) : itoa(10);

        if (sample == (BIT_LENGTH))
        {
            sample = 0;
            if (bit == 0)
            {
                valFiltered = itoa(-1);
            }
            else
            {
                succsiveOne = succsiveOne++;
                valFiltered = 1;
            }
            sampledDMA[y] = bit;
            appendBit(bit, buff_ptr);
            if (*buff_ptr == CODE_4B6B_PRE2 && preamb == 0)
            {
                preamb = 1;
            }
            else if(*buff_ptr == CODE_4B6B_PRE2 && preamb == 1)
            {
                preamb = 0;
            }
            if (preamb == 1)
            {
                if (bit == 1)
                {
                    bitBuffer |= (1u << (7 - counter));
                }
                counter++;
                if (counter > 7)
                {
                    FIFO_write_trample(&AdcFIFO, &bitBuffer, 1);
                    counter = 0;
                    bitBuffer = 0x00;
                }

            }
            y = y++;
            #ifdef UART
            HAL_UART_Transmit(&huart2, (uint8_t*)valFiltered, strlen(valFiltered), 10);
            HAL_UART_Transmit(&huart2, (uint8_t*)&",", 2, 10);
            #endif
        }
        else
        {
            valFiltered = itoa(0);
            #ifdef UART
            HAL_UART_Transmit(&huart2, (uint8_t*)valFiltered, strlen(valFiltered), 10);
            HAL_UART_Transmit(&huart2, (uint8_t*)&",", 2, 10);
            #endif
        }

        valBit = itoa(bit);
        #ifdef UART
        HAL_UART_Transmit(&huart2, (uint8_t*)valBit, strlen(valBit), 10);
        HAL_UART_Transmit(&huart2, (uint8_t*)&"\r", 4, 10);
        #endif


        /* DISABLE FOR TEST
        bit = ARA_ADC_Threashold(ADCVoltageValue);
        if (bit == 1)
        {
            bitBuffer |= (1u << (7 - counter));
        }
        counter++;
        if (counter > 7)
        {
            FIFO_write_trample(&AdcFIFO, &bitBuffer, 1);
            counter = 0;
            bitBuffer = 0x00;
        }
        */
    }

//HAL_UART_Transmit(&huart2, (uint8_t*)&"\r", 4, 10);
}

/**
  * @}
  */

/**
  * @}
  */

/*****************************END OF FILE
    *************************************/
```

Listing D.5: adc.c

#*

```
/**
    ******************************************************************************

  * @file    Inc/main.h
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-March-2015
  * @brief   Main ADC conversion with buffering

    ******************************************************************************

  */
```

```

/* Define to prevent recursive inclusion
    -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

/* Includes
    ------------------------------------------------------------------*/
#include <stdint.h>
#include <stdio.h>
#include "stm32l0xx_hal.h"

/* Exported types
    ------------------------------------------------------------*/
/* Exported constants
    --------------------------------------------------------*/
/* Exported macro
    ------------------------------------------------------------*/
#define BUFFER_SIZE             2048
/* Exported functions
    ------------------------------------------------------- */
void Error_Handler(void);

#endif /* __MAIN_H */

/******************************END OF FILE
    ***************************************/
```

Listing D.6: main.h

#*

```
/**
    ******************************************************************************
 * File Name          : Inc/i2c.h
 * Date               : 05/04/2015 10:49:21
 * Description        : This file provides code for the configuration
 *                      of the I2C instances.
```

```
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __i2c_H
#define __i2c_H
#ifdef __cplusplus
    extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32l0xx_hal.h"
#include "gpio.h"
/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/

/* User can use this section to tailor I2Cx/I2Cx instance used and associated
   resources */

/* Definition for I2Cx clock resources */
#define I2Cx                              I2C1
#define I2Cx_CLK_ENABLE()                __I2C1_CLK_ENABLE()
#define I2Cx_SDA_GPIO_CLK_ENABLE()       __GPIOB_CLK_ENABLE()
#define I2Cx_SCL_GPIO_CLK_ENABLE()       __GPIOB_CLK_ENABLE()

#define I2Cx_FORCE_RESET()               __I2C1_FORCE_RESET()
#define I2Cx_RELEASE_RESET()             __I2C1_RELEASE_RESET()

/* Definition for I2Cx Pins */
#define I2Cx_SCL_PIN                    GPIO_PIN_8
#define I2Cx_SCL_GPIO_PORT              GPIOB
#define I2Cx_SCL_AF                     GPIO_AF4_I2C1
#define I2Cx_SDA_PIN                    GPIO_PIN_9
#define I2Cx_SDA_GPIO_PORT              GPIOB
#define I2Cx_SDA_AF                          GPIO_AF4_I2C1

/* Definition for I2Cx's NVIC */
#define I2Cx_IRQn                         I2C1_IRQn
#define I2Cx_IRQHandler                   I2C1_IRQHandler

/* Size of Trasmission buffer */
#define TXBUFFERSIZE                           50
/* Size of Reception buffer */
#define RXBUFFERSIZE                           1

/* Exported macro
    ------------------------------------------------------------*/
#define COUNTOF(__BUFFER__)   (sizeof(__BUFFER__) / sizeof(*(__BUFFER__))
    )
/* Exported functions
    ------------------------------------------------------- */
extern I2C_HandleTypeDef I2CxHandle;

void MX_I2C1_Init(void);

void ARA_I2C_Listen(void);

#ifdef __cplusplus
}
#endif
#endif /*__ i2c_H */

/**
  * @}
  */

/**
  * @}
  */

/****************************END OF FILE
    ***************************************/
```

Listing D.7: i2c.h

#*

```c
/**
 ******************************************************************************
  * File Name          : fifo.
  * Date               : 05/04/2015 10:49:20
  * Description        : This file provides code for the configuration
  *                        of the FIFO ABC Buffer.
 ******************************************************************************
  */

#ifndef FIFO_H
#define FIFO_H

#include <stdint.h>
#include <stddef.h>

typedef struct
{
    uint8_t *bufptr;    // pointer to the buffer array
    size_t bufsize;    // size of buffer
    size_t rdidx;    // points to next address to be read
    size_t wridx;    // points to next address to be written
#if(FIFO_LOG_MAX_USAGE == 1)
    size_t max;
#endif
} FIFO_t;

// RESULT Enum
typedef enum
{
    RES_OK,          ///< Function executed successfully
    RES_FAIL,        ///< Function failed to execute properly
    RES_INVALID,     ///< Parameters returned are not valid
    RES_NOTFOUND,    ///< Item requested was not found
    RES_FULL,        ///< Buffer/Memory/other is full
    RES_UNDERRUN,    ///< Buffer underrun has occurred
    RES_OVERRUN,     ///< Buffer overrun has occured
    RES_PARAMERR,    ///< Invalid input parameter
    RES_END,         ///< Reached the end of a buffer
    RES_BUSY,        ///< Device is busy
    RES_CANCEL,      ///< Operation has been cancelled
    RES_UNKNOWN      ///< Unknown Error
} RES_t;


void FIFO_init(FIFO_t *fifo, void *bufptr, size_t bufsize);

RES_t FIFO_write(FIFO_t *fifo, void *src, size_t size);

void FIFO_write_trample(FIFO_t *fifo, void *src, size_t size);

RES_t FIFO_read(FIFO_t *fifo, void *dst, size_t size);

size_t FIFO_read_max(FIFO_t *fifo, void *dst, size_t max_size);

RES_t FIFO_peek(FIFO_t *fifo, void *dst, size_t size);

void FIFO_clear(FIFO_t *fifo);

size_t FIFO_rdcount(FIFO_t *fifo);

size_t FIFO_wrcount(FIFO_t *fifo);

#endif

/*****************************END OF FILE
    ***************************************/
```

Listing D.8: fifo.h

#*

```
/**
    ******************************************************************************
  * @file    Inc/dma.h
  * @author  Alexis Duque
  * @version V0.0.1
  * @date    30-March-2015
  * @brief   Main ADC conversion with buffering
    ******************************************************************************
  */

/* Define to prevent recursive inclusion
    -------------------------------------*/
#ifndef __DMA_H
#define __DMA_H

/* Includes
    ------------------------------------------------------------------*/
#include "stm32l0xx_hal.h"

void MX_DMA_Init(void);
/* Exported types
    ------------------------------------------------------------*/
/* Exported constants
    --------------------------------------------------------*/
/* Exported macro
    ------------------------------------------------------------*/
/* Exported functions
    ------------------------------------------------------- */

#endif /* __DMA_H */


/****************************END OF FILE
    ************************************/
```

Listing D.9: dma.h

#*

```
/**
    ******************************************************************************
  * File Name          : Inc/adc.h
  * Date               : 05/04/2015 10:49:20
  * Description        : This file provides code for the configuration
  *                       of the ADC instances.
  
    ******************************************************************************
  
  */

/* Define to prevent recursive inclusion
    -------------------------------------*/
#ifndef __adc_H
#define __adc_H
#ifdef __cplusplus
    extern "C" {
#endif

/* Includes
    ------------------------------------------------------------------*/
#include "stm32l0xx_hal.h"
#include "gpio.h"
#include "main.h"
#include "i2c.h"
#include "fifo.h"
#include "uart.h"
#include "math.h"

extern ADC_HandleTypeDef AdcHandle;
extern DMA_HandleTypeDef DmaHandle;

extern FIFO_t AdcFIFO;
extern uint8_t AdcBuffer[BUFFER_SIZE];

void MX_ADC_Init(void);
uint32_t ARA_ADC_GetValue(void);
double meanArray(uint32_t * adcArray, long arraySize);
uint32_t ARA_ADC_Start(void);

#ifdef __cplusplus
}
#endif
#endif /*__ adc_H */

/**
```

```
  * @}
  */

/**
  * @}
  */

/*****************************END OF FILE
    ************************************/
```

Listing D.10: adc.h

Appendix E

Java software sources

#*

```
/**
 * Created by Alexis DUQUE - alexisd61@gmail.com.
 * Date : 10/04/15.
 */

package edu.upc.entel.wng.vlcAraModule;

import android.content.Context;
import android.hardware.I2cManager;
import android.hardware.I2cTransaction;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

import java.io.IOException;
import java.util.Date;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class Sensor {
    // The 7-bit slave address
    private static final int address = (0x50 >> 1);
    private static final String bus = "/dev/i2c-4";

```

```java
    private static final int BUFFER_SIZE = 50;
    private static final String TAG = "AraVLC";
    private static final String BUF = "RxData@";

    private Context context;
    private I2cManager i2c;
    private Handler handler;
    private ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);

    private static I2cTransaction WriteReg(int reg, int b1, int b2, int b3) {
        return I2cTransaction.newWrite(0x01, reg, b1, b2, b3);
    }

    private static final I2cTransaction[] bufferRead = {
        I2cTransaction.newRead(BUFFER_SIZE),
    };

    public Sensor(Context context, Handler handler) {
        this.context = context;
        this.i2c = (I2cManager)context.getSystemService("i2c");
        this.handler = handler;
    }

    private I2cTransaction[] execute(I2cTransaction[] txns) {
        I2cTransaction[] results;
        try {
            results = null;
            for (I2cTransaction txn: txns)
                results = i2c.performTransactions(bus, address, txn);
        } catch (IOException e) {
            throw new RuntimeException("I2C error: " + e);
        }
        return results;
    }

    public void start() {
        executor.scheduleAtFixedRate(collector, 30, 30, TimeUnit.MILLISECONDS);
    }

    public void stop() {
        executor.shutdown();
        try {
            executor.awaitTermination(30, TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
            assert false;
        }
    }


    private final Runnable collector = new Runnable() {
        public void run() {
            I2cTransaction[] results;
            byte[] data;
            int[] val;
            results = execute(bufferRead);
            data = results[0].data;
            logBinary(data);
        }
    };

    private void logBinary(byte[] data) {
        String binary;
        Message m = new Message();
        Date rxTimestamp = new Date();
        StringBuilder sbBinary = new StringBuilder("");
        for (byte b1 : data) {
            String s1 = String.format("%8s", Integer.toBinaryString(b1 &
0xFF)).replace(' ', '0');
            sbBinary.append(s1);
        }
        binary = sbBinary.toString();
        Bundle b = new Bundle();
        b.putString("rxString", binary);
        m.setData(b);
        handler.sendMessage(m);
        Log.d(BUF + rxTimestamp.getTime() + ':', binary);
    }

    private int[] getBinary(byte[] bytesArray) {
        int bits[] = new int[BUFFER_SIZE];
        int i = 0;
        for (byte b : bytesArray) {
            String binary = Integer.toBinaryString(b & 255 | 256).
    substring(1);
            bits[i] = Integer.parseInt(binary);
            i++;
        }
        return bits;
    }

}
```

Listing E.1: Sensor.java

#*

```
package edu.upc.entel.wng.vlcAraModule;

import android.content.Context;
import android.os.Environment;
import android.util.Log;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

/**
 * Created by Alexis DUQUE - alexisd61@gmail.com.
 * Date : 25/05/15.
 */
public class VlcLogger {
    private Context context;

    public VlcLogger() {
    }

    /* Checks if external storage is available for read and write */
    public boolean isExternalStorageWritable() {
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            return true;
        }
        return false;
    }

    /* Checks if external storage is available to at least read */
    public boolean isExternalStorageReadable() {
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state) ||
                Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
            return true;
        }
        return false;
    }

    public void saveRxBits(String rxBuffer) {
        if (isExternalStorageReadable() && isExternalStorageWritable()) {
            saveRxBitsExternal(rxBuffer);
        } else {
            saveRxBitsInternal(rxBuffer);
        }
    }
    private void saveRxBitsExternal(String rxBuffer) {
        String root = Environment.getExternalStorageDirectory().toString
    ();
        File myDir = new File(root + "/VlcAraLogs/");
        myDir.mkdirs();
        SimpleDateFormat sdf = new SimpleDateFormat("MMddyy-HHmmss",
   Locale.FRANCE);
        String date = sdf.format(new Date());
        String filename = "record-"+ date +".txt";
        File file = new File (myDir, filename);
        if(file.exists())
        {
             writeToFile(file, rxBuffer);
        } else {
            try {
                file.createNewFile();
                writeToFile(file, rxBuffer);
            } catch (Exception e) {
                Log.e("VlcLogger", "Cant't create file" + file.getAbsolutePath());
            }
        }
    }

    private void writeToFile(File file, String string) {
        try {
            FileOutputStream fOut = new FileOutputStream(file, true);
            OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
            myOutWriter.append(string);
            myOutWriter.close();
            fOut.close();
        } catch(Exception e) {
            Log.e("VlcLogger", "Cant't write to file" + file.getAbsolutePath());
        }
    }

    private void saveRxBitsInternal(String rxBuffer) {
        String root = Environment.getExternalStorageDirectory().toString();
        File myDir = new File(context.getFilesDir() + "/VlcAraLogs/");
        myDir.mkdirs();
        SimpleDateFormat sdf = new SimpleDateFormat("MMddyy-HHmmss", Locale.FRANCE);
        String date = sdf.format(new Date());
        String filename = "record-"+ date +".txt";
        File file = new File (myDir, filename);
        if(file.exists())
        {
            writeToFile(file, rxBuffer);
        } else {
            try {
                file.createNewFile();
                writeToFile(file, rxBuffer);
            } catch (Exception e) {
                Log.e("VlcLogger", "Cant't create file" + file.getAbsolutePath());
                }
            }

        }


}
```

Listing E.2: VlcLogger.java

#*

```
/**
 * Created by Alexis DUQUE - alexisd61@gmail.com.
 * Date : 10/04/15.
 */

package edu.upc.entel.wng.vlcAraModule;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;

public class VlcAraActivity extends Activity {
    private Chart redChart;
    private Sensor sensor;
    private EditText rxText;
    private Button clearBtn;
    private Button saveBtn;
    private Switch enLog;
    private VlcLogger vlcLogger;

    private final Handler rxHandler = new Handler() {
        public void handleMessage(Message msg) {
            String rxBits = msg.getData().getString("rxString");
            if (rxText.getText().length() > 1000) clearEditText();
            if (enLog.isChecked()) vlcLogger.saveRxBits(rxBits);
            rxText.append(rxBits + System.getProperty("line.separator"));
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ara_plox);

        TextView irTextView = (TextView) findViewById(R.id.irTextView);
        irTextView.setTextColor(Color.WHITE);

        rxText = (EditText) findViewById(R.id.rxText);

        saveBtn = (Button) findViewById(R.id.saveBtn);
        saveBtn.setOnClickListener(new View.OnClickListener() {
                                               public void onClick(View v) {
                                                   saveEditText();
                                               }
                                           });

        clearBtn = (Button) findViewById(R.id.clearBtn);
        clearBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                clearEditText();
            }
        });

        vlcLogger = new VlcLogger();

        enLog = (Switch) findViewById(R.id.switch1);

        rxText.append(System.getProperty("line.separator") + "App is
started !!");
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensor = new Sensor(this, rxHandler);
        sensor.start();
    }

    @Override
    protected void onPause() {
        sensor.stop();
        sensor = null;
        rxHandler.removeMessages(0);
            super.onPause();
    }

    private void clearEditText() {
        rxText.setText("");
    }

    private void saveEditText() {
        vlcLogger.saveRxBits(rxText.getText().toString());
    }
}
```

Listing E.3: VlcAraActivity.java

#*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.upc.entel.wng.vlcAraModule"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="19" />
    <uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="VLC Ara Module"
        android:theme="@style/AppTheme" >
        <activity
```

```
            android:name="edu.upc.entel.wng.vlcAraModule.VlcAraActivity"
            android:label="VLC Ara Module" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
    />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Listing E.4: AndroidManifest.xml

Appendix F

JNI software sources

#*

```
#define LOG_TAG "I2cServiceJNI"
#include "utils/Log.h"

#include "jni.h"
#include "JNIHelp.h"

#include <fcntl.h>
#include <sys/ioctl.h>
#include <stdint.h>
#include <uapi/linux/i2c.h>
#include <uapi/linux/i2c-dev.h>

namespace android
{

// These I/O direction and status enums must match those in Java
// (see I2cTransaction.java)

enum {
    IO_DIRECTION_WRITE = 0,
    IO_DIRECTION_READ = 1,
};

enum {
    STATUS_NOT_STARTED = 1,
    STATUS_OK = 0,
    STATUS_ERR = -1,
};

static struct i2c_transaction_offsets_t
{
    jfieldID mIoDirection;
    jfieldID mStatus;
    jfieldID mData;
} gI2cTransactionOffsets;

static jint com_projectara_server_I2cService_native_perform_i2c_txns(
    JNIEnv *env, jobject thiz, jstring name, jint address, jobjectArray txns)
{
    jint ret = 0;

    const char *pathStr = env->GetStringUTFChars(name, NULL);
    int fd = open(pathStr, O_RDWR);
    if (fd < 0) {
        ALOGE("can't open %s", pathStr);
        env->ReleaseStringUTFChars(name, pathStr);
        return fd;
    }
    env->ReleaseStringUTFChars(name, pathStr);

    jsize nTxns = env->GetArrayLength(txns);
    struct i2c_msg messages[nTxns];
    for (jsize i = 0; i < nTxns; i++) {
        jobject txn = env->GetObjectArrayElement(txns, i);
        jint iod = env->GetIntField(txn, gI2cTransactionOffsets.mIoDirection);
        jobject dataObj =
            env->GetObjectField(txn, gI2cTransactionOffsets.mData);
        jbyteArray *dataAp = reinterpret_cast<jbyteArray*>(&dataObj);
        jsize dataLength = env->GetArrayLength(*dataAp);
        jbyte *datap = env->GetByteArrayElements(*dataAp, NULL);

        messages[i].addr = (__u16)address;
        messages[i].flags = (iod == IO_DIRECTION_READ) ? I2C_M_RD : 0;
        messages[i].len = (__u16)dataLength;
        messages[i].buf = (__u8*)datap;

        env->DeleteLocalRef(txn);
    }

    struct i2c_rdwr_ioctl_data packets;
    packets.msgs = messages;
    packets.nmsgs = nTxns;
    int io_rc = ioctl(fd, I2C_RDWR, &packets);
    if (io_rc < 0) {
        ALOGE("I/O error: %d", io_rc);
        ret = io_rc;
    }

    for (jsize i = 0; i < nTxns; i++) {
        jobject txn = env->GetObjectArrayElement(txns, i);
        jobject dataObj =
            env->GetObjectField(txn, gI2cTransactionOffsets.mData);
        jbyteArray *dataAp = reinterpret_cast<jbyteArray*>(&dataObj);

        env->ReleaseByteArrayElements(*dataAp, (jbyte*)messages[i].buf,
    0);
        env->SetIntField(txn, gI2cTransactionOffsets.mStatus,
                         ret < 0 ? STATUS_ERR : STATUS_OK);

        env->DeleteLocalRef(txn);
    }

    close(fd);
    return ret;
}

static JNINativeMethod method_table[] = {
    { "native_perform_i2c_txns",
      "(Ljava/lang/String;I[Lcom/projectara/hardware/bridge/
    I2cTransaction;)I",
      (void*)com_projectara_server_I2cService_native_perform_i2c_txns },
};

int register_com_projectara_server_I2cService(JNIEnv *env)
{
    const char *serviceName = "com/projectara/server/I2cService";
    const char *txnName = "com/projectara/hardware/bridge/I2cTransaction"
;

    jclass clazz = env->FindClass(serviceName);
    if (clazz == NULL) {
        ALOGE("Can't find %s", serviceName);
        return -1;
    }
    clazz = env->FindClass(txnName);
    LOG_FATAL_IF(clazz == NULL,
                 "Can't find com.projectara.hardware.bridge.
I2cTransaction");
    gI2cTransactionOffsets.mIoDirection = env->GetFieldID(clazz,
                                                         "ioDirection",
                                                         "I");
    gI2cTransactionOffsets.mStatus = env->GetFieldID(clazz, "status", "I"
);
    gI2cTransactionOffsets.mData = env->GetFieldID(clazz, "data", "[B");
    jniRegisterNativeMethods(env, serviceName, method_table,
                             NELEM(method_table));
    return 0;
}

}
```

Listing F.1: jni-i2c.c

#*

```
package com.projectara.hardware.bridge;

import android.content.Context;
import android.os.RemoteException;
import android.util.Log;

import java.io.IOException;

/**
 * @brief This class allows module support apps to control I2C devices
 *        on modules through bridge ASICs.
 *
 * Typical use (e.g. from an Activity):
```

```
 *
 * <pre>
 * // Get an I2cManager object, and find out the available I2C buses.
 * I2cManager i2c = ...;
 *
 * // Next, set up the I2C I/O you want to do.
 * //
 * // First, we'll write the three bytes 0x00, 0x01, 0x02.
 * I2cTransaction wTxn = I2cTransaction.newWrite(0x00, 0x01, 0x02);
 * // Then, we'll read 15 bytes back.
 * I2cTransaction rTxn = I2cTransaction.newRead(15);
 *
 * // Now, actually do the I/O. We pick the I2C bus arbitrarily,
 * // as the first one we got back.
 * I2cTransaction[] results;
 * try {
 *     results = i2c.performTransactions(buses[0], deviceAddr7bit,
 *                                        wTxn, rTxn);
 * } catch (IOException e) {
 *     // Something went wrong. Do your error handling here.
 * }
 *
 * // Otherwise, everything completed successfully.
 * // results[0] is the result of the write transaction.
 * // results[1] is the result of the read transaction.
 * for (byte b: results[1].data) {
 *     doSomethingWith(b);
 * }
 * </pre>
 */
public class I2cManager {
    private static final String TAG = "I2cManager";

    private final Context mContext; // currently ignored; likely
    necessary later
    private final II2cManager mService;

    /**
     * @hide
     */
    public I2cManager(Context context, II2cManager service) {
```

```java
        mContext = context;
        mService = service;
    }

    /**
     * Get the names of all available I2C buses, or null if none are
     * available.
     */
    public String[] getI2cBuses() {
        try {
            return mService.getI2cBuses();
        } catch (RemoteException e) {
            Log.e(TAG, "RemoteException in getI2cBuses", e);
            return null;
        }
    }

    /**
     * Perform a sequence of I2C transactions on a given bus.
     *
     * @param bus Name of I2C bus to perform transactions on
     * @param address Address of I2C slave, 7-bit, in bottom 7 bits
     * @param txns Transactions to perform
     *
     * @return Transaction results. The returned array is parallel to
     *         txns. That is, if txns[i] is a read transaction, the
     *         returned_array[i].data contains the read data.
     *
     * @see I2cTransaction
     */
    public I2cTransaction[] performTransactions(String bus, int address,
                                               I2cTransaction... txns)
        throws IOException {
        String errorMsg = "error while performing I2C transaction";
        checkAddrOk(address);
        int status;
        try {
            status = mService.performTransactions(bus, address, txns);
        } catch (RemoteException e) {
            throw new IOException(errorMsg, e);
        }
        if (status < 0) {
            throw new IOException(errorMsg);
        }
        return txns;
    }

    private static void checkAddrOk(int addr) {
        if (addr < 0 || (addr & ~0xFF) != 0) {
            throw new IllegalArgumentException("invalid address " + addr)
    ;
        }
    }
}
```

Listing F.2: I2cManager.java

#*

```java
package com.projectara.hardware.bridge;

import android.os.Parcel;
import android.os.Parcelable;

import java.util.Arrays;

/**
 * The I2cTransaction class represents a fixed amount of data to read
 * or write on an I2C bus. It is intended to be used with {@link
 * performTransactions I2cManager#performTransactions}.
 *
 * Typical use:
 *
 * <pre>
 *     // Makes a transaction which will write the tree bytes 0xAA, 0xBB,
 *     // and 0xCC, in that order.
 *     I2cTransaction writeTxn = I2cTransaction.newWrite(0xAA, 0xBB, 0xCC
   );
 *
 *     // Makes a transaction which will read two bytes
 *     I2cTransaction readTxn = I2cTransaction.newRead(2);
 * </pre>
 *
 * @see I2cManager
 */
public class I2cTransaction implements Parcelable {

    // These I/O directions and status definitions must keep up to date
    // with JNI (currently in com_projectara_server_I2cService.cpp).

    /** Transaction should read data. */
    public static final int IO_DIRECTION_READ = 1;
    /** Transaction has data to write. */
    public static final int IO_DIRECTION_WRITE = 0;

    /** Transaction hasn't been sent */
    public static final int STATUS_NOT_STARTED = 1;
    /** Transaction finished correctly (data was read/written) */
    public static final int STATUS_OK = 0;
    /** Error occurred while handling transaction */
    public static final int STATUS_ERR = -1;

    /**
     * Whether this transaction is a read or write.
     * @see IO_DIRECTION_READ
     * @see IO_DIRECTION_WRITE
     */
    public final int ioDirection;

    /**
     * Status of this I2C transaction.
     *
     * For platform internal use only.
     *
     * @hide
     */
    public int status;

    /**
     * Data read from or written to.
     *
     * Invalid until status==STATUS_OK.
     */
    public final byte[] data; // TODO: replace with ByteBuffer?

    // To prevent DoS attacks
    private static int MAX_TRANSACTION_SIZE = 512;

    private I2cTransaction(int ioDirection, byte[] data) {
        this(ioDirection, STATUS_NOT_STARTED, data);
    }

    private I2cTransaction(int ioDirection, int status, byte[] data) {
        this.ioDirection = ioDirection;
        this.status = status;
        this.data = data;
    }

    /**
     * Create a new read transaction.
     *
     * @param nBytes number of bytes to read
     */
    public static I2cTransaction newRead(int nBytes) {
        if (nBytes < 0) {
            throw new IllegalArgumentException("nBytes must be positive:
   " +
                                               nBytes);
        }
        if (nBytes > MAX_TRANSACTION_SIZE) {
            throw new IllegalArgumentException("nBytes " + nBytes +
                                               " is too large");
        }
        return new I2cTransaction(IO_DIRECTION_READ, new byte[nBytes]);
    }

    /**
     * Create a new write transaction.
     *
     * @param data Bytes to write
     */
    public static I2cTransaction newWrite(byte... data) {
        if (data == null) {
            throw new IllegalArgumentException("data must not be null");
        }
        if (data.length > MAX_TRANSACTION_SIZE) {
            throw new IllegalArgumentException("data.length " + data.
    length +
                                               " is too large");
        }
        if (data.length == 0) {
            throw new IllegalArgumentException("no data specified");
        }
        return new I2cTransaction(IO_DIRECTION_WRITE,
                                 Arrays.copyOf(data, data.length));
    }

    /**
     * Create a new write transaction.
     *
     * This is a convenience function to avoid having to cast literal
     * arguments to bytes. It works the same way as newWrite(byte...).
     *
     * @param data data to write; each value is converted to a byte first
    .
     */
    public static I2cTransaction newWrite(int... data) {
        if (data == null) {
            throw new IllegalArgumentException("data must not be null");
        }
        if (data.length > MAX_TRANSACTION_SIZE) {
            throw new IllegalArgumentException("data.length " + data.
    length +
                                               " is too large");
        }
        if (data.length == 0) {
            throw new IllegalArgumentException("no data specified");
        }
        byte[] bData = new byte[data.length];
        for (int i = 0; i < data.length; i++) {
            bData[i] = (byte)data[i];
        }
        return newWrite(bData);
    }

    // --------------- Parcelable API -------------------------------

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel out, int flags) {
        out.writeInt(ioDirection);
        out.writeInt(status);
        out.writeInt(data.length);
        out.writeByteArray(data);
    }

    public static final Parcelable.Creator<I2cTransaction> CREATOR =
        new Parcelable.Creator<I2cTransaction>() {

        @Override
        public I2cTransaction createFromParcel(Parcel source) {
            int ioDirection = source.readInt();
            int status = source.readInt();
            int dataLen = source.readInt();
            byte[] data = new byte[dataLen];
            source.readByteArray(data);
            return new I2cTransaction(ioDirection, status, data);
        }

        @Override
        public I2cTransaction[] newArray(int size) {
            return new I2cTransaction[size];
        }
    };
}
```

Listing F.3: I2cTransaction.java

#*

```
package com.projectara.server;

import java.io.File;
import java.io.FileFilter;
```

```java
import java.util.Arrays;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.regex.Pattern;

import com.projectara.hardware.bridge.II2cManager;
import com.projectara.hardware.bridge.I2cTransaction;
import android.util.Log;

/*
 * FIXME: add mechanism to request "my" module's i2c device (by vid/pid or something)
 */

/** @hide */
public class I2cService extends II2cManager.Stub {

    private static final String TAG = "I2cService";

    private static final String sBusNamePattern = "^i2c-\\d+$";
    private static final String sBusDirectory = "/dev";

    // Prevent DoS attacks
    private static final int MAX_TRANSACTIONS = 50;

    private static final String[] sBuses;
    private static final HashMap<String, Object> sBusLocks;

    // FIXME: add hotplug support
    static {
        ArrayList<String> busNames = new ArrayList<String>();
        sBusLocks = new HashMap<String, Object>();
        // Get an array of files in sBusDirectory whose names match
        // sBusNamePattern and which we can read and write.
        File[] buses = new File(sBusDirectory).listFiles(new FileFilter() {
                @Override
                public boolean accept(File f) {
                    return (f.canRead() && f.canWrite() &&
                            Pattern.matches(sBusNamePattern, f.getName()));
                }
            });
        // For each discovered bus, make a lock object.
        for (File bus: buses) {
            String busPath = bus.getAbsolutePath();
            sBusLocks.put(busPath, new Object());
            busNames.add(busPath);
            Log.d(TAG, "discovered bus: " + busPath);
        }
        // Cache the discovered bus names.
        sBuses = busNames.toArray(new String[busNames.size()]);
    }

    public I2cService() {
    }

    // FIXME: remove? Or at least add hotplug callbacks.
    public String[] getI2cBuses() {
        return Arrays.copyOf(sBuses, sBuses.length);
    }

    public int performTransactions(String bus, int address,
   I2cTransaction[] txns) {
        // XXX: add and use mechanism for checking caller permission
        if (!sBusLocks.containsKey(bus)) {
            throw new IllegalArgumentException("invalid bus: " + bus);
        }
        if (txns.length > MAX_TRANSACTIONS) {
            throw new IllegalArgumentException("too many transactions;
   maximum is " +
                                                        MAX_TRANSACTIONS);
        }
        if (txns.length == 0) {
            return 0;
        }
        synchronized (sBusLocks.get(bus)) {
            return native_perform_i2c_transactions(bus, address, txns);
        }
    }

    private static native int native_perform_i2c_txns(String bus, int
   address,
                                                             I2cTransaction[]
   txns);
   }
```

Listing F.4: I2cService.java

Appendix G

STM32L0 Nucleo Schmeatic

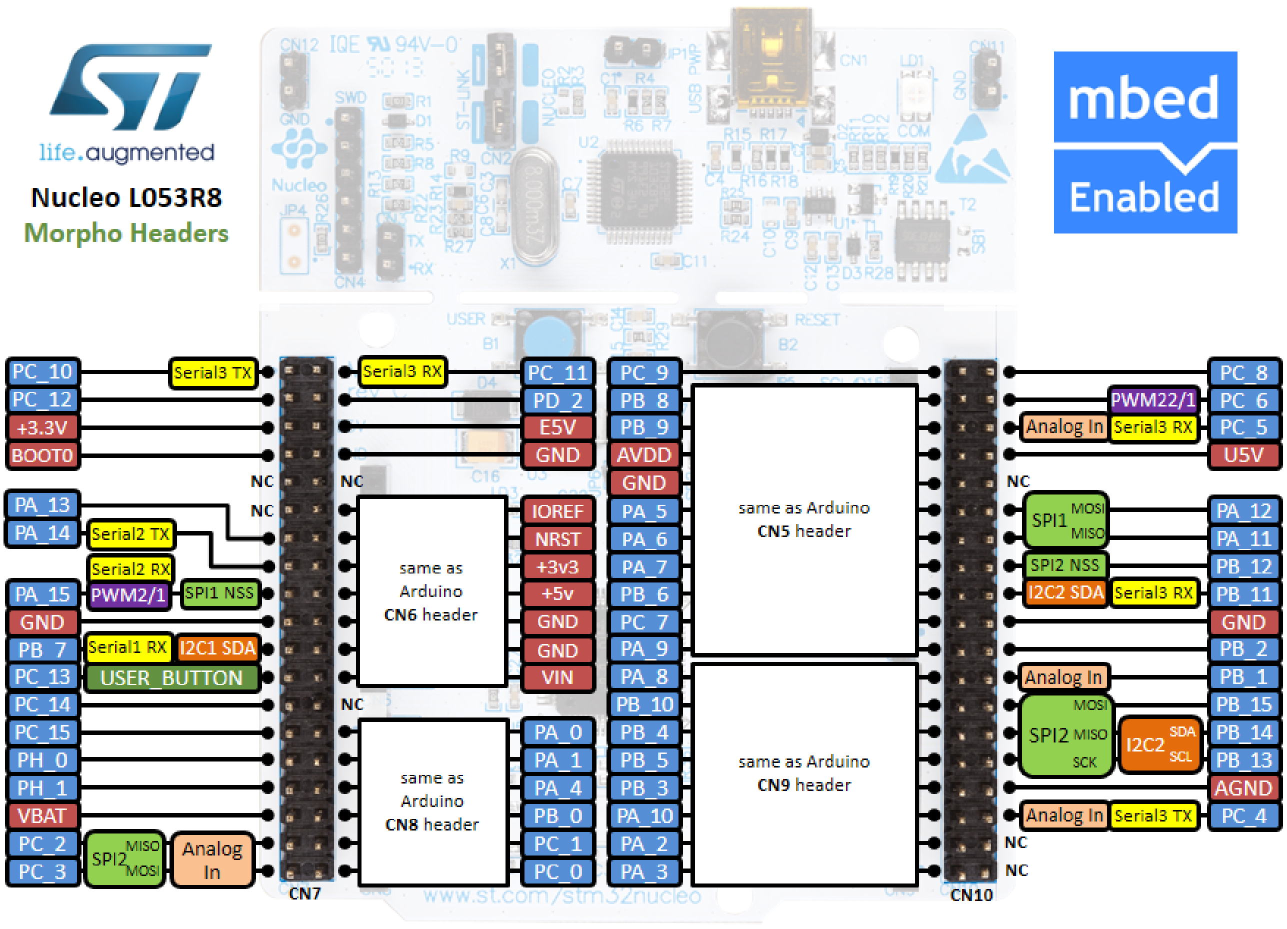


Figure G.1: STM32L0 Nucleo Schematic