A hardware and software platform for characterization and prototyping of a low-power energy-harvesting SoC

Doru-Stefan Irimescu

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.8.2018

Supervisor

Prof. Kari Halonen

Advisor

Tuomas Haapala

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Doru-Stefan Irimescu

Title A hardware and software platform for characterization and prototyping of a low-power energy-harvesting SoC

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems **Code of major** ELEC3025

Supervisor Prof. Kari Halonen

Advisor Tuomas Haapala

Date 31.8.2018 **Number of pages** 75 **Language** English

Abstract

Energy consumption is an important performance indicator for wireless devices. Developing ICs that address this issue for IoT applications is a complex task, which relies not only on design, but also on testing and characterization as a large part of the process.

This thesis develops a framework for testing, characterizing and prototyping of an ultra-low power IC developed at Aalto. The framework consists of both hardware and software components. The hardware involves a large four-layer PCB, various components that support the IC's functions and a smaller PCB which interfaces with a one-bit display, both implemented with Altium Designer, together with a UWB filter and an impedance matching network.

The software part consists of a flexible IC programming and configuration interface written in Python, two LabVIEW VIs for wireless data transmission and reception and a set of measurement automation libraries written in Python. The framework is successfully tested with the one-bit display driver and is used by the researchers for evaluating their IC blocks.

Keywords IC programming interface, Python, PCB design, IC characterization, low-power, energy harvester, measurement automation

Preface

I want to thank Prof. Kari Halonen for offering me this unique opportunity for working and growing within a team of experts. Without his wise coordination and managerial flexibility, this project would not have been possible. I would like to thank my advisor Tuomas Haapala for his extensive guidance and feedback, which have taught me new skills and good practices related to research and electronics. He has been committed to offering his expertise throughout my entire stay at the research group. I would also like to express my gratitude towards my research colleagues at the ECD group: Mika Pulkkinen, Jarno Salomaa and Mohammad Mehdi Moayer for their collaborative assistance.

Last, but not least, I thank my parents and girlfriend for their support.

Otaniemi, 31.8.2020

Doru-Stefan Irimescu

Contents

Symbols and abbreviations

Symbols

ε_R	Relative permittivity
γ	Propagation constant
Γ_L	Reflection coefficient measured at load
λ	Wavelength
$\phi(w)$	Phase response
Ω	Ohm
A_{max}	Maximum passband ripple
A_{min}	Minimum stopband attenuation
C_{out}	Output capacitance
C_p	Parasitic capacitance
C_x	Capacitor nr. x
D	Delta samples
$D(w)$	Group delay
E_{cap}	Energy stored in a capacitor
G	Gap between the signal line and coplanar ground layer
H	Dielectric height between the signal and ground plane
I	Current
I_{bias}	Bias current
I_{disp}	Display current
I_{ref}	Reference current
k	Index of sample s_k
L_p	Parasitic inductance
L_x	Inductor nr. x
N	Filter order
P	Power, peak sample set
$P(w)$	Phase delay
p	Percentage expressed in decimal points
p_k	Peak sample nr. k
Q	Electric charge
R_p	Parasitic resistance
R_{sense}	Sense resistor
S	Sample set
s_k	Value of sample nr. k
S_{ij}	Scattering parameter from port j to port i
T	Copper layer thickness
T_{symbol}	Time duration of a symbol in samples
$T_{timeslot}$	Time duration of a time slot in samples
t	Time
$tan\ \delta_e$	Dielectric loss tangent
V	Voltage
V_A	Voltage at node A
V_L	Voltage across inductor
V_{out}	Output voltage
V_s	Supply voltage
V_i^-	Reflected voltage at port i
V_i^+	Incident voltage at port i

w_{s1}	Stopband lower angular frequency limit
w_{s2}	Stopband upper angular frequency limit
w_{p1}	Passband lower angular frequency limit
w_{p2}	Passband upper angular frequency limit
W	Signal line width
z	Spatial distance from load towards generator
Z_{ic}	Integrated circuit input impedance
Z_{in}	Input impedance
Z_{ox}	Characteristic impedance nr. x

Abbreviations

ADC	Analog to Digital Converter
ADS	Advanced Design System
AMOLED	Active Matrix Organic Light Emitting Diode
API	Application Programming Interface
ASK	Amplitude Shift Keying
BPF	Bandpass Filter
CAD	Computer-Aided Design
CMOS	Complementary Metal-Oxide-Semiconductor
COM	Component Object Model
CPHA	Clock Phase
CPOL	Clock Polarity
CPWG	Coplanar Waveguide With Ground
CRC	Cyclid Redundancy Check
DAC	Digital to Analog Converter
DC	Direct Current
DNL	Differential Nonlinearity
DPPM	Differential Pulse-Position Modulation
DSO	Digital Storage Oscilloscope
EDA	Electronic Design Automation
EH	Energy Harvester
ESL	Equivalent Series Inductance
ESR	Equivalent Series Resistance
FIR	Finite Impulse Response
FPC	Flexible Printed Circuit
FPGA	Field Programmable Gate Array
GNU	GNU's Not Unix !
GPIB	General Purpose Interface Bus
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IF	Intermediate Frequency
INV	Inverter
IoT	Internet of Things
LAN	Local Area Network
LCD	Liquid Crystal Display
LCR	Inductance, Capacitance, Resistance
LNA	Low Noise Amplifier
LO	Local Oscillator
LOC	Lines Of Code
LS	Level Shifter
LSB	Least Significant Bit
LXI	Local area network eXtensions
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
MSB	Most Significant Bit
NB	Narrow Band
NI	National Instruments
NMOS	N-Channel Metal-Oxide-Semiconductor
NOC	Non Overlapping Clock
OOK	On-Off Keying

OOP	Object-Oriented Programming
OTA	Operational Transconductance Amplifier
PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative
PMOS	P-Channel Metal-Oxide-Semiconductor
PPM	Pulse-Position Modulation
QFN	Quad-Flat No-leads
REGU	Regulator
RF	Radio Frequency
RLC	Resistor, Inductor, Capacitor
SCL	Serial Clock Line
SCLK	Serial Clock
SCPI	Standard Commands for Programmable Instruments
SDA	Serial Data
SDR	Software Defined Radio
SM	Surface Mounted
SMA	SubMiniature version A
SMB	SubMiniature version B
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SS	Slave Select
SoC	System on Chip
TDD	Test-Driven Development
T&M	Testing and Measurement
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
USI	Universal Sensor Interface
USRP	Universal Software Radio Peripheral
UWB	Ultra-Wide Band
VI	Virtual Instrument
VISA	Virtual Instrument Software Architecture
VNA	Vector Network Analyzer

1 Introduction

In recent years, the energy efficiency requirements for electronic devices has dramatically increased, as manufacturers have to upgrade their design methodologies in order to conform with the latest standards for mitigating the accelerating impacts of global warming [1].

Lately, there has been a rise in demand for wireless devices that can seamlessly embed sensorial, computational and communicational capabilities into current technologies, or even into our everyday life in the form of the Internet of Things (IoT) [2]. Due to their small form factor, often hard-to-reach location and large scale deployment, IoT devices require stable, long-term power sources, typically in the form of a battery. Thus, full energy autonomy is becoming a highly sought-after characteristic for the next generation of IoT devices [3].

In order to address the problem of energy consumption in wireless devices, a low-power energy-harvesting System on Chip (SoC) was developed by Aalto University's Department of Electronics and Nanoengineering. A SoC is a complex integrated circuit (IC) that performs multiple different functions, typically integrating at least a processing unit, volatile and/or non-volatile memory, power regulation, timing sources and analog interfaces to name a few [4]. Developed as part of the department's prior research on ultra-low power circuits, Aalto's SoC design (Figure 1) consists of the following subsystems: an energy harvester (EH) and a voltage regulator (REGU) [5]-[6], a general-purpose sensor interface [7], a gesture sensor interface [8], a narrowband (NB) radio transceiver [9], an ultra-wideband (UWB) radio transmitter [10]-[11], a Serial Peripheral Interface (SPI) [12] and a one-bit display driver, all these features being integrated into an ultra-low power consumption solution.

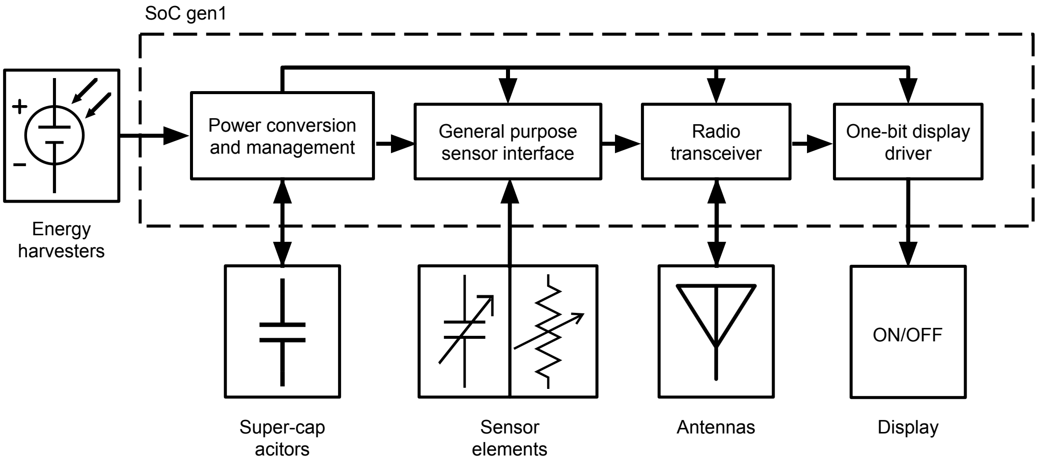


Figure 1: System on chip block diagram.

A representative IC design process (Figure 2) starts with the computer-aided design (CAD) of the integrated circuit based on some theoretical results, with the purpose of producing a functional IC that meets a set of specifications, forming a basis for submitting a new research paper. Next, the design is sent to a manufacturing company. In the case of the project in hand, the fabrication process is a 180-nm complementary metal-oxide-semiconductor (CMOS). Once the physical chips have been delivered, the testing and characterization process begins. The testing phase is particularly important in determining if there are any design or manufacturing flaws and if the chip functions as it was intended to, while the characterization process establishes and documents the operating parameters of the chip, which are then compared to the simulated results.

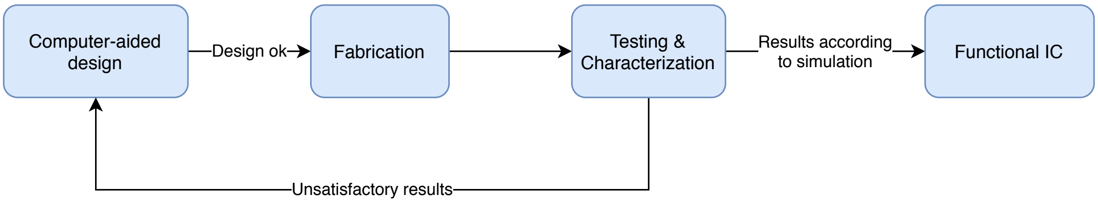


Figure 2: Typical IC design process.

The aim of this thesis is to construct a platform (software and hardware) that enables the research team to carry out the required testing and characterization, perform tests and evaluations on various IC blocks and build the foundation for a prototype that showcases the main features of the device. To accomplish this, the thesis first constructs a four-layer printed circuit board (PCB). Components that support the correct functioning of the IC are selected, with a focus on the universal sensor interface (USI), the UWB transmitter and NB receiver. Then, wireless communication between the IC and a computer is established. A programming interface that enables configuration and communication with the IC is developed. The measurement automation setup which puts together the characterization PCB, programming software and measurement devices is presented. Finally, the developed platform is put to the test by automating the measurements for the one-bit display driver.

Since this thesis is confined to designing the necessary hardware and software for accomplishing the aforementioned aim, the design of any of the IC blocks presented in the following chapters will remain beyond the scope of the thesis, as it constitutes the work performed by the group of researchers.

The remainder of this work is structured as follows: Chapter 2 reports the work for the USI, Chapter 3 presents the characterization PCB, Chapter 4 summarizes the work carried out for the NB receiver, Chapter 5 describes a UWB filter design, Chapter 6 presents the wireless communication framework, Chapter 7 introduces the Python interface for programming the device, Chapter 8 explains the measurement automation setup, Chapter 9 describes the evaluation of the one-bit display driver and, finally, Chapter 10 discusses the conclusions.

2 Universal sensor interface

This chapter describes the work that was carried out for enabling the measurements of the USI block. Section 2.1 introduces the USI and its common characteristics. Section 2.2 shows how sensors are emulated for prototyping the USI interface: the communication protocol is described, along with the tunable capacitor and digitally programmable potentiometer. Finally, Section 2.3 presents the Arduino libraries which were developed for interfacing the aforementioned devices.

2.1 Block description

The USI (Figure 3) is an ultra-low power, wide dynamic-range circuit which interfaces capacitive and resistive sensors. It is detailed in [13]. The USI can interface with capacitances ranging from 0.6 pF to 550 pF and resistances ranging from 3.7 kΩ to 5100 kΩ. The power consumption is situated in the range of 0.39 μW-3.56 μW, while the circuit is supplied with 1.2 V.

The capacitance and resistance values are converted to a voltage output. A switched-capacitor capacitance-to-voltage converter is used for interfacing capacitors. For measuring resistance, a voltage amplifier is used to measure a voltage drop produced by a reference current flowing through the sense resistor. An operational transconductance amplifier (OTA) is shared for realizing the switched-capacitor capacitance to voltage converter and the voltage amplifier. It is biased by a programmable bias generator circuit. A non-overlapping clock (NOC) phase generator is used for running the switched-capacitor circuits. Memory registers are implemented for configuring the front end's operation mode and range for the sensors. On the topical IC, a current reference is implemented for the resistive sensor reading.

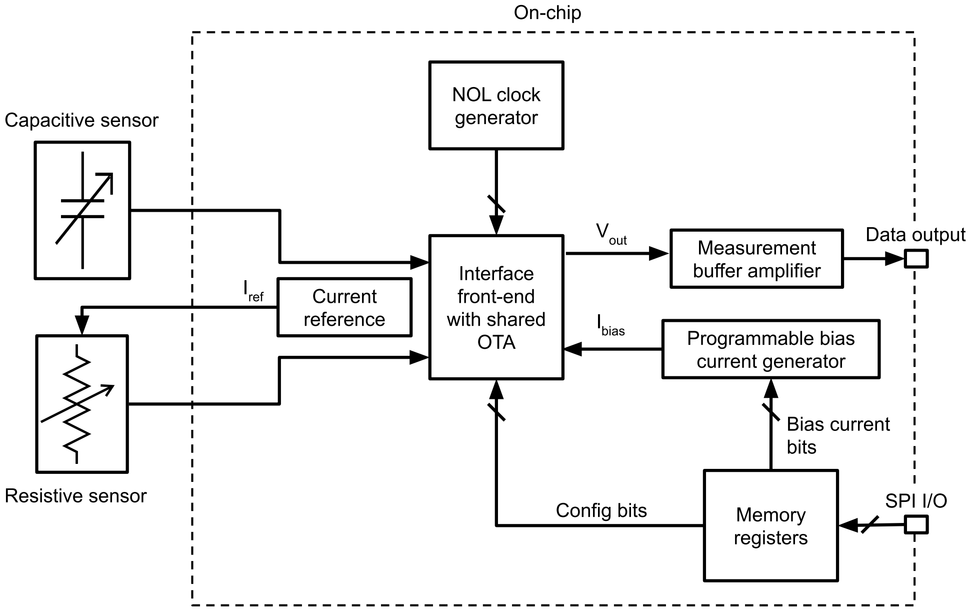


Figure 3: USI conceptual block diagram. Adapted from [13].

2.2 Sensor emulation

In order to demonstrate the capabilities of the IC, a means of emulating the resistive and capacitive sensor outputs under any conditions had to be found. It would be impractical to have a prototype that showcases the capabilities of the USI with, for example, a temperature and humidity sensor in a demonstrative scenario where the temperature and humidity cannot be controlled. Therefore, two components were chosen, a NCD2400M for emulating a capacitive sensor and a MAX5419 for emulating a resistive sensor. Finding a tunable capacitor within the required operating range was cumbersome and another candidate circuit, PE64102 from pSemi, had been considered and disregarded. The reason for disregarding the PE64102 was the low output capacitance range and high frequency operating range.

The selected ICs can be digitally programmed to output different values within their range of operation. Both components use the inter-integrated circuit (I2C) protocol. The components had to be chosen and tested prior to the making of the characterization PCB.

2.2.1 Inter-integrated circuit protocol

The I2C protocol was developed by Philips Semiconductors (now NXP Semiconductors) in 1982, with the purpose of offering a simple protocol that allows a small number of devices to communicate on a single card. The protocol is described in [14]. Over the years, it has become an industry standard, being used by virtually all major IC manufacturers. It is a synchronous, multi-master multi-slave serial communication protocol. Only two lines are required for communication, the serial data (SDA) and the serial clock (SCL) line. Every device connected to the bus must have an unique address. Depending on the mode, bitrates from 0.1 Mbit/s up to 3.4 Mbit/s can be achieved. The master has to provide the clock signal and initiate the communication. In order to send data over the bus, a series of 9 steps has to be performed as follows. First, the master has to wait until the bus is free: both the SDA and SCL have to be high. Second, the master signals that it has control of the bus and all the other devices listen for the incoming address. Third, the master provides the clock signal on the SCL line. During the fourth step, the master sends the unique address of the slave on the SDA line. In the fifth step, an additional bit is sent, signifying that the master is either sending or requesting data. Sixth, the master waits for an acknowledge bit. During the seventh step, the slave sends the acknowledge bit. In the eight step, the master sends or receives 8-bit data words, each followed by an acknowledge bit from the receiver (either slave or master, depending on which is sending the data). Finally, in the ninth step, the master sends a stop message and frees the bus.

In this project, the NCD2400M and MAX5419 chips are configured as slaves, receiving commands and replying to the Arduino Due microcontroller board which is acting as the bus master.

2.2.2 NCD2400M

NCD2400M from IXYS Integrated Circuits is a wide capacitance range, non-volatile digitally programmable capacitor. It was chosen for this project due to its suitable features, such as: 1.7 pF to 194 pF programmable output capacitance, in 512 steps of 376 fF each; 0 MHz to 150 MHz operating frequency; I2C programming interface; 2.5 V to 5.5V power supply; and, finally, non-volatile and volatile memory operation modes. It comes with a pre-programmed I2C bus address that cannot be changed. However, this does not constitute an issue, since only one such device is used in this project. The device accepts read and write commands and can operate in either volatile or non-volatile mode. The non-volatile operation makes it useful for scenarios when the prototype should retain its programmed value between power-up cycles, while in the volatile mode the programmed value is lost between power cycles. The output capacitance is measured between two pins, CP and CN. Depending on how these pins are configured, the device operates in two modes: shunt, when CN is connected to ground, or series otherwise. In the shunt mode, the parasitic capacitance of the CP pin to ground adds to the total equivalent output capacitance. For this project, the series mode of operation was used. According to the datasheet [15], the output capacitance C_{out} as a function of the programmed code (0 to 512) in the series mode can be calculated using:

$$C_{out} = code * \frac{193.84 - 1.7}{511} + 1.7\ pF. \tag{1}$$

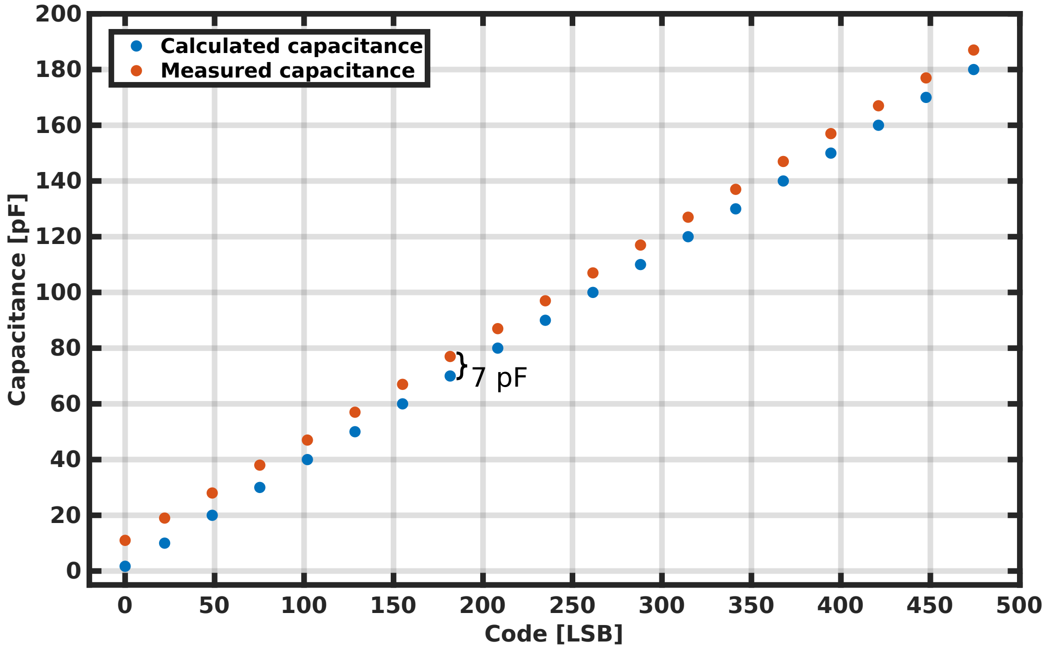


Figure 4: Measured output capacitance of the NCD2400M.

The correct operation of the device was tested using a Smart-Tweezers LCR (inductance, capacitance, resistance) meter from Ideal-tek and a vector network analyzer (VNA). The Smart-Tweezers device was used to measure the capacitance for the 472 code (180 pF) and then a VNA was used to compensate for the electrical delay of the fixture at a chosen frequency of 100 MHz, matching the measured impedance to the impedance measured by the Smart-Tweezers. Then, different codes were tried with the VNA and the outputs were plotted in Figure 4.

The reason why the Smart-Tweezers could not be used alone is the lack of precision at lower capacitance values. Measuring such low capacitance values turned out to be particularly problematic due to the instrument's lack of accuracy and possible leakage of the component's pads. However, the measurements show that the instrument responds to the programmed codes and there are strong reasons to believe that the offset arose due to the measurement procedure.

2.2.3 MAX5419

MAX5419 from Maxim Integrated is a 256-tap digital potentiometer. It covers a range of 325 Ω to 200 kΩ and operates with a power supply range from 2.7 V to 5.25 V, which makes it ideal for demonstrating the resistive sensor interface. Due to the differential nonlinearity (DNL) error and the wiper resistance that varies with the tap position, along with the fact that the actual end-to-end resistance can vary between 150 kΩ and 250 kΩ, it is better to characterize the device and store the mappings between each tap position and output resistance. The DNL error means that the slope of the resistance output is not constant and the wiper resistance variation means that the starting output resistance is a few hundreds of ohms, instead of 0Ω and varies throughout the operation range with the tap position. The typical wiper resistance is 325 Ω, up to a maximal value of 675 Ω.

Measurements of the output resistance were taken with an HP 34401A digital multimeter. The maximum DNL error was found to be at the tap point 64, with an output resistance step of 903.1 Ω, the mean value of a step being 864 Ω. The maximum differential nonlinearity error expressed in least significant bits (LSB) is 0.046 LSB, which falls within the datasheet specification of 0.05 LSB [16]. The complete plot of the differential nonlinearity error can be seen in Figure 5. The integral nonlinearity (INL) error shows the deviation of the output resistance from a straight line and is plotted in Figure 6. It stays within the 0.05 LSB boundary and corresponds overall to the graph specified in the datasheet.

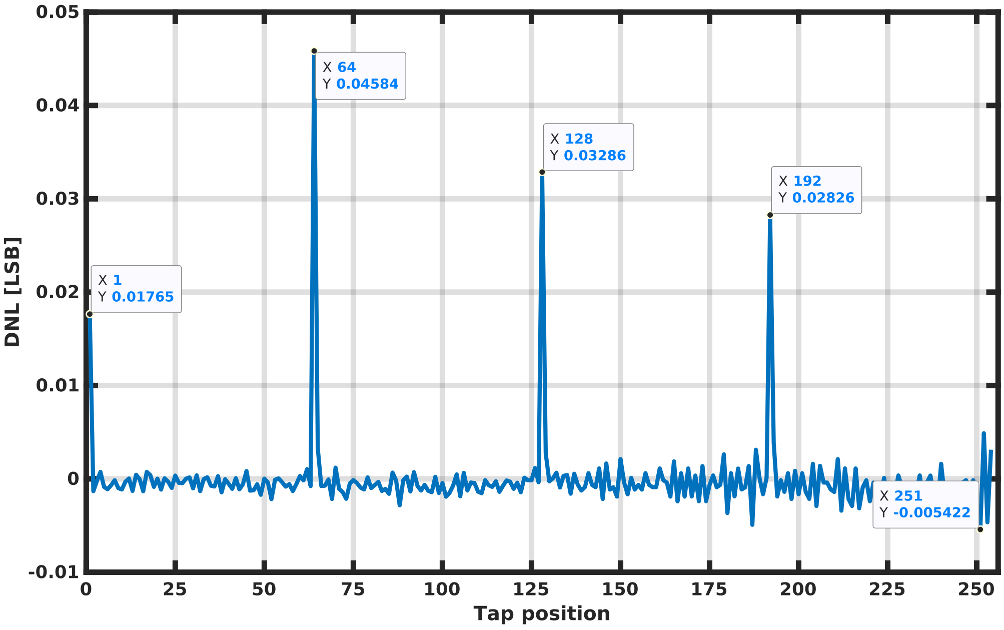


Figure 5: Differential nonlinearity error vs tap position.

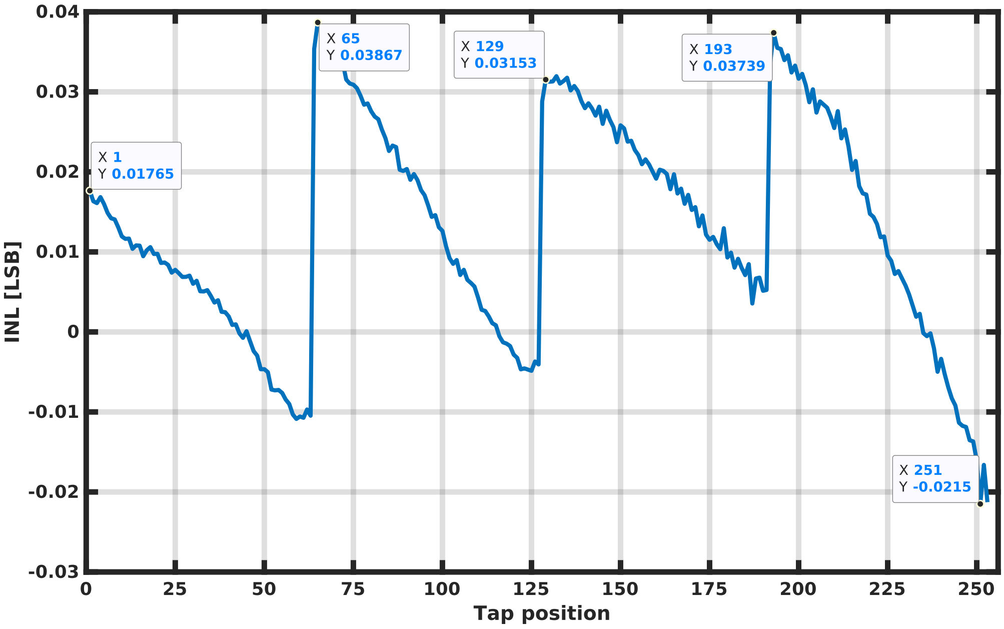
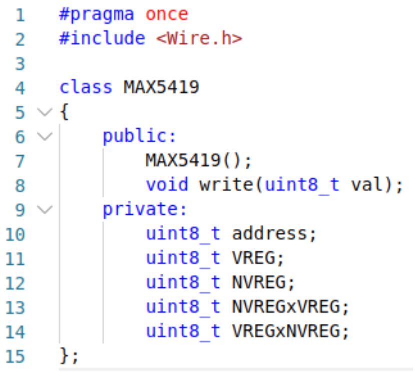


Figure 6: Integral nonlinearity error vs tap position.

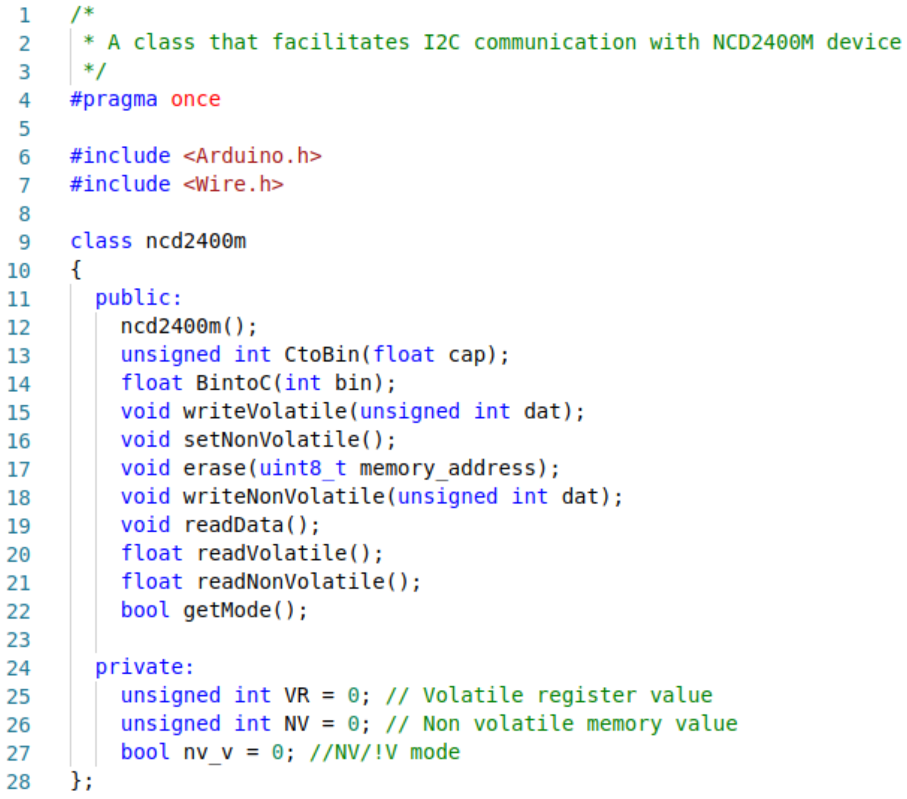
2.3 Arduino libraries

Arduino libraries are used for encapsulating extra functionalities and extending the Arduino environment. The libraries consist of classes which are written in C++. Two classes were developed for communicating and operating with the devices that were previously described. These classes are used in the final prototype and integrated with the Python interface that is described in Chapter 7. Figures 7 and 8 present the class declarations for the MAX5419 and NCD2400M devices, respectively.



```cpp
#pragma once
#include <Wire.h>

class MAX5419
{
    public:
        MAX5419();
        void write(uint8_t val);
    private:
        uint8_t address;
        uint8_t VREG;
        uint8_t NVREG;
        uint8_t NVREGxVREG;
        uint8_t VREGxNVREG;
};
```

Figure 7: Class declaration for MAX5419 device.



```cpp
/*
 * A class that facilitates I2C communication with NCD2400M device
 */
#pragma once

#include <Arduino.h>
#include <Wire.h>

class ncd2400m
{
  public:
    ncd2400m();
    unsigned int CtoBin(float cap);
    float BintoC(int bin);
    void writeVolatile(unsigned int dat);
    void setNonVolatile();
    void erase(uint8_t memory_address);
    void writeNonVolatile(unsigned int dat);
    void readData();
    float readVolatile();
    float readNonVolatile();
    bool getMode();

  private:
    unsigned int VR = 0; // Volatile register value
    unsigned int NV = 0; // Non volatile memory value
    bool nv_v = 0; //NV/!V mode
};
```

Figure 8: Class declaration for NCD2400M device.

3 Printed circuit board development

This chapter describes the development of the PCB that enables the characterization and prototyping of the IC. Section 3.1 introduces PCBs. Next, the IC footprint is discussed in Section 3.2. The layer stack is presented in Section 3.3. In Section 3.4 the controlled impedance trace design is explained. In Section 3.5 the concept of decoupling capacitance is elaborated. Section 3.6 summarizes the auxiliary components which were used for supporting the IC. Finally, the results of the PCB design are presented in Section 3.7.

3.1 Printed circuit board

A PCB is a board which offers mechanical support and electrical connections for electronic components, such as ICs and other discrete components that are mounted onto it by soldering. The main purpose of the PCB in this project is to offer an operational setting in which different blocks of the IC could be measured and connected to other devices for demonstrating the chip's capabilities. PCBs are typically composed of alternating layers of laminated copper and dielectric material, also known as substrate. In most applications, a solder resist layer is applied to the top and the bottom of the board, for protecting the PCB against oxidation and preventing unintentional solder bridges during the soldering process. Altium Designer was used as the electronic design automation (EDA) software for capturing the schematics and laying out the circuits.

A total of 8 schematic sheets contain the circuit schematics that describe the following circuits: the main chip, electrochromic display driver, energy harvester, voltage regulator, gesture sensor, narrowband transceiver, SPI communication, universal sensor interface and ADC. The final PCB consists of 232 components, having a size of 145 mm by 158 mm.

3.2 IC footprint

A footprint and custom schematic symbol were developed for the IC (Figure 9). The symbol has 109 pins, one for each of the signals that are connected to the IC die's pads. However, the footprint follows the standard QFN-100 package and therefore has only 100 pins. In order to achieve a correct mapping that allows bonding the IC die to the package, some of the ground pins were connected to the cavity of the package and the cavity was connected to only two package ground pins. Figure 9 presents this mapping in the following manner: the symbol pin numbers, which refer to the pad numbers of the IC die are positioned externally on the footprint pads, while the footprint pin numbers are positioned internally on the pads.

Because of the parasitic inductance that is associated with the leads and bonding wires of the packaged IC, the UWB transmitter block was not bonded to the package and was, therefore, left outside the scope of this thesis.

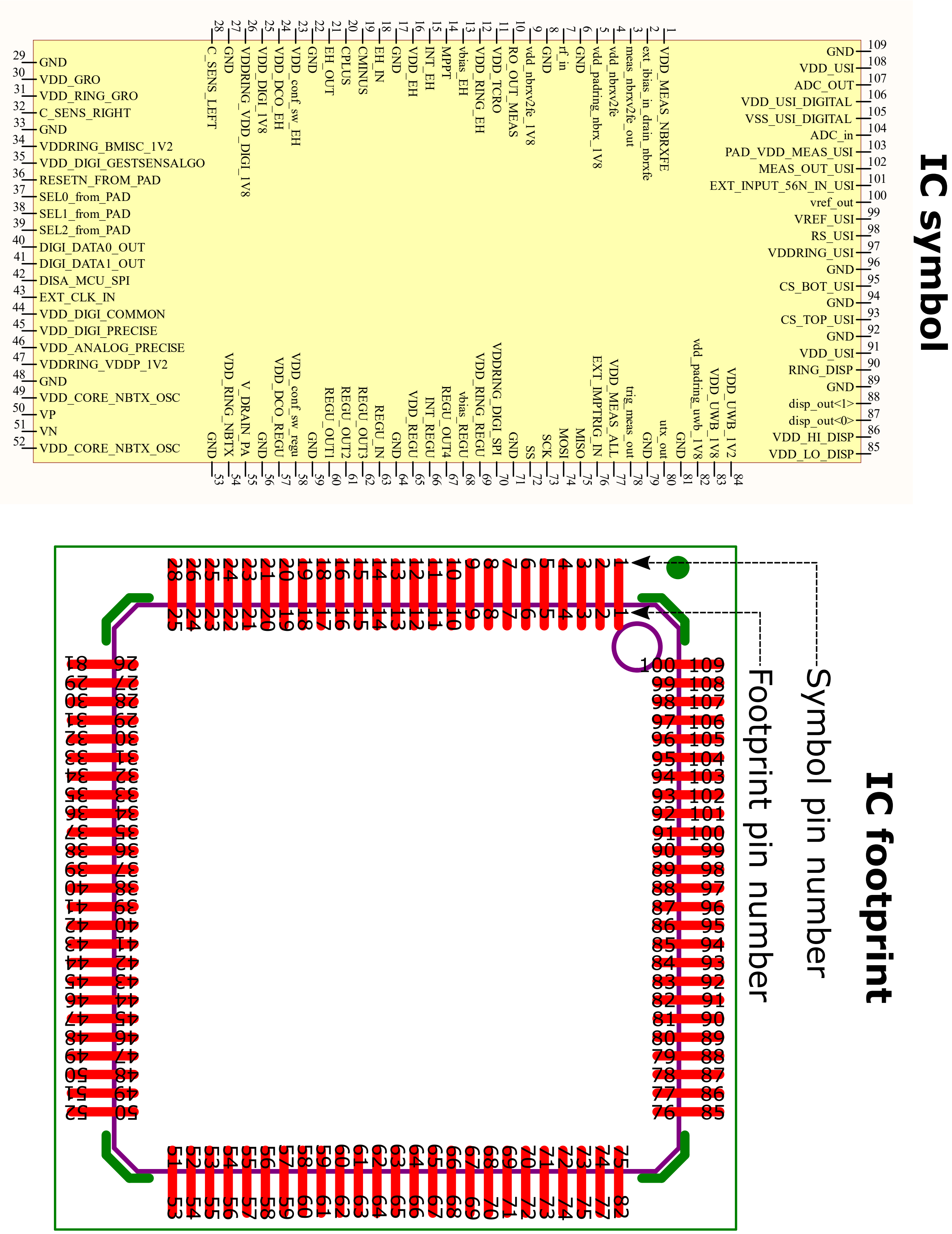


Figure 9: IC symbol and footprint.

3.3 Layer stackup

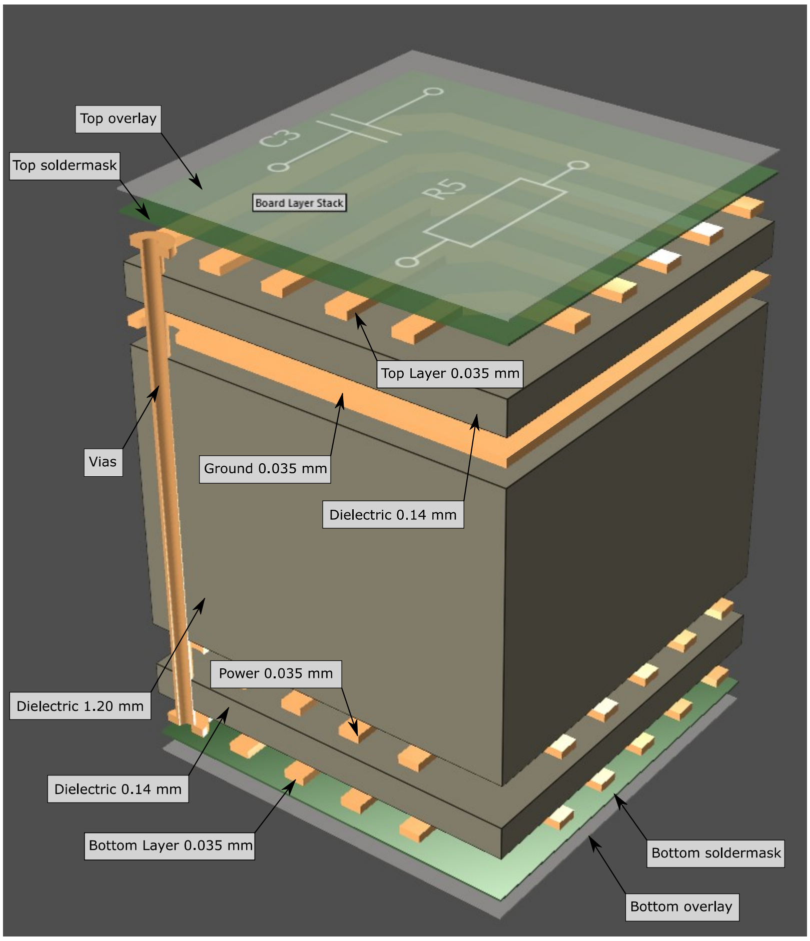


Figure 10: Layer stackup of the characterization PCB.

The layer stackup of the PCB is shown in Figure 10. Four copper layers can be observed: two signal layers (Top and Bottom), a power layer (Power) and ground (Ground). In between the copper layers, there are three dielectric layers, the prepreg being located in the middle. While the copper layers are used for transmitting the signals, the dielectric's role is to create electrical isolation between the conducting layers and thus its fabrication material is important in establishing the characteristics of the resulting transmission lines. The prepreg acts as a "skeleton", offering mechanical support to the thin layers that are above and below, in order to prevent bending. The type of the dielectric's material was chosen to be FR4, as the operating frequencies of the radio signal circuits were not high enough to demand a lower loss tangent dielectric. The thickness of the copper, dielectric and prepreg layers is 0.035 mm, 0.14 mm and 1.2 mm respectively. The total thickness of the board is 1.6 mm, ensuring that the PCB is robust enough to withstand both mechanical stress and thermal expansion.

3.4 Controlled impedance

Some of the traces that were layed out for this project need special consideration: at 434 MHz they act as transmission lines and their characteristic impedance needs to be matched to 50 Ω. Controlled impedance refers to the design of PCB traces with a predetermined characteristic impedance.

As the complex layout of the PCB requires 4 layers (Figure 10), precautions needed to be taken when designing the radio frequency (RF) transmission lines, in order not to radiate noise to other signal traces below. According to [17], placing the signal trace on the top layer and having a ground plane right below it ensures that the other two layers are electrically shielded from interferences. This arrangement is called a coplanar waveguide with ground (CPWG), also known as a grounded coplanar waveguide. A CPWG (Figure 11) is a transmission line model that comprises of a signal line which is coplanar with a ground plane on both sides, as well as another ground plane below it. The upper and lower ground planes are held at the same potential with via fences.

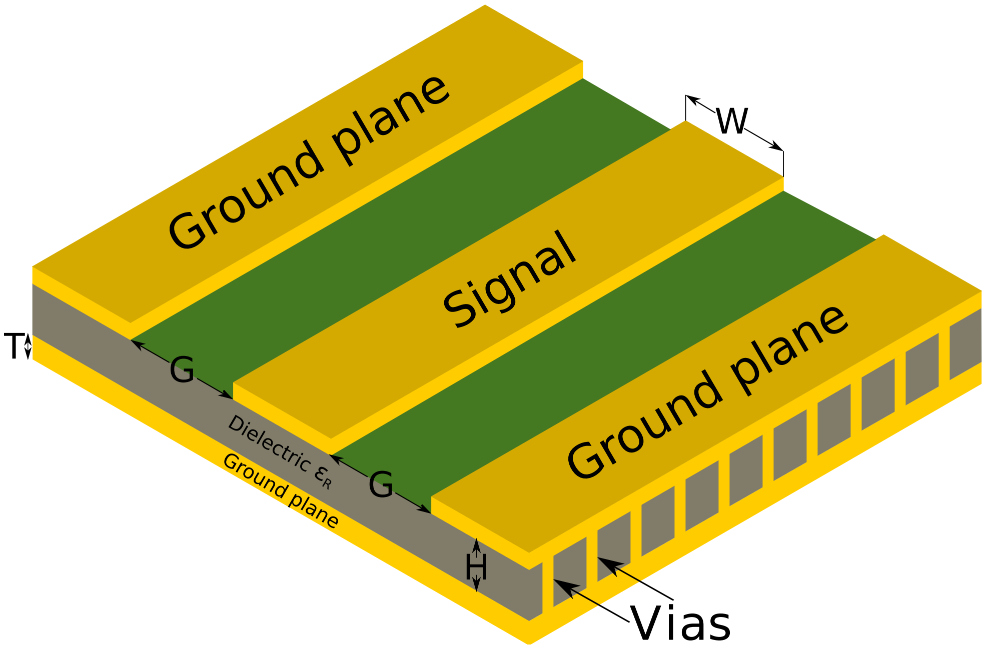


Figure 11: Coplanar waveguide with ground.

By varying certain geometrical parameters, a desired transmission line characteristic impedance can be achieved. Its most important parameters are:

- T, the thickness of the copper layers
- W, the width of the signal line
- G, the gap between the signal line and the coplanar ground layer
- H, the height between the signal (or coplanar ground layer) and the ground plane below

In addition to the geometric parameters, the dielectric constant ε_R and the dielectric loss tangent $tan\ \delta_e$ must be known at the desired operating frequency. The dielectric which was used for this PCB is a standard FR4, ISOLA Duraver DE104. The dielectric loss tangent is a critical parameter in high frequency applications, as it contributes to the insertion loss of the transmission line. FR4 is known for having inferior dielectric loss characteristics compared to other high frequency dielectrics [18]. However, at the intended operating frequency of 434 MHz, the additional cost of using a superior dielectric is not justified.

The LineCalc tool of Advanced Design System (ADS) from Keysight was used to find a suitable combination of the W and G parameters, in order to achieve a characteristic impedance close to 50 Ω. Table 1 summarizes the transmission line parameters.

Table 1: Controlled impedance trace parameters.

Parameter	**Value**
T	35 μm
W	270 μm
G	300 μm
H	140 μm
ε_R	4.4
$tan\ \delta_e$	0.002

In its CPWG model, LineCalc assumes that the top and bottom ground planes are held at the same potential. However, in practice, a via fence is used for connecting the two planes. The impact of via placement on CPWG transmission lines is presented in [17]. The study found that the distance between the signal trace and the vias which construct the fence has to be kept below one fourth of the wavelength corresponding to the transmission line's highest frequency of operation. At 434 MHz, the wavelength is approximately 82 mm. However, the distance between the vias is below 3 mm, thus ensuring correct behavior over the specified frequency range. LineCalc assumes that the electric field between the signal line and the coplanar ground planes is formed through the air. Using a solder resist over the signal and ground lines alters the effective dielectric constant of the waveguide [19]. Therefore, the portion of the PCB traces that form the transmission line was left uncovered, as suggested by the research group's past experience (Figure 12).

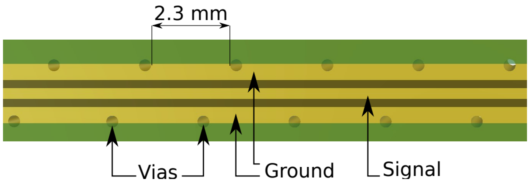


Figure 12: 50 Ω characteristic impedance trace layout.

3.5 Decoupling capacitance

The capacitance provided by the power plane, dielectric and ground ensemble is not enough to ensure a stable voltage rail, especially since the power plane is split into 0.9 V, 1.2 V, 1.8 V and 3.3 V rails. Additional decoupling capacitors are needed to ensure that the different IC blocks have enough power during switching times. The capacitors which are placed close to the IC prevent the droop in the power supply voltage for a given amount of time. The voltage droop occurs in the following way: due to the parasitic inductance of the interconnects between the power supply and the IC, when there is a change in the supplied current (caused by the switching of a logic gate, for example), according to Faraday's law, an induced voltage V_L will be dropped across the interconnect, causing a rail droop. This scenario is presented in Figure 13.

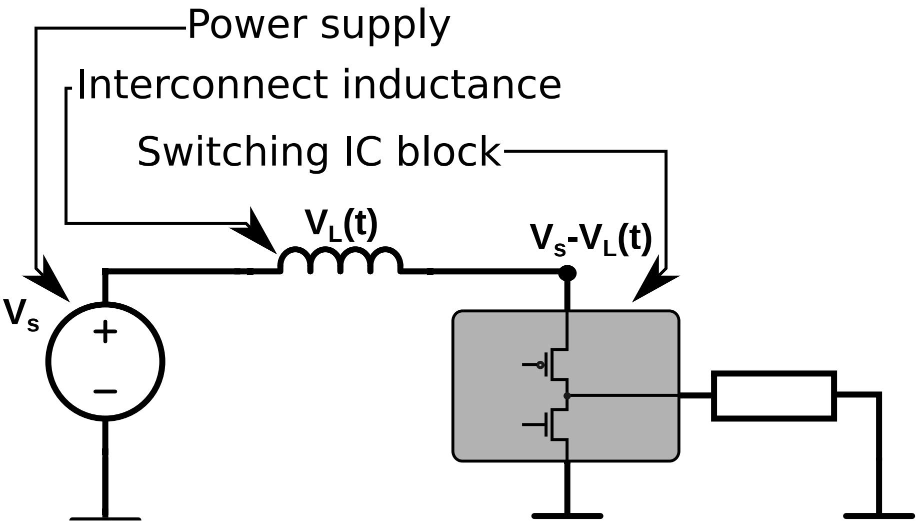


Figure 13: Voltage droop caused by interconnect inductance.

If the power supply rail collapse is to be held within a p percent of the supply voltage during a given period of time t, an expression for the required decoupling capacitance C can be derived as an adaptation from [19]. In [19] the amount of decoupling capacitance is derived as a function of absolute voltage levels, while here we examine the allowed relative voltage drop p.

The energy E_{cap} stored in a capacitor with capacitance C can be derived from

$$E_{cap} = Pt = \int_{T_1}^{T_2} VI\ dt = \int_{T_1}^{T_2} V\frac{dQ}{dt}dt = \int_{Q_1}^{Q_2} V\ dQ, \tag{2}$$

where P is the power, t is the time during which the capacitor is charged, T_1 and T_2 are the initial and final times of the charging process, V is the voltage across the capacitor's terminals, I is the current flowing through the capacitor, Q is the time-dependent charge stored on the capacitor's plate, Q_1 and Q_2 are the initial and final charges stored on the capacitor's plate.

The voltage across a capacitor is given by the well-known equation

$$V = \frac{Q}{C}. \tag{3}$$

Allowing a maximum droop percentage p of the supply voltage V_s, the initial $V_{initial}$ and final V_{final} voltages that appear at the input of the switching IC block can be modeled as

$$\begin{aligned} V_{initial} &= V_s(1-p) \rightarrow Q_1 = V_sC(1-p) \\ V_{final} &= V_s \;\; \rightarrow \;\; Q_2 = V_sC. \end{aligned} \tag{4}$$

Replacing 3 and 4 in 2 results in

$$\begin{aligned} Pt = \int_{Q_1}^{Q_2} V\ dQ &= \int_{Q_1}^{Q_2} \frac{Q}{C} dQ = \frac{1}{C}\left[\frac{Q^2}{2}\right]_{Q_1}^{Q_2} \\ &= \frac{1}{2C}\left\{(V_sC)^2 - [V_sC(1-p)]^2\right\} \\ &= \frac{1}{2C}(V_sC)^2\left[1-(1-p)^2\right]. \end{aligned} \tag{5}$$

Finally, the required capacitance can be expressed in terms of the average power dissipation of the chip (Watts), the time during which the voltage droop occurs (seconds), supply voltage (Volts) and droop percentage by using

$$C = \frac{Pt}{V_s^2\left[1-(1-p)^2\right]}. \tag{6}$$

According to [19], the decoupling capacitors have to be placed as close as possible to the IC, in order to minimize the additional interconnect inductance. Vias should be kept short and the distance between the power (and ground) plane and the decoupling capacitor should be minimized. In this project, the most critical decoupling capacitors were placed on the bottom layer, as the distance to the power layer is 0.14 mm, compared to 1.34 mm if the IC was placed on the top layer. The size of the capacitors' bodies should be small, in order to minimize their parasitic inductance. It is recommended to use multiple parallel capacitors, in order to attenuate the parasitic inductance's effects at high frequencies.

3.6 Auxiliary components

Some electronic components were selected together with the research team for supporting the operation, measurement and programming of the IC. Buffering amplifiers were needed, as some of the signals that needed to be measured were supplied by drivers that could support the measurement instruments' input impedance. For this purpose, AD8655 (single) and AD8656 (dual) precision CMOS operational amplifiers from Analog Devices were employed. One of their main applications was to buffer ADC signals, which requires the ability to operate with rail-to-rail input and output voltages. The amplifiers have a bandwidth of 28 MHz which makes them suitable for measuring the higher frequency clock signals on the board, that can go up to 10 MHz.

The IC uses 1.2 V signals and the Arduino Due board which was used for programming it uses 3.3 V signals. Level shifters are therefore needed for communicating via SPI with the chip. In addition, the 1-bit ADC output is also shifted to 3.3V, for measurement purposes. Texas Instruments' SN74AVC4T774 was chosen for fulfilling these requirements. It is a 4-bit level shifter specially designed for digital communication, supporting 1.2 V to 3.3 V bi-directional translation, with data rates up 100 Mbps.

For slave selection, a SN74HC164D from Texas Instruments serial in parallel out shift register was used as in previous designs: three parallel slave select signals (representing a 3-bit demultiplexer select) are outputted from a serial data line and a clock signal.

Figure 14 shows how slave selection is achieved at a high level. The Slave select, Serial data and Serial clock signals are provided by the device used for programming the IC. Previously, this device was an Aardvark I2C/SPI Host Adapter. In this thesis, the Arduino Due board was used for the SPI programming, as discussed in Chapter 7. The Slave select signal is routed to one specific SPI slave.

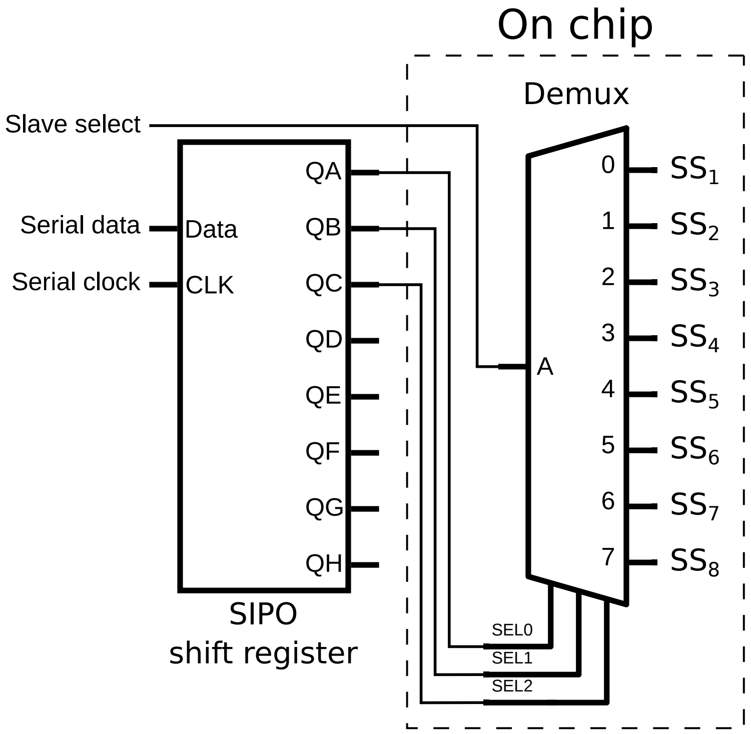


Figure 14: Slave selection.

3.7 Results

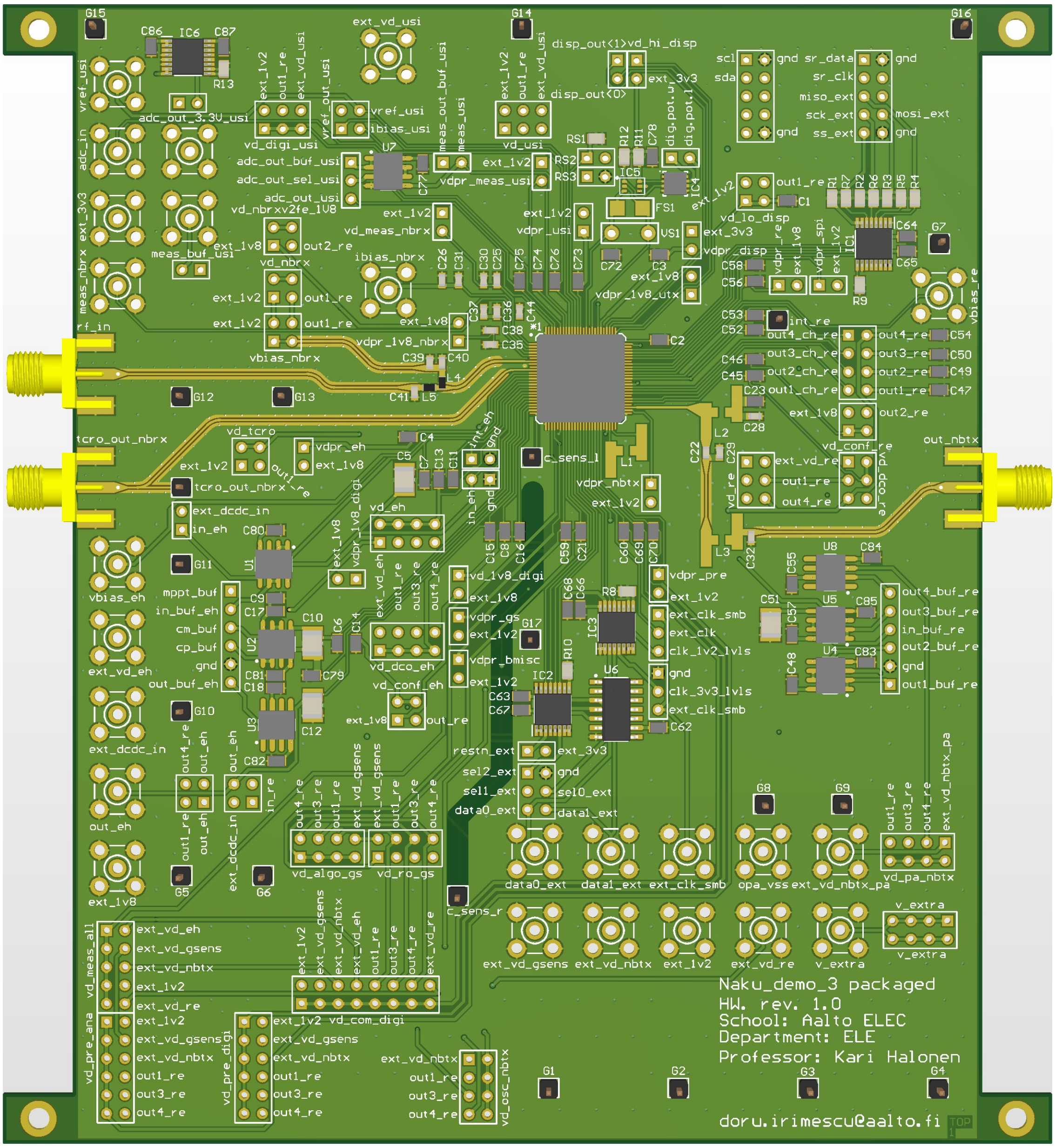


Figure 15: Top view of the characterization PCB.

Figure 15 presents a 3D rendering of the top view of the characterization PCB: the IC package can be seen, along with the supporting components and connectors. Three SubMiniature version A (SMA) connectors, located at the left and right edges of the board are used for connecting the NB receiver input, NB transmitter output, as well as for measuring the output of a temperature-compensated ring oscillator. Johnson SMA connectors from Chinch Connectivity Solutions were used for high frequency signals and SubMiniature version B (SMB) connectors from Molex were used for powering the different blocks of the IC. Although both are designed for coaxial cable connections, SMA offers a more robust connection covering a wider frequency range from DC to 18 GHz, while SMB is more compact, less robust and covers lower frequencies. Nevertheless, it is the optimal choice for supplying DC and signals with frequencies up to 4 GHz.

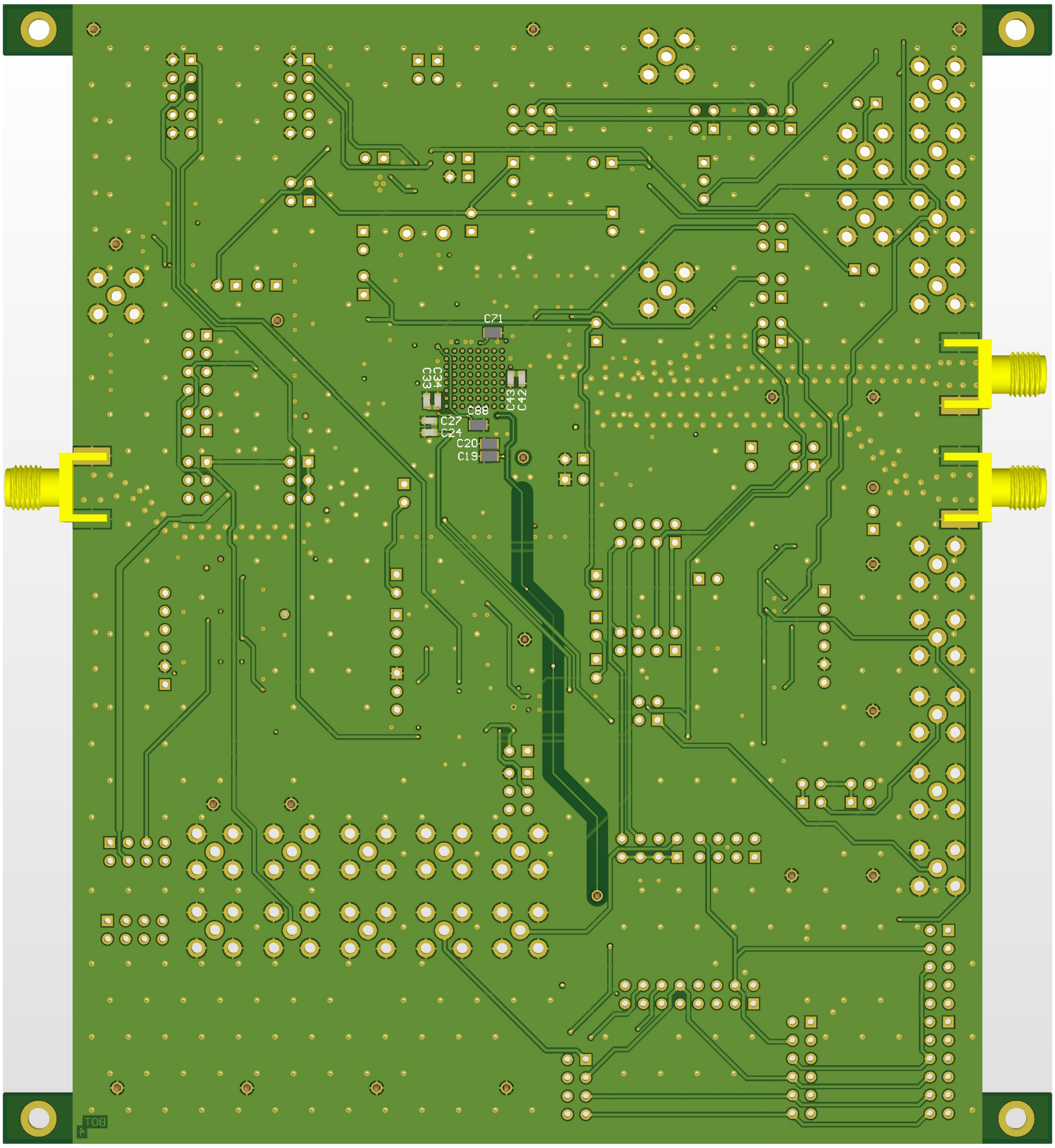


Figure 16: Bottom view of the characterization PCB.

Figure 16 presents a 3D rendering of the bottom view of the characterization PCB: next to the bottom of the IC package footprint, 10 decoupling capacitors can be observed. These capacitors can supply power to sensitive blocks demanding fast bursts of energy during signal level switching. If the placement of these components was further from the IC, the parasitic inductance would become large, preventing the supply current from rising in the required time, as discussed in Section 3.5.

4 Narrowband receiver impedance matching

This chapter documents the realization of an impedance matching network for the NB receiver, that matches the antenna to the input of the receiver front end. Section 4.1 briefly introduces the NB receiver. Section 4.2 describes the impedance matching process, discussing the VNA, the specification for the matching network, its implementation and results.

4.1 Narrowband receiver

The NB receiver is a functional block built on the chip, which implements a radio receiver that operates in the 434 MHz band. Its supported modulation schemes are on-off keying (OOK), pulse-position modulation (PPM) and differential pulse-position modulation (DPPM) with Manchester line code. A typical architecture of a microwave receiver is illustrated in Figure 17. The low noise amplifier (LNA) amplifies the signal received by the antenna, which is then fed into the bandpass filter (BPF) that rejects all the other frequencies outside the spectrum of interest. The radio frequency (RF) signal outputted by the BPF is sent into a mixer block which multiplies it with the signal from a local oscillator (LO) and thus subtracts the frequency of the LO from the frequency of the radio signal. The resulting signal is then amplified and filtered. The output signal is called intermediate frequency (IF).

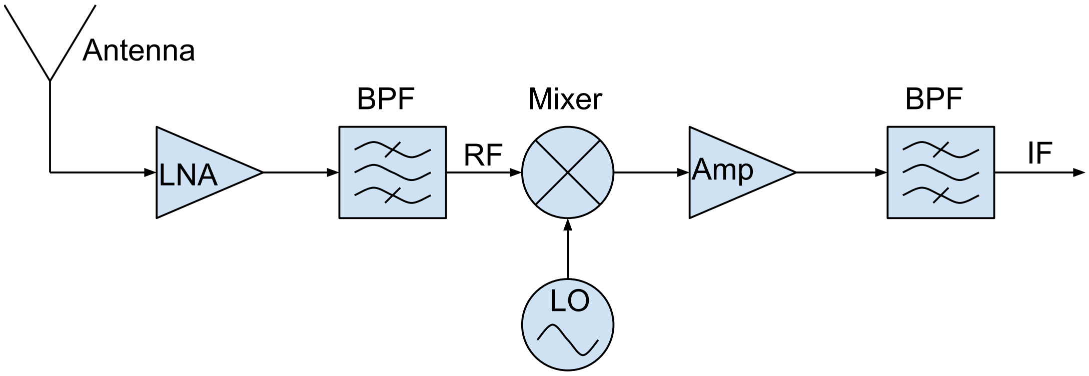


Figure 17: Typical radio receiver architecture.

4.2 Impedance matching

When power is being transmitted from a source to a load, the load's impedance has to be equal to the complex conjugate of the source's impedance in order to ensure a maximum power transfer. As this is often not the case, special matching networks have to be designed in this respect. It is crucial that the network consists only of reactive components, so that all the transmitted power is actually consumed by the load's resistive impedance. Matching networks can be achieved with either lumped components or transmission lines. In this section, a lumped-component matching network is presented.

4.2.1 Vector network analyzer

The VNA is an important measurement device in the industry of RF electronics. Its name suggests that it can measure simultaneously both the magnitude and the phase of a signal, in order to characterize an electrical network. By taking advantage of bespoke calibration standards, the VNA can compensate for systematic errors that arise from the cables, fixtures or the VNA itself, in order to give very accurate measurements, such as:

- S parameters
- Impedance
- Standing wave ratio
- Electrical length and delay.

The simplified principle of the VNA's operation is that an incident signal produced by the VNA is compared to either the reflected or the transmitted signal from the device under test [20]. The device used for designing the matching network is an Agilent 8722ES S-parameter Vector Network Analyzer.

4.2.2 Impedance matching specification

The impedance matching network has to match a 50 Ω RF input to the on-chip radio receiver at a frequency of 434 MHz. The matching is considered appropriate if the input return loss $-20log_{10}(|S_{11}|)$ is above 10 dB at the target frequency. This scenario is represented in Figure 18, where the chip and the RF input can be observed, accompanied by the L matching network in the middle.

The L matching network architecture is comprised of the two black passive components, one in series and one in shunt connection (C_1 and L_1 in Figure 19), together with a third gray passive component (C_2 in Figure 19) that creates an AC ground for the matching network, decoupling the amplifier biasing line from the rest of the network. The trace which provides the biasing DC voltage to the amplifier input (Z_{o2} in Figure 19) is the one parallel to the main trace in Figure 18. If C_2 was not used, the biasing trace would act as a transmission line between the L network and the ground, severely modifying the response of the matching network.

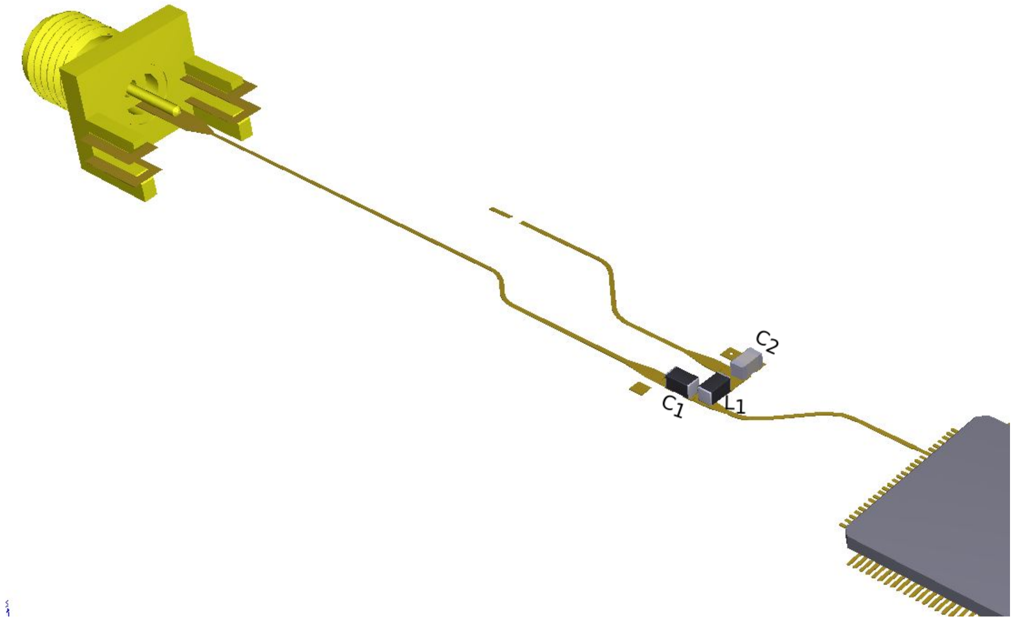


Figure 18: Antenna matching network.

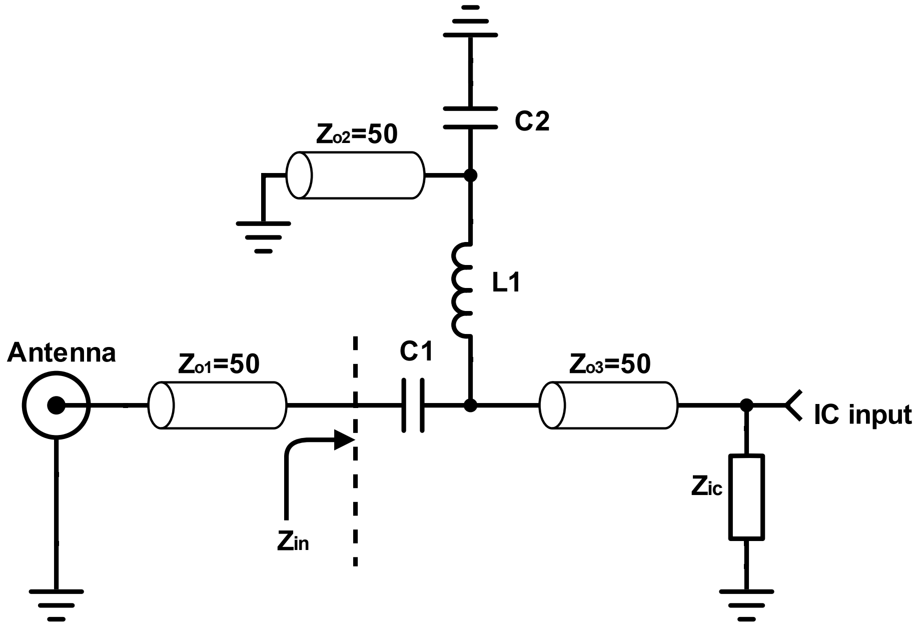


Figure 19: Schematic representation of the matching implementation.

4.2.3 Impedance matching implementation

The implementation process begins by measuring the input impedance looking towards the IC input at the precise point where the filter is located on the PCB. Normally, this would not be an easy task to achieve, since the only point for connecting a measurement device to the trace is located at the antenna input. With the aid of the VNA, the electrical length between the antenna connection and the point where the filter is located (Z_{o1} in Figure 19) can be removed.

The reflection coefficient S_{11} can be expressed as a function of the distance z from the load, as [21]:

$$S_{11}(z) = \Gamma_L e^{2\gamma z}, \tag{7}$$

where Γ_L is the reflection coefficient measured at the load and $\gamma = j\frac{2\pi}{\lambda}$. It is trivial to observe that, for a given frequency and real Γ_L, the phase shift of the reflection coefficient is:

$$arg[S_{11}(z)] = \frac{4\pi}{\lambda}z. \tag{8}$$

Moreover, the S_{11} parameter's phase shift at the load ($z = 0$) is 0 degrees and its magnitude remains unchanged at any distance away from the load.

Removing the electrical length of Z_{o2} was achieved in practice by creating an open at the point where the filter was placed (opening C_1 and L_1) and measuring the phase of the S_{11} parameter while adjusting the electrical delay of the measurement. The electrical delay was adjusted so that at 434 MHz there was a zero-degrees phase shift. Once this calibration had been established, C_1 was replaced with a short and the measured input impedance was measured to be $Z_{in} = 6 - j105$ Ω. The location of Z_{in} is shown in Figure 19.

The input impedance is plotted on a Smith chart, seen as the yellow dot (Z_{in}) in Figure 20. A suitable value for L_1 was chosen such that the impedance reached the 50 Ω constant resistance circle, seen as the green dot (Z_{mid}) in Figure 20. Finally, an appropriate value for C_1 was chosen such that the impedance reached 50 Ω, seen as the red dot in Figure 20. At this point, a reference value for the inductance and capacitance was achieved. However, due to the non-ideal component characteristics, the actual impedance is never the same as in the simulations. Therefore, the C_1 capacitor was first replaced with a short and several values for the L_1 inductor were tried on the PCB, while measuring the input impedance with the VNA. Then, starting from the reference value that had been theoretically obtained for the capacitor, several capacitors were tried on the PCB in order to reach 50 Ω.

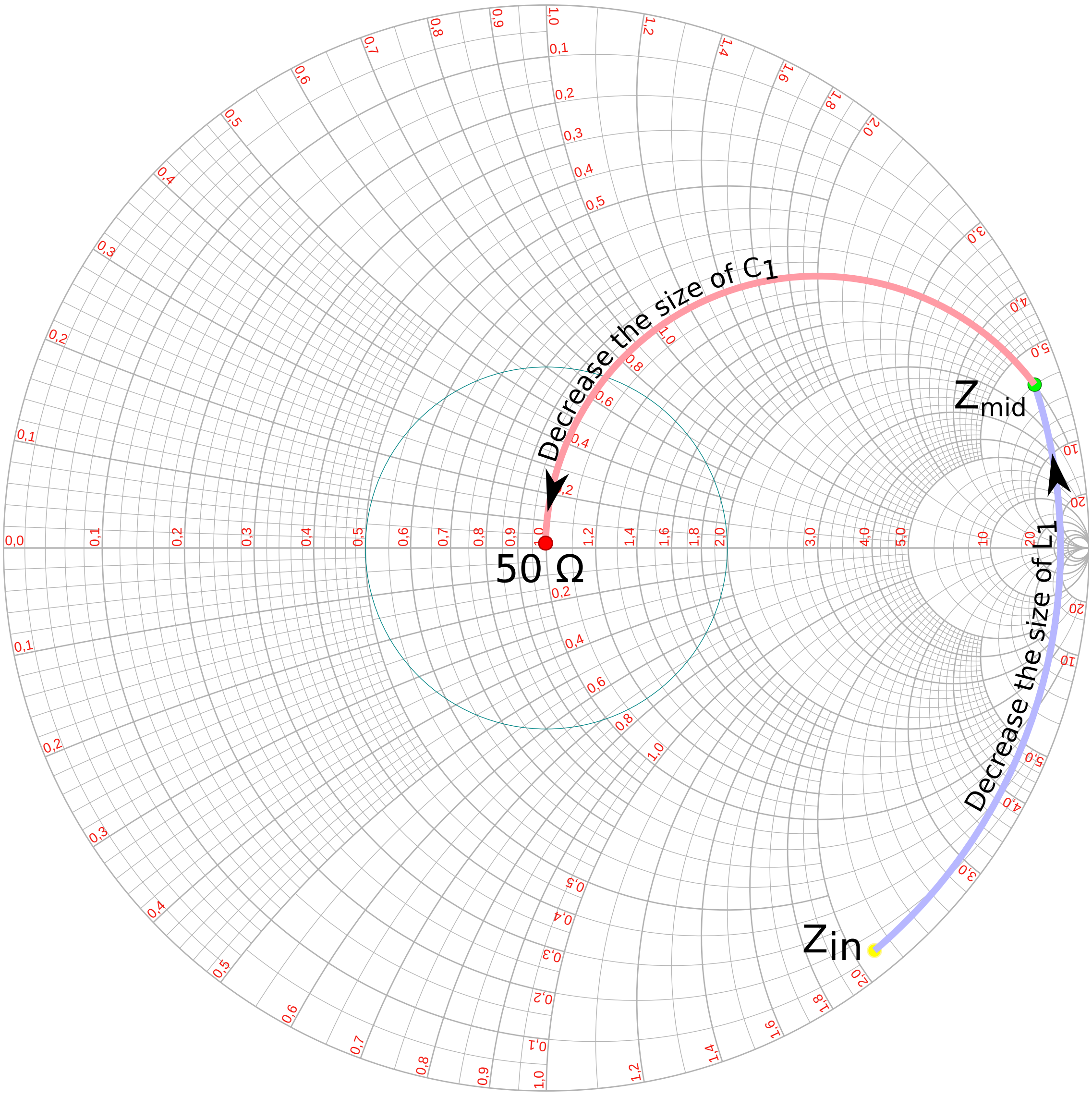


Figure 20: Smith chart plot of the impedance matching process.

The end result is an impedance matching of 18 dB at 434 MHz, making possible the wireless communication described in Chapter 6.

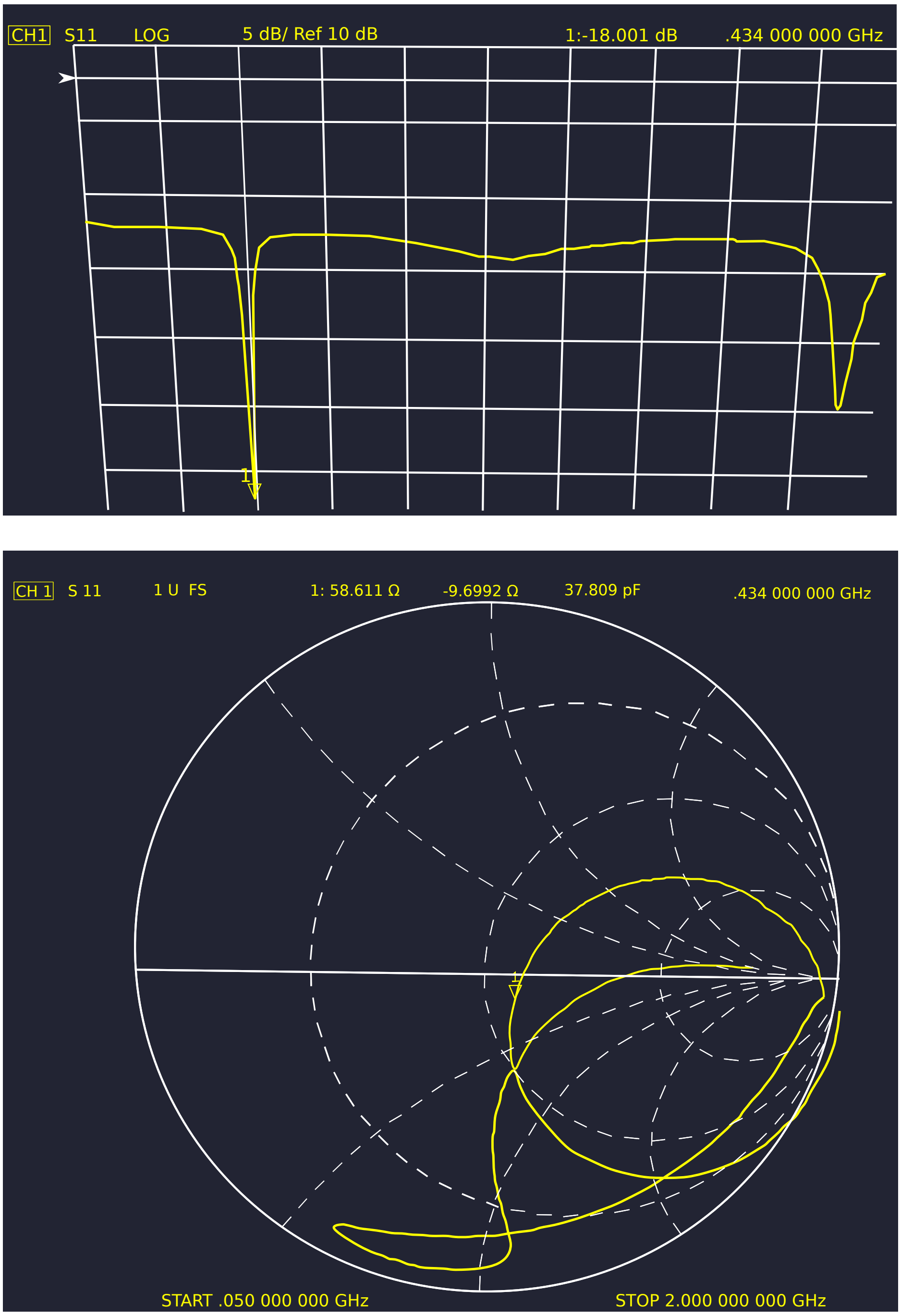


Figure 21: Impedance matching measurement.

5 UWB filter design

This chapter describes the necessary theory and the implementation of a BPF, used to match the ultra-wideband impulse radio amplifier output to the transmitter antenna. The filter is a two-port matching network, which ensures that the maximum power is delivered from the amplifier to the antenna in the passband with an acceptable group delay, while reflecting power in the stopband spectrum. Section 5.1 introduces the scattering parameters. Section 5.2 describes the analog filters, with an emphasis on BPF specifications. Section 5.3 presents the non-ideal model of passive components used for developing the filter. Finally, Section 5.4 reports the design of the filter, elaborating the specifications, the implementation and its results.

5.1 Scattering parameters

The scattering parameters, or the S-parameters characterize linear circuits, being particularly useful in RF applications. Measuring directly the z or y parameters requires the ports of the network to be alternatively shorted and opened [22]. In many practical cases, this is infeasible and the S-parameters come as a viable alternative for describing the operation of two-port networks, such as filters and matching networks.

The S-parameters describe electrical networks in terms of incident and reflected voltages for a port,

$$S_{ij} = \left.\frac{V_i^-}{V_j^+}\right|_{V_k^+=0,\ k\neq j} \tag{9}$$

where V_i^- denotes the reflected voltage at port i, V_j^+ represents the incident voltage at port j and $V_k^+ = 0$ indicates that all the incident voltages at the other ports are 0. Thus, S_{ii} denotes the reflection coefficient of port i and S_{ij} represents the transmission coefficient from port j to port i.

In this thesis, the focus is only on two-port networks. A two-port network is called symmetric, if $S_{11} = S_{22}$ and reciprocal if $S_{21} = S_{12}$. The S parameters do not depend only on the properties of the electrical network, but also on the reference impedance, which is commonly chosen as 50 Ω in RF applications, this also being the case for this thesis.

As the S parameters are calculated for the same reference impedance, it is common to express them in decibels and postulate the descriptive names [22]: return loss $= -20log_{10}|S_{11}|$ and insertion loss $= -20log_{10}|S_{21}|$.

5.2 Analog filters

Analog filters are passive or active electrical networks that are used for changing the shape of signals in the frequency domain. Ideally, a filter should have a unity gain in the pass band, that is, the range of frequencies that characterize voltages which should not be attenuated by the filter. Consequently, a filter should have a zero gain inside the stopband, that is, at all the other frequencies that are not in the passband.

For a practical filter, we need to define the passband ripple A_{max}, which is the maximum gain variation of the filter's amplitude response in the passband, as well as minimum stopband attenuation A_{min}, that specifies a desired upper bound on the attenuation in the stopband. The steepness of the filter, called "roll-off", which describes how fast the response transitions between passband and stopband is defined as the order of the filter, N. For a BPF, the stopband is typically defined in terms of w_{s1} and w_{s2} and the passband is defined in terms of w_{p1} and w_{p2}. Consequently, a lowpass or highpass filter stopband and passband would only be defined in terms of w_s, w_p respectively. Figure 22 presents these general BPF specifications.

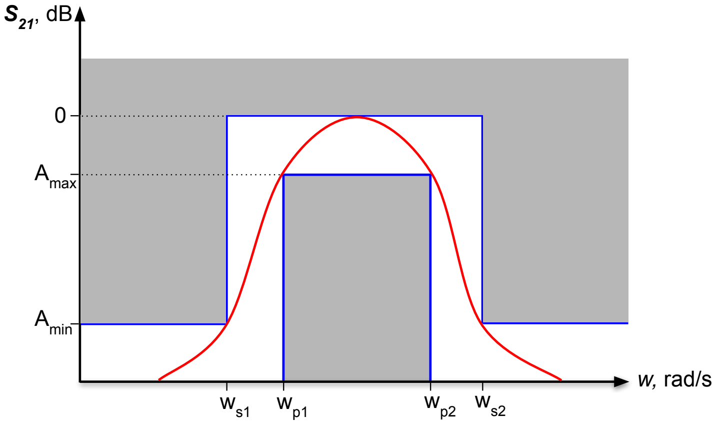


Figure 22: BPF specification.

As of this point, only the amplitude specifications of the filter have been discussed. Nevertheless, as the signal passes through the filter, a phase shift occurs. For an ideal filter, the phase shift is linear throughout the whole frequency spectrum, while in practice the phase response is a function of frequency

$$\phi(w) = arg\{S_{21}(jw)\}. \tag{10}$$

Thus, if a sine wave which completes 2π radians in T seconds is subjected to a phase shift of $\phi(w)$ rad/s, it is being delayed by $P(w)$ seconds, which can be expressed as:

$$\frac{-\phi(w)}{2\pi} = \frac{P(w)}{T} \rightarrow P(w) = -\frac{\phi(w)T}{2\pi} = -\frac{\phi(w)}{2\pi f} \rightarrow P(w) = -\frac{\phi(w)}{w}. \tag{11}$$

$P(w)$ is called the phase delay, showing how much a sine wave passing through the filter is delayed.

Another measure of the filter's phase response is the group delay, which is a measure of phase nonlinearity: [25]

$$D(w) = -\frac{d}{dw}\phi(w). \tag{12}$$

When the phase shift is linear, the group delay and phase delay give the same result.

Once the specifications of the filter are clarified, the next step is to find a transfer function named the filter approximation which fulfills the given requirements. It is customary to either derive the filter approximation from tables, or use specialized software for the task.[24]

The most common filter approximation methods are:

- Bessel : maximally flat phase response, gentle roll-off.
- Butterworth: steepest roll-off, without passband ripple. Worse phase response than Bessel.
- Chebyshev: superior roll-off to Butterworth, with a proportionally large passband (or stopband) ripple. Requires less components for realization than the Elliptic filter.
- Elliptic (Cauer): steepest roll-off, with ripple present in both passband and stopband. Highest phase nonlinearity among the four alternatives.

5.3 Passive component model

A non-ideal capacitor exhibits, besides its capacitance, an equivalent series resistance (ESR), as well as an equivalent series inductance (ESL), mainly because of the component's leads. These characteristics start becoming problematic at high frequencies, as the capacitor becomes a series RLC resonant circuit.

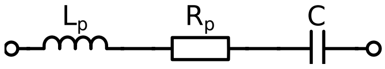


Figure 23: RF model of a surface-mounted (SM) capacitor C including the effect of parasitic inductance L_p and resistance R_p.

A non-ideal inductor exhibits, similar to the capacitor, an equivalent series resistance and a parallel capacitance, mainly because of its leads and windings.

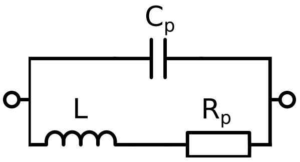


Figure 24: RF model of an SM inductor L including the effect of parasitic capacitance C_p and resistance R_p.

Thus, the designer needs to ensure that the component is modeled accordingly and the self-resonant frequency is accounted for in the simulation. At high frequencies, the parasitic resistances become frequency-dependent and are harder to be properly simulated. Component manufacturers provide S-parameter files that accurately describe their components' behavior for a specified frequency range and can be incorporated in EDA software tools such as the Advanced Design System (ADS) from Keysight.

5.4 Filter design

5.4.1 Filter specification

The purpose of the UWB filter under design is:

- To couple a DC biasing signal to the RF amplifier
- To match the RF amplifier's output to a transmitting antenna.

The inputs to the filter and the quality of their signals are presented in Figure 25.

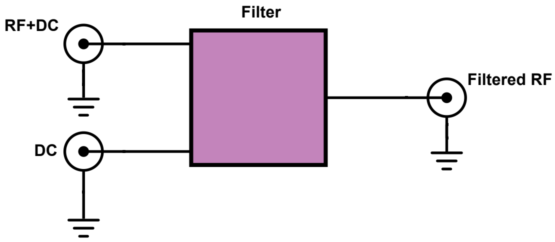


Figure 25: Filter inputs and outputs.

The filter is designed for the European generic UWB mask and it needs to have a passband of 2.5 GHz, starting from 6 GHz up to 8.5 GHz [26]. The maximum passband ripple should be no larger than 5 dB. The stopband frequency specifications are 2 GHz and 12.5 GHz, at which the minimum attenuation should be 25 dB. The group delay of the filter should not be larger than 150 ps. The reference impedance is 50 Ω for both of the filter ports.

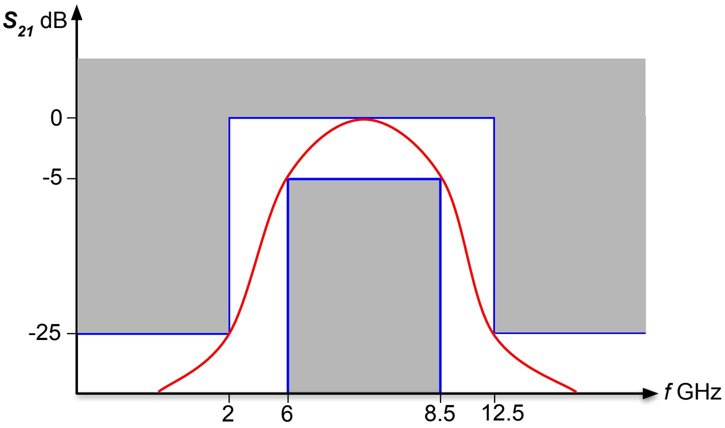


Figure 26: Filter specification.

In addition to these specifications, due to the limited PCB space and for decreasing costs, the filter should use a minimum amount of components and have a minimal area.

5.4.2 Filter implementation

The filter implementation began by designing a second-order Chebyshev filter, using the ADS. Then, a suitable family of components was selected and their S-parameter files (achieved from the manufacturer's website) were used for accurately simulating the real component behavior. For the filter inductors and capacitors, the AVX Accu-L series and AVX Accu-P series were used, respectively. The size of the components which realize the Chebyshev approximation was chosen to be 0603 (in metric units). A special component was selected for realizing the AC ground that decouples the DC signal from the rest of the filter, namely ATC 560L, an ultrabroadband DC-blocking capacitor, which has a size of 1005 (in metric units). Figure 27 shows the schematic of the filter: J1 and J2 are the input and output ports of the filter and J3 is the DC biasing port, which is decoupled by C3. The four components C1, L1, C2 and L2 together make up the Chebyshev filter.

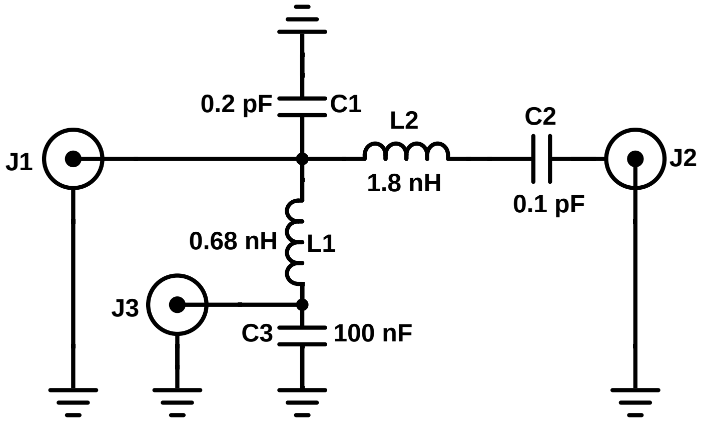


Figure 27: Schematic of the filter implementation.

As the operating frequency of the filter is in the order of gigahertz, the PCB design can have a dramatic effect on the performance of the filter, due to parasitic reactance and large electrical lengths. Altium Designer was used for laying out the PCB and the Momentum 3D EM simulator was used for improving the design to match the specifications (Figure 28).

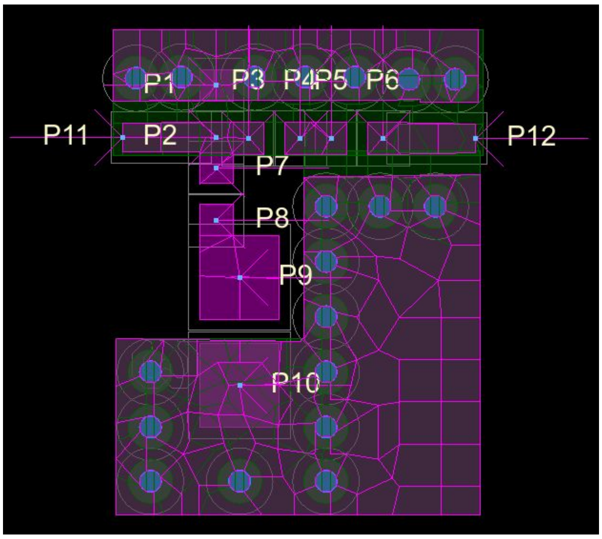


Figure 28: Layout of the filter implementation.

Figure 29 presents the simulated results of the BPF implementation. The reflection coefficient is below -5 dB between 6 GHz and 9.5 GHz, peaking at -14 dB at 7.9 GHz. The transmission coefficient is above -4 dB between 6 GHz and 9 Ghz, with a stopband attenuation below 30 dB, thus satisfying the given specifications. A maximum group delay of 140 ps was achieved in the passband, below the required upper limit of 150 ps.

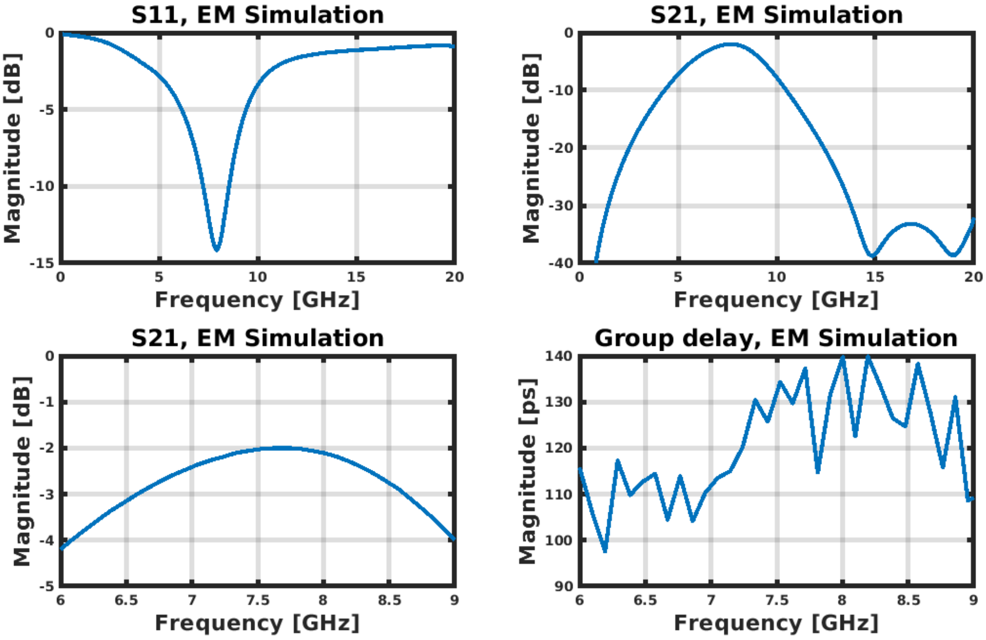


Figure 29: Filter implementation simulated results.

6 Wireless communication setup

For testing and demonstrating the wireless communication capabilities of the IC, a setup comprising of a computer, an universal sofware radio peripheral (USRP) and a PCB supporting the integrated circuit is assembled. The USRP device is presented in Section 6.1. A computer which is running National Instruments' (NI) LabVIEW (discussed in Section 6.2)-based software is communicating via a gigabit ethernet connection with the software radio peripheral. Cyclic redundancy check (CRC, presented in Section 6.3) codes are used to ensure that the integrity of the messages is not compromised. The USRP is an Ettus Research USRP N210 model, streaming Manchester (Section 6.5) OOK (Section 6.4) data to the IC and receiving differential pulse position modulated (DPPM, discussed in Section 6.6) OOK data from the IC.

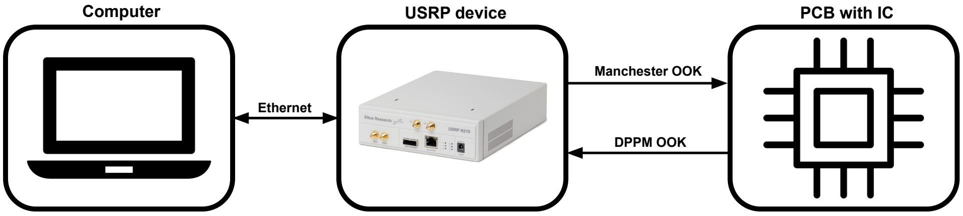


Figure 30: Wireless communication setup.

The LabVIEW design of the Manchester transmitter is reported in Section 6.7. Section 6.8 describes the development of the DPPM receiver, which comprises two parts: implementing a DPPM transmitter simulator, which eases the design process and the actual receiver.

6.1 USRP

USRPs are sofware-defined radio (SDR) hardware peripherals, used in various RF applications for transceiving radio signals. Signal processing software such as LabVIEW or GNU Radio is commonly used in order to stream signals to and from the peripheral.

A typical USRP block diagram is presented in Figure 31. The radio device used in this thesis is an USRP N210 by Ettus Research, which offers a frequency range of DC - 6 GHz, a 14-bit analog to digital converter (ADC) at 100 MS/s, a 16-bit digital to analog converter (DAC) at 400 MS/s and a 25MHz RF bandwidth with 16 samples [28].

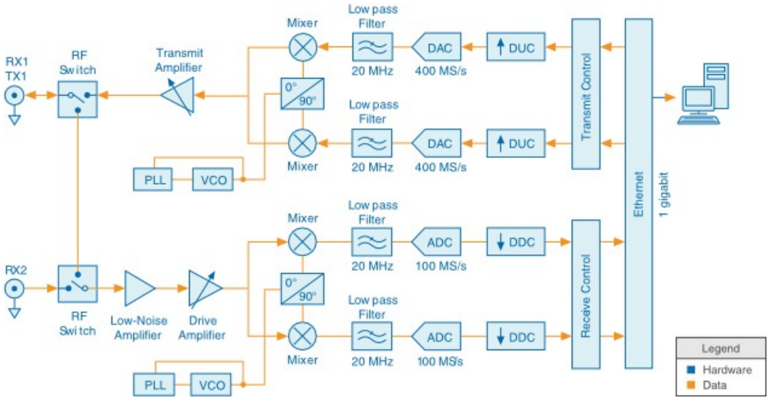


Figure 31: USRP block diagram. From [28].

6.2 LabVIEW environment

LabVIEW is a software development environment for fast development of engineering applications that require testing, measurement or control. Some of its main features are the enhanced hardware integration with software defined radios, measurement equipment or FPGA-based embedded computer hardware, to name a few. Programs are designed in a visual manner, the programmer having to connect blocks with wires which represent data rather than writing code in a traditional way. This programming paradigm is referred to as dataflow programming. Some advantages of the development environment in question include rapid software design, built-in special functions such as FIR filters or PID algorithms, built-in data visualization tools and a graphical approach to parallel programming [29].

A LabVIEW application implements a virtual instrument (VI) that comprises a front panel and a block diagram. The front panel includes various interactive user interface elements, through which the engineer can visualize and modify the desired operational parameters of the instrument. A block diagram describes the required technical behavior of the application, by making use of dataflow programming methods. LabVIEW applications are cross-platform, as they can be deployed on operating systems such as Windows, Linux Solaris, NT embedded or Mac [30].

As an example, the front panel of a cyclic redundancy code (CRC) generator which was designed for the DPPM receiver is presented in Figure 32. Consequently, the block diagram of the same virtual instrument is presented in Figure 33.

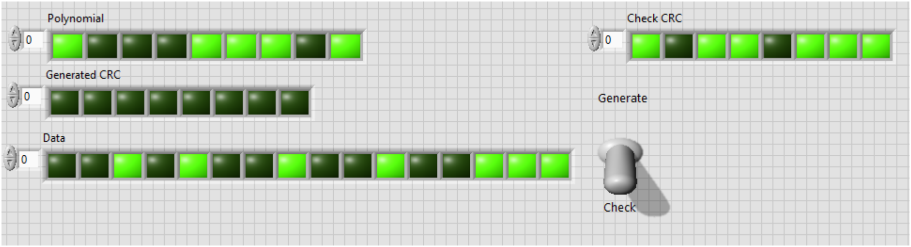


Figure 32: CRC VI front panel.

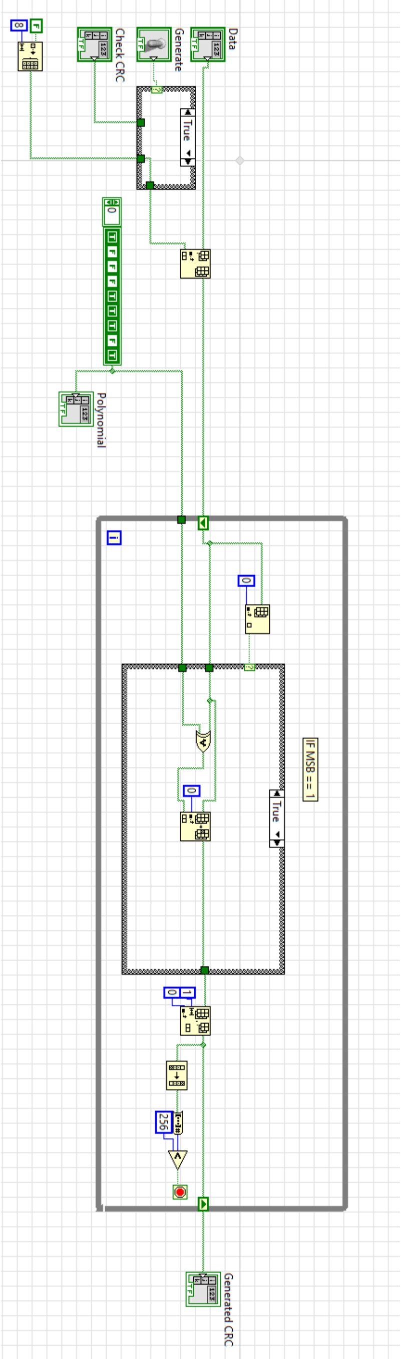


Figure 33: CRC vi block diagram.

One handy advantage that LabVIEW offers is that it allows hierarchical design, through the use of sub VIs. As a result, the CRC generator can be included in later designs (such as the DPPM receiver) as a stand-alone block called a sub-VI.

6.3 Cyclic redundancy check

When a message is being sent over a communication channel, it is practical for the receiver to know if the integrity of the data has been altered or not. By using an error detecting code, the transmitter and receiver can ensure that the data is correctly communicated over the channel. Parity bit is the simplest error detecting code, where the parity of the data's binary representation is appended to the end of the message. The receiver calculates the parity of the data, compares it to the parity bit sent by the transmitter and requests another transfer, if the data is corrupted. However, parity bits fail to catch many types of errors that may occur during transmission (for example, if two bits are being inverted).

CRC is an error-detecting code which is commonly used in digital communications, particularly suitable for applications which involve detecting noise-induced errors. Its name comes from three facts: (i) it is based on cyclic codes; (ii) it is redundant in the sense that it does not add any new information to the payload; (iii) it is intended for checking the integrity of the data [31]. Being based on polynomial division, CRC makes use of a generator polynomial, which is commonly agreed between the transmitter and receiver. The degree of the generator polynomial, denoted by n, sets the type of the CRC-n code, some common values for n being: 4, 8, 16 and 32. Parity bits are a special case of CRC, where the degree of the generator polynomial is 1. When choosing the degree of the generator polynomial, it is important to note three factors [32]. First, the computation time will increase with a higher degree polynomial. Second, a CRC-n code will only output one of 2^n unique values for any given message, the size of the data word impacting the performance of a given CRC-n. Third, the selection of the polynomial severely affects the performance of a given CRC-n.

For the purpose of demonstrating the capabilities of the NB transmitter and receiver, the data word size varies between 12 (one 6-bit time reference symbol and one 6-bit command) to 48 bits (one 6-bit time reference and 7 commands comprising 42 data bits). A CRC-8 code was chosen for this project as it provides a good performance for the given data size [32] range.

6.4 On-off keying

OOK is a digital passband modulation technique where the amplitude of the carrier wave can take two values, typically a one (fully on) or a zero (fully off). It is the simplest form of amplitude shift keying (ASK). ASK is a type of amplitude modulation used for the transmission of digital data, where ones and zeros are encoded by varying the carrier signal's amplitude. Figure 34 represents the transmission of a message consisting of "0101101001" using OOK. In this project, OOK is the only digital bassband modulation which is used. A computer transmits data using OOK-encoded Manchester modulation and the IC transmits data using OOK-encoded DPPM.

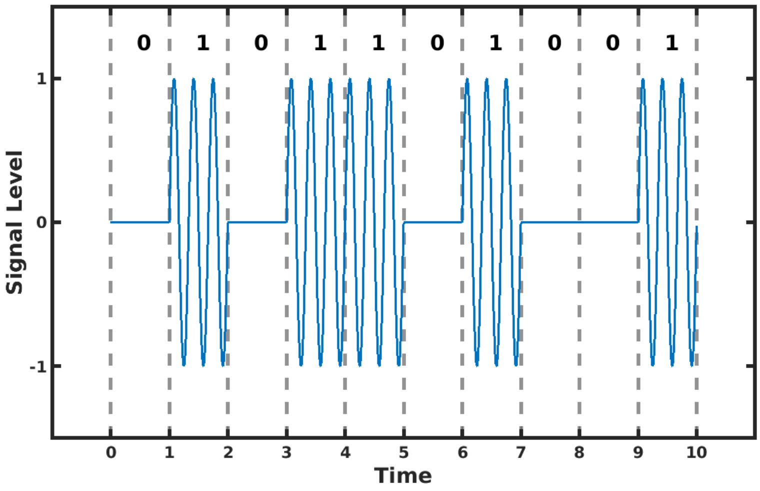


Figure 34: On-off keying.

6.5 Manchester code

Manchester code is a baseband modulation technique, where each bit is encoded as a signal transition either from high to low, or from low to high. The produced signal is self-clocking and thus there is no need for synchronization between the receiver and the transmitter. Moreover, the signal does not present a DC component. Figure 35 represents the transmission of a message consisting of "01100" encoded with Manchester code and transmitted via OOK. In Figure 35, a "0" is represented as a low to high transition.

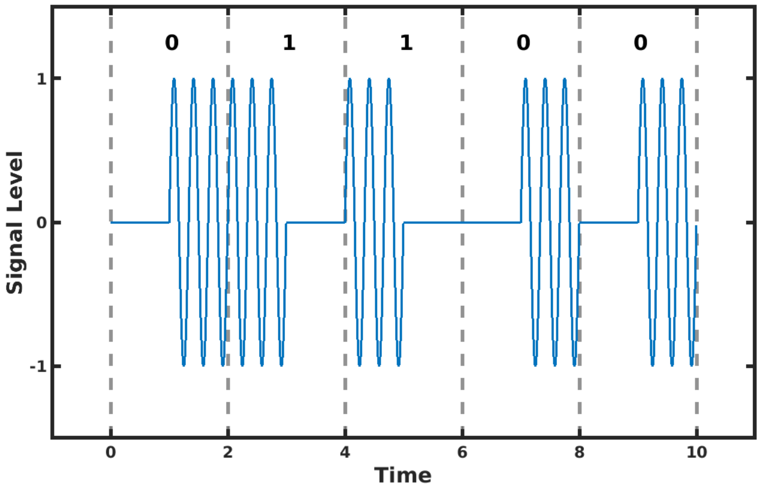


Figure 35: Manchester code with OOK.

6.6 Differential pulse position modulation

DPPM is a line coding technique (also called baseband modulation), where the data to be transmitted is represented as time differences between a sequence of pulses. Its main advantage over the more common pulse position modulation, where the position of one pulse with respect to the beginning of the symbol defines the encoded data, is that the transmitter and receiver do not need to be synchronized [27].

For example, "0" is encoded as "101", "1" is encoded as "1001", "2" is encoded as "10001" and so forth. Figure 36 illustrates a scenario where a message "0 1 2" is encoded as "1010010001" with DPPM and transmitted via OOK.

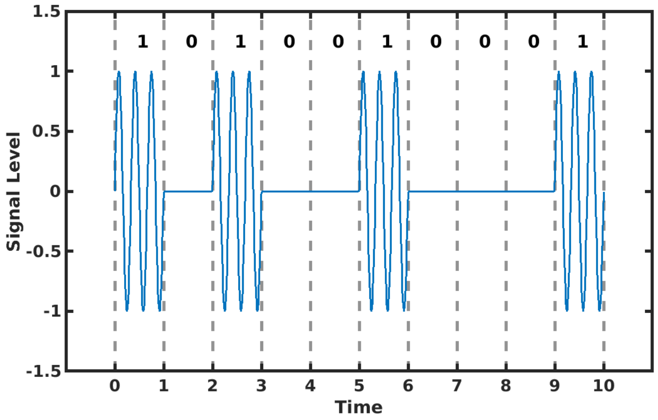
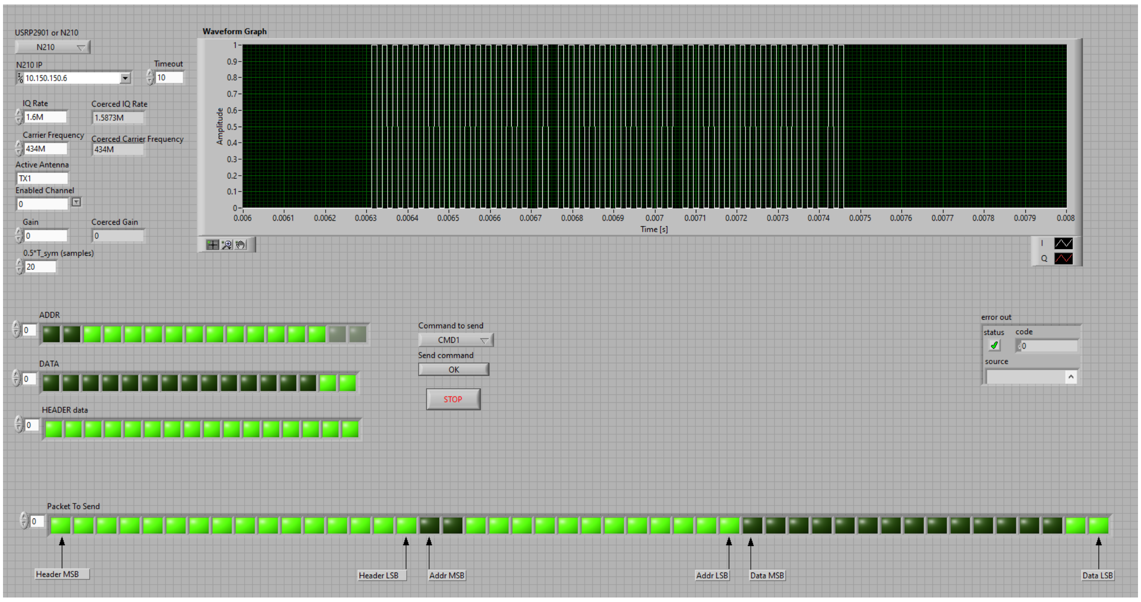


Figure 36: DPPM with OOK.

Each individual one or zero takes the same duration: one time slot. In Figure 36 the time slot length is one normalized time unit, for illustration purposes, whereas in the actual implementation, the typical time slot length is 2.3 us. Clearly, by adding more zeros, the total time duration between two consecutive pulses is increased and more symbols can be represented. The data packet has to start with a time reference symbol whose value is known by the receiver, so that the rest of the pulses can be decoded with respect to it. In this project, this value was chosen to be 15, corresponding to "10000000000000001".

6.7 Manchester transmitter

The USRP transmitter interface (Figure 37) design involves the modification of a pre-existing virtual instrument made by M.Sc. Pulkkinen, in order to transmit specific commands to the IC. The data packet which is being sent consists of a 16-bit header, a 14-bit address and a 16-bit data field. All the header bits are set to the complement of the address' most significant bit (MSB). The address is used to identify the receiving IC and thus needs to correspond to the device id that is programmed on chip. Due to an unforeseen chip design issue, it was found during the testing phase that the data cannot be properly received, unless each bit is doubled. Thus, only 8 bits of effective data can be transmitted without errors.



USRP2901 or N210: N210
N210 IP: 10.150.150.6
Timeout: 10
IQ Rate: 1.6M
Coerced IQ Rate: 1.5873M
Carrier Frequency: 434M
Coerced Carrier Frequency: 434M
Active Antenna: TX1
Enabled Channel: 0
Gain: 0
Coerced Gain: 0
0.5*T_sym (samples): 20
Command to send: CMD1
error out: status ✓, code 0, source

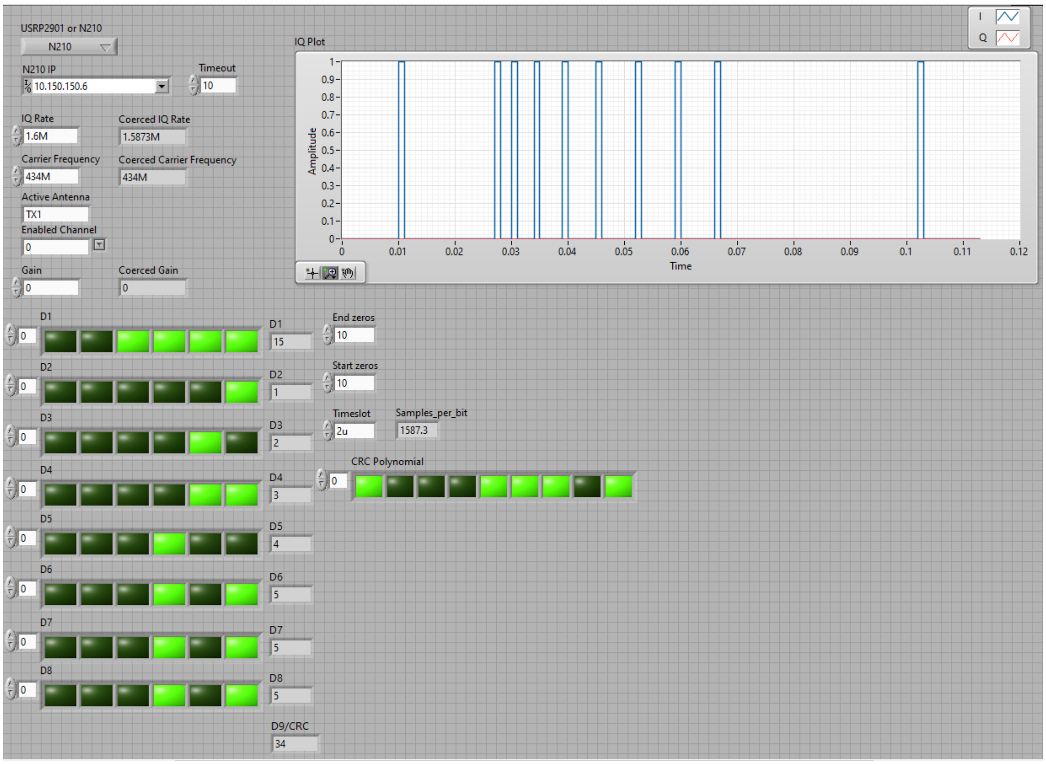
Figure 37: Manchester transmitter front panel.

For ensuring that the data format is appropriate, a list of six predefined commands was implemented, where each command represents a specific action that the chip will take, resulting in a reply from the IC. The commands represent the following binary strings: "11", "1100", "1111", "110000", "110011", "111100", which correspond to the well-known sequence "1", "10", "11", 100", "101", "110" where each bit has been doubled.

6.8 DPPM receiver

6.8.1 Transmitter simulator

With the purpose of designing an USRP receiver interface for the on-chip transmitter, a LabVIEW-based USRP DPPM transmitter simulator is implemented, for development and testing purposes. Nine 6-bit data words (D1-D9 in Figure 38) are transmitted, the first word (the time reference) always being a 15 and the last word representing the 8-bit CRC of the message. The front panel is depicted in Figure 38.



USRP2901 or N210: N210
N210 IP: 10.150.150.6
Timeout: 10
IQ Rate: 1.6M
Coerced IQ Rate: 1.5873M
Carrier Frequency: 434M
Coerced Carrier Frequency: 434M
Active Antenna: TX1
Enabled Channel: 0
Gain: 0
Coerced Gain: 0
D1: 15
D2: 1
D3: 2
D4: 3
D5: 4
D6: 5
D7: 5
D8: 5
D9/CRC: 34
End zeros: 10
Start zeros: 10
Timeslot: 2u
Samples_per_bit: 1587.3
CRC Polynomial

Figure 38: DPPM transmitter simulator front panel.

For a successful transmission, it is crucial that both the DPPM transmitter and receiver agree upon the number of bits per word, the value of the timing reference word and the CRC polynomial.

The block diagram running behind the transmitter front panel is seen in Figure 39 and the DPPM modulator in Figure 40.

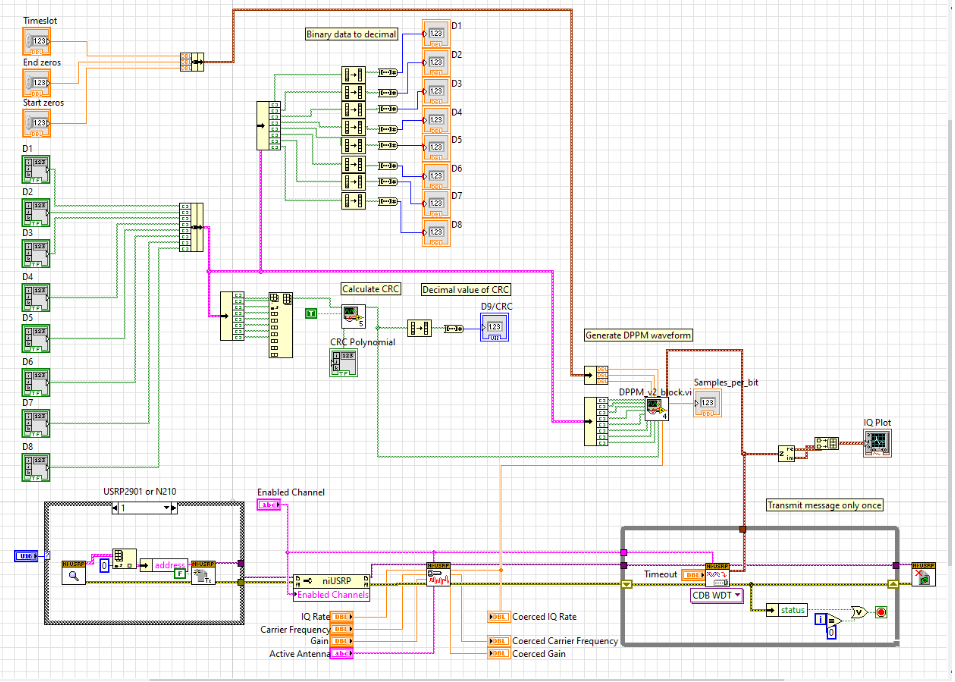


Figure 39: DPPM transmitter block diagram.

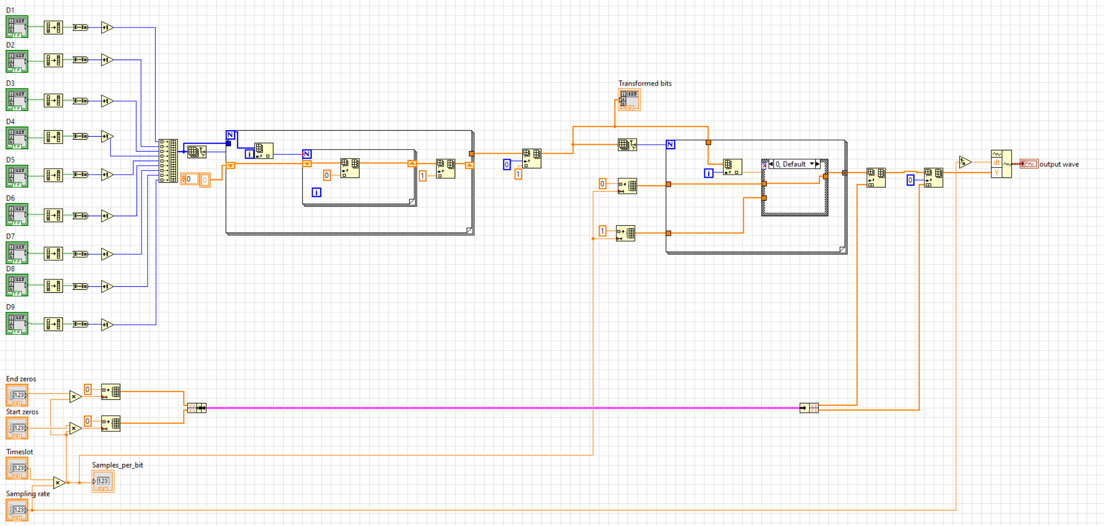


Figure 40: DPPM modulator.

6.8.2 Receiver

The front panel of the DPPM receiver is shown in Figure 41. The receiver is continuously scanning for messages and once the signal level rises with a given threshold above the noise floor, the scanning stops. The absolute values of the demodulated in-phase and quadrature signals (shown on the "IQ Graph" plot in Figure 41) are summed together into a single signal that represents the pulses better. A finite impulse response (FIR) filter is used for implementing a moving average that shapes the pulses in order to present more distinguishable peaks (shown on the "Filtered" plot in Figure 41).

Then, the peaks are extracted and shown in the "Deltas" graph in Figure 41. The peak extraction can be easily explained mathematically. Denote as S the set of ordered pairs

$$S = \{(s_k\ , k)\}, \tag{13}$$

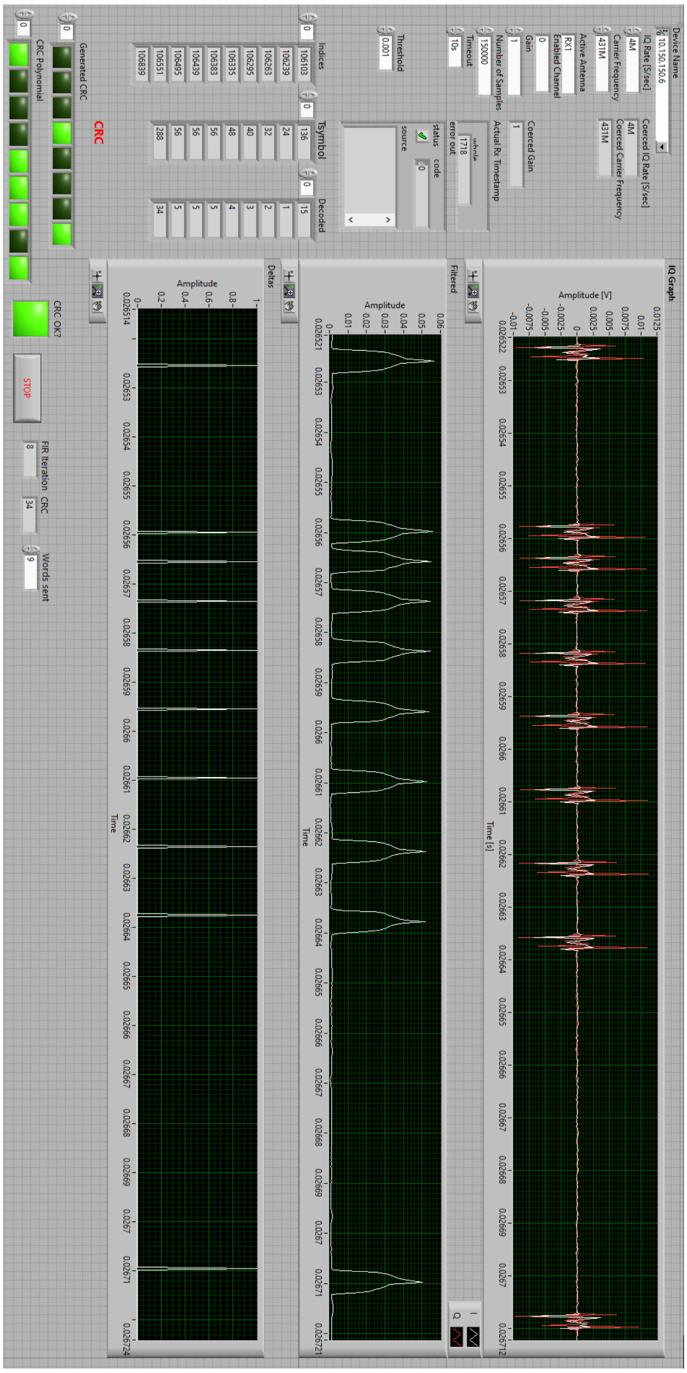
where s_k is the sample's value and k is the sample's index. We can construct a set P which selects only the peak samples p_k from S using the rule

$$P = \{(p_{k+1},\ k+1) \in S \mid p_{k+1} > p_{k+2}\ \mathit{and}\ p_{k+1} > p_k\ \mathit{and}\ p_{k+1} > c\}, \tag{14}$$

where c is an empirical constant used for preventing "false" peaks that arise due to noise, which was found to be 0.05 in this project. As the exact value of the peaks is not of primary interest, a set D (corresponding to "Deltas" in Figure 41) is constructed

$$D(x) = \begin{cases} (1,\ k) & (x,k) \in P \\ (0,\ k) & (x,k) \notin P \end{cases},\ \forall x \in \mathbb{R}\ , k \in \mathbb{N},\ 1 \leq k \leq |S|. \tag{15}$$

Then, the indices of the peak samples are stored (shown on the "Indices" column, bottom left in Figure 41). The indices are then converted to time differences (shown on the "Tsymbol" column in Figure 41), by subtracting the previous index from the current index, for all the peaks. These values correspond to the duration (in samples) of the received symbols. For example, the first value from the "Tsymbol" column is 136 and corresponds to the difference between the second and the first indices from the "Indices" column, namely 106239 and 106103.



Device Name: 10.150.150.6

IQ Rate [S/sec]: 4M

Coerced IQ Rate [S/sec]: 4M

Carrier Frequency: 431M

Coerced Carrier Frequency: 431M

Active Antenna: RX1

Enabled Channel: 0

Gain: 1

Coerced Gain: 1

Number of Samples: 150000

Actual Rx Timestamp: whole 1718

Timeout: 10s

error out: status ✓ code 0 source

Threshold: 0.001

Indices	Tsymbol	Decoded
106103	136	15
106239	24	1
106263	32	2
106295	40	3
106335	48	4
106383	56	5
106439	56	5
106495	56	5
106551	288	34
106839		

FIR Iteration: 8

CRC: 34

Words sent: 9

Figure 41: DPPM receiver front panel.

Figure 6.8.2 shows how the samples are processed by applying the moving average and then the peak detection to the data. In this example, the duration of a time slot is 10 samples and the symbol being represented is "10001". After the peaks have been detected, it is important to notice that 10 samples (one time slot) are missing. Thus, if the symbol comprises 5 time slots, after the peak detection it consists of 4 time slots.

When converting an unprocessed DPPM symbol of T_{symbol} samples to its numerical value x,

$$x = \frac{T_{symbol} - 3T_{timeslot}}{T_{timeslot}} \tag{16}$$

can be used if the $T_{timeslot}$ duration in samples of a timeslot is known. For the example given in Figure 6.8.2, $\frac{50-30}{10} = 2$ corresponds to the code "10001". However, since the peaks need to be used for calculating the T_{symbol}, equation 16 needs to be modified so that

$$x = \frac{T_{symbol} - 2T_{timeslot}}{T_{timeslot}} = \frac{T_{symbol}}{T_{timeslot}} - 2, \tag{17}$$

to account for the loss of one time slot duration. Knowing that the first received symbol (from a series of nine received symbols) always represents a time reference of 15 (17 time slots after peak detection), the duration of one time slot $T_{timeslot}$ is found by dividing the T_{symbol} of the first data message by 17. From Figure 41, $T_{symbol} = \frac{136}{17} = 8$ samples. With this information, all the other Tsymbol values can be decoded: $\frac{24}{8} - 2 = 1$, $\frac{32}{8} - 2 = 2, \frac{40}{8} - 2 = 3, \frac{48}{8} - 2 = 4$, etc.

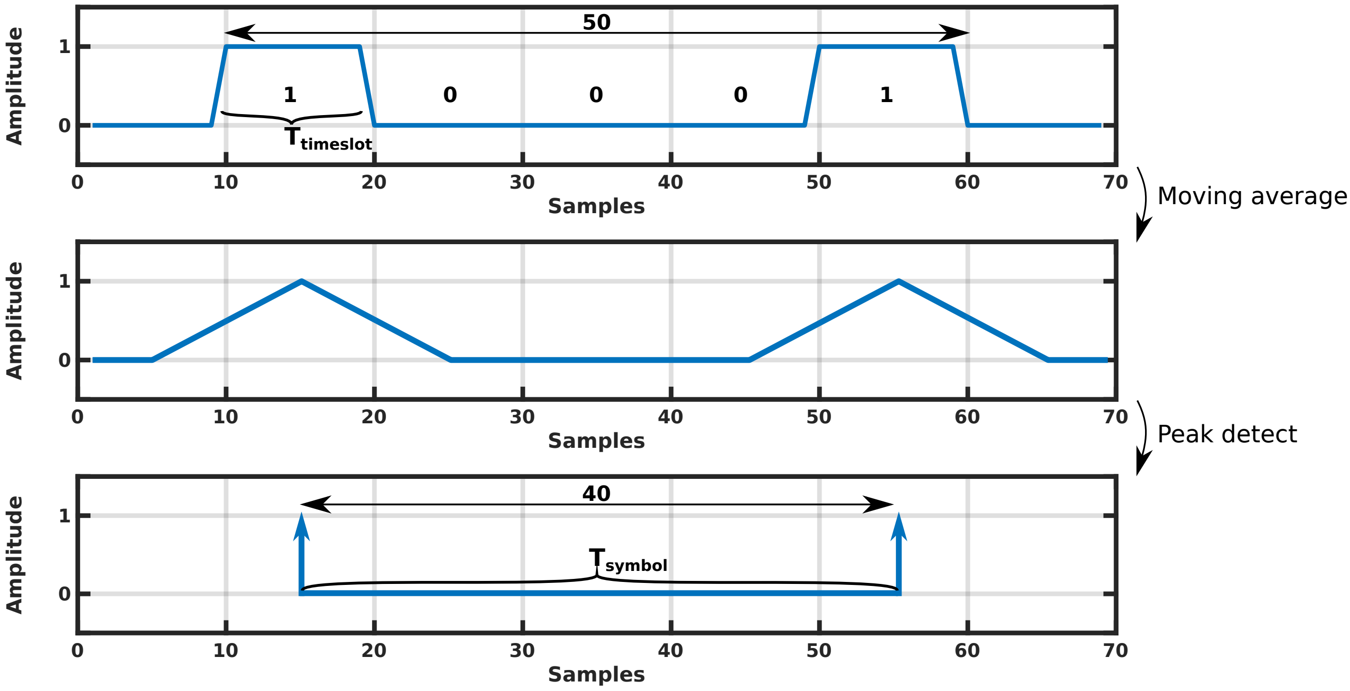


Figure 42: Sample processing.

However, the data can be correctly decoded only if the time reference symbol is received successfully. In order to achieve this, CRC-8 error detection codes are used. The CRC-8 is calculated for the first eight received data words and compared to the last data word, corresponding to the CRC-8 value calculated by the transmitter. If the values do not match, the FIR window is increased and the peak detection process starts again, until the data is correctly decoded, or the packet is declared as erroneous. If the values match, the packet is regarded as correctly received, shown by the "CRC OK?" indicator on the front panel from Figure 41.

7 Python programming interface

In this chapter, the Python programming interface for controlling and configuring the IC is presented. A programming interface is a vital software component for both the testing and characterizing, as well as for prototyping an application involving the IC, as it allows users to automate and configure the operation of the IC. The programming environment, the software architecture and testing methods, together with the IC communication are discussed in this chapter. Section 7.1 gives the motivation behind the choice of programming language. Section 7.2 describes object-oriented software engineering, the principles that were used in this work and their relationship with software qualities. Section 7.3 presents the programming interface: first, the design requirements are stated; second, the communication protocol is introduced, and; third, the interface architecture is presented, along with its main components. Section 7.4 explains the IC communication and the software that was developed for this purpose. Section 7.6 examines the testing and validation methodology. Section 7.7 presents the software documentation process. Section 7.8 concludes the chapter with a summary.

7.1 The Python programming language

Python is a multi-paradigm interpreted programming language, which supports popular paradigms such as object-oriented, structured and functional programming. It is a dynamically typed language, which makes it faster for development, as programmers do not have to explicitly specify types, in contrast to languages that are statically typed (Java, C/C++, etc). Some important core principles that constitute the foundation of the Python programming language are beauty, explicitness, simplicity and readability [33].

Python was chosen to replace the previous programming interfaces that had been written using the proprietary programming languages MATLAB and LabVIEW. The reason for this choice is the language's high popularity [34], free of charge availability, large community and ease of deployment.

7.2 Object-oriented software engineering

Object-oriented programming (OOP) is a programming paradigm that focuses on objects which are instantiations of classes. Classes are software bundles which combine data and data processing methods into unitary modules, that act as "blueprints" for creating objects. Four key concepts are central to object-oriented design: abstraction, encapsulation, inheritance and polymorphism. Abstraction promotes making public only the relevant features of an object, while hiding unnecessary information. Encapsulation means that the data together with the methods that act upon it are wrapped into a single class and direct data access can be restricted outside of the class. Inheritance is the ability of generating subclasses which acquire selected properties and behavior from superclasses. Polymorphism means "many forms", referring to the fact that a subclass can reimplement (or define) a different behavior from its superclass.

7.2.1 Software qualities

The goal of a good object-oriented design is to achieve a high-quality software product. A set of eight qualities were defined by the ISO/IEC 25010 standard, in order to evaluate the quality of a software system: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability [35].

The most important qualities for this project are: functional suitability, as it is crucial for the software to operate correctly; compatibility, as the software has to be usable with other similar ICs and should not depend on a particular IC implementation; usability, as the interface has to be uncomplicated and easy to use by the IC designers; reliability, as the users should immediately be notified if an operation cannot proceed, or did not proceed accordingly, and; maintainability, as the program will be analyzed and updated by other researchers after the completion of this project.

7.2.2 Software design principles

The three design principles which were prioritized in the development of the interface are modularity, generality and single responsiblity. Modularity is typically achieved by high cohesion and low coupling. Cohesion represents the degree of interaction between components within a software module, while coupling represents the degree of interaction between different modules. Generality refers to implementing a solution for a class of problems that includes the problem at hand, in order to facilitate code reuse. The single responsiblity principle states that every module must be responsible for only one functionality implemented by the software and it should encapsulate it completely.

7.3 Programming interface

The programming interface is the central part of a larger software system (Figure 43), that enables prototyping and characterization of the IC. The system consists of the Python interface, the IC communication software running on Arduino Due and the LabVIEW software discussed in Chapter 6, running on the USRP device and on the computer that hosts the Python interface as well.

The programming interface controls the IC and the external devices previously discussed in Chapter 2 via an Arduino Due microcontroller development board. The interface sends commands to the Arduino board, which in turn parses the commands, executes the commands and replies with data.

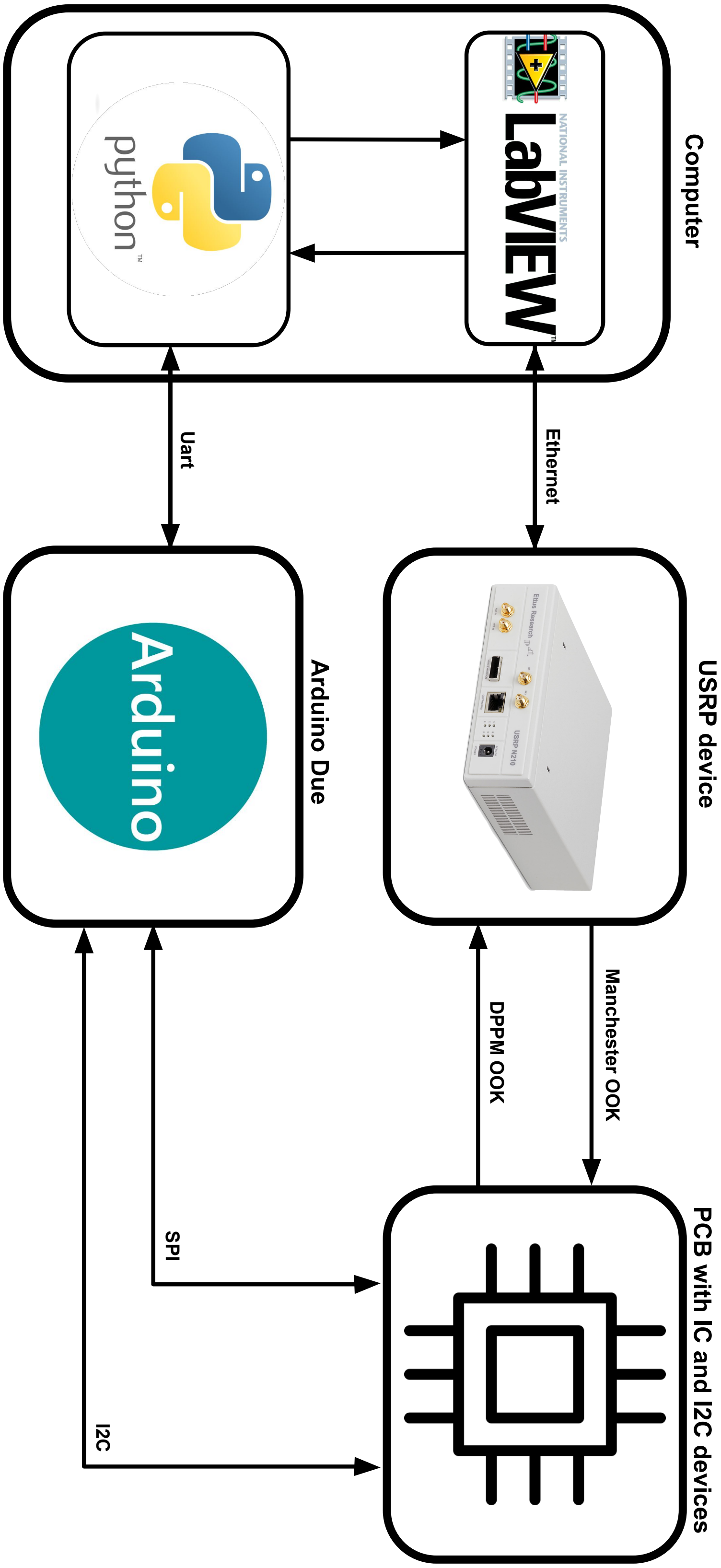


Figure 43: Prototyping and characterization software system.

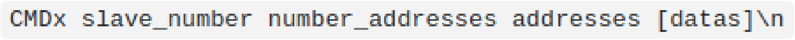
7.3.1 Requirements

Four main requirements were set for the Python programming interface. First, the interface must offer a set of libraries to the IC programmer, in order to facilitate the IC programming and configuration. Second, the software must keep track of the chip's register contents at any time: data that was previously written to the chip is read from the interface, as it takes less time than to read it from the IC. A third requirement is that the interface must be easily adaptable for usage with a different IC and should not depend upon a particular IC implementation. An object-oriented design approach was selected for satisfying the given requirements. In order to fulfill the second requirement, the interface was designed to keep an identical representation of the register data of the IC. The last requirement is that the interface should support commands that address systems which are external to the IC, such as controlling the USI interface sensor emulators or resetting the IC.

7.3.2 Communication protocol

Commands are sent from PC to Arduino via universal asynchronous receiver-transmitter (UART) communication. Two UART devices can transmit and receive data in full duplex, by using a minimum of two data lines: Tx and Rx. A UART packet consists of a start bit, a data frame containing 5 to 8 bits, or 9 bits if no parity is used, an optional parity bit and 1 or 2 stop bits. The data rate is specified by the baudrate parameter, that indicates how many bits per second are transmitted. Thus, the communicating devices need to agree upon the baudrate, parity bit, stop bits and data frame size. Pyserial is used for accessing the serial port of the computer with the purpose of sending and receiving data to Arduino.

A command (Figure 44) can be, for example, a read or a write. In this case, it consists of the command name, SPI slave number, the number of register addresses, the register addresses and optionally the data fields, in case of a write command. Every command must end with a newline control character.



```
CMDx slave_number number_addresses addresses [datas]\n
```

Figure 44: Read/write command syntax.

If the type of command is "read", the Arduino Due board replies with the requested data, if the communication was successful, otherwise it replies with a diagnostic message. If the type of command is "write", the reply consists of an agreed-upon message, indicating a successful or unsuccessful write operation. Other commands refer to the NCD2400M and MAX5419 devices, or externally reset the IC.

7.3.3 Interface architecture

The programming interface (Figure 45) comprises four superclasses: ReadOnlyRegister, Command, CommandWrapper and TuneCommandWrapper. The ReadOnlyRegister and its subclass, Register, mirror the data that is stored within the IC registers. Command is a superclass that enables ReadOnlyRegisterCommand, RegisterCommand and TuneCommand to access and alter the contents of the Register objects. CommandWrapper is the superclass from which ReadOnlyRegisterCommandWrapper, TuneCommandWrapper and RegisterCommandWrapper inherit. These classes act as a binder between the software data representation and the IC, offering the programmer an application programming interface (API) for interacting with the chip. CommandWrapper-type objects have three types of operations that allow the user to write or read data either from the IC or from the internal memory representation. The ReadOnlyRegisterCommandWrapper allows only reading data directly from the IC, as it interfaces with a read-only register. The SlowMemoryTuneCommandWrapper is a special type of TuneCommandWrapper, that allows interfacing with slow memory blocks. The ArduinoCommunication object handles the command communication between the interface and the IC. It implements two operations, for sending and receiving commands to/from the IC.

7.3.3.1 Registers

Registers are classes that store data regarding the on-chip registers. ReadOnlyRegister objects only store the address information, while Register objects store, additionally, data and an interval for the valid data values. Each time a new data value is written to a register, it is checked against the valid interval, otherwise the user is alerted.

7.3.3.2 Register commands

RegisterCommands are objects used for manipulating Register data at software interface level. These commands have the following attributes: information, that best describes the register command; SPI slave number, that identifies and validates the SPI address of the IC block that has to be accessed; a Register object, upon which the command acts; data to be written to the software register. In addition, a method for reading the data stored in the Register and a method for writing the command data to the Register are provided. The command data should be written to the software Register only if the same data was successfully written to a physical register inside the IC. ReadOnlyRegisterCommands are used for encapsulating ReadOnlyRegisters and do not provide any means of altering data.

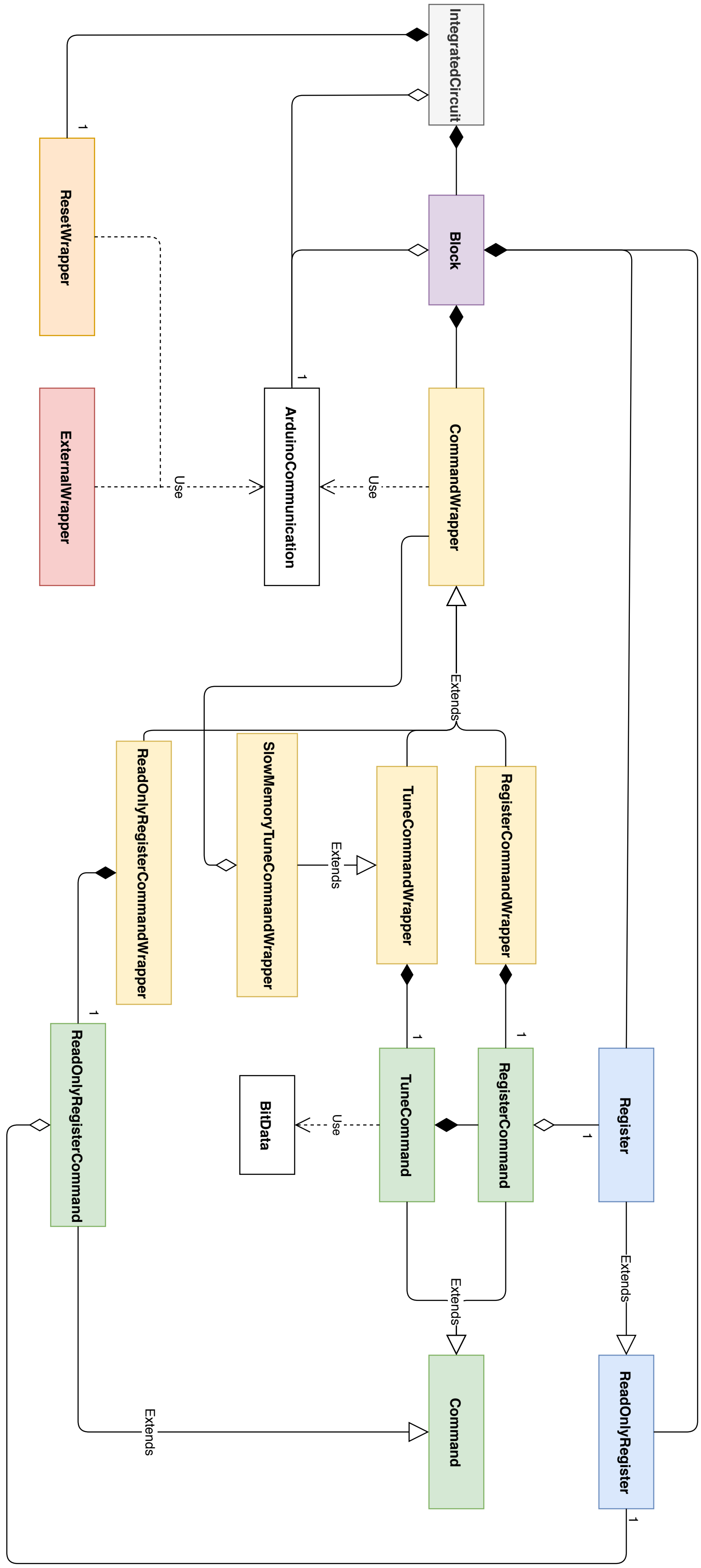


Figure 45: Python programming interface class diagram.

7.3.3.3 Tune commands

TuneCommands are used for accessing multiple bits of one or more registers as a single data value. For example (Figure 46), a 5-bit parameter value might be stored as the sixth and fourth bits of one register "x" and the sixth, fourth and first bits of another register "y". For achieving this purpose without any complications, these commands create a list of RegisterCommands. Compared with the RegisterCommands, TuneCommands are initialized with a list of registers instead of a single register attribute and a list of lists of positions (one list for each register), that define the bits which make up the data of interest.

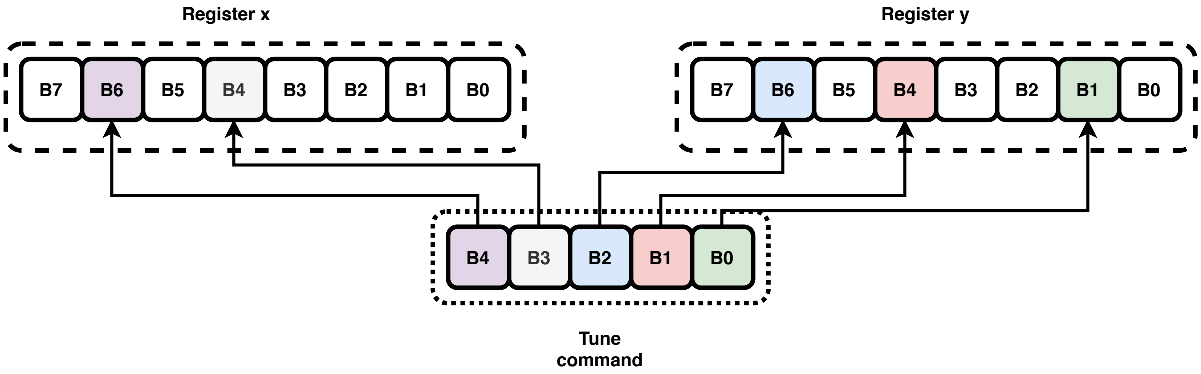


Figure 46: Tune command example.

7.3.3.4 Command wrappers

CommandWrapper objects are wrapping Command objects and are intended to be used by the programmer who wishes to control the IC. These commands share an ArduinoCommunication object and must ensure that the data was written properly to the IC before updating the Register objects. CommandsWrappers store an interval that specifies the valid range of the data to be written. They are also responsible with informing the user regarding the status of the executed command.

7.3.3.5 Slow memory

Some of the registers of the IC cannot be accessed directly. These registers constitute a "slow memory", due to the fact that they are accessed in an indirect, more complicated (and time-consuming) fashion. First, the data that is to be written to the target register has to be written to a special data register. Then, the address of the target register has to be written to another special address register. Finally, a bit that triggers the write process has to be set and cleared. For reading the data from a slow memory register, only two special registers are needed: one that stores the address which is to be read and another one that retrieves the requested data. A class called SlowMemoryTuneCommandWrapper was developed, for aiding the developers in programming the IC. Slow memory data can be read and written using the same methods that the other wrappers provide.

7.3.3.6 Blocks

A Block is an abstract class, which is to be implemented by each IC block defined by the IC programmer. Its attributes are a shared ArduinoCommunication object which is common to all the other blocks of the IC and an information string that describes the block. It should have multiple Register and CommandWrapper objects that ressemble the architecture of the IC block and the operations that are to be performed on its registers. Writing the Block classes which mirror the blocks of the current IC took a considerable amount of work, accounting for one third of the Python code written for the interface.

7.3.3.7 Integrated circuits

More than one integrated circuit can be defined and used in the same program, with the IntegratedCircuit class. Assuming that each IC has its own ArduinoDue connected to the PC, the IC programmer can automate the configuration and operation of multiple ICs with the same piece of software. The IntegratedCircuit class consists of programmer-defined blocks, a ResetWrapper and ExternalWrappers. The ResetWrapper is used for externally resetting the IC, while the ExternalWrapper enables communication with I2C devices such as the ones presented in 2.

7.4 IC Communication

Arduino Due establishes the communication between the PC and the IC, by forwarding incoming commands from the PC to the IC. On the IC side, the communication protocol is the Serial Peripheral Interface (SPI), which is used for accessing individual registers. The SPI is a synchronous serial communication interface, composed of four bus lines: Serial Clock (SCLK), Master Output Slave Input (MOSI), Master Input Slave Output (MISO) and Slave Select (SS).

SPI communication is established by the master selecting the slave through the SS line (typically active low). A clock signal is provided by the master to the slave via the SCK line. Then, data is transferred one bit per clock cycle, from the slave to the master, via the MISO line and concurrently from the master to the slave via the MOSI line.

Depending on the clock polarity (CPOL) and phase (CPHA), four different SPI modes (0-3) can be distinguished. If the clock idle state is LOW, then CPOL equals 0, otherwise it is 1. CPHA determines the timing of the data bits with respect to the clock signal. If CPHA is 0, the first half of the clock cycle is idle and the second half of the clock cycle is asserted. When CPHA is 1, the contrary holds valid.

The description of the IC's SPI interface is published in [12] and is presented in Figure 47. The CPOL is 0 and CPHA is 0, therefore the SPI mode is 0. An SPI command consists of 16 bits. The first 6 bits constitute the address of the register to be accessed. The next bit represents the read or write quality of the command. An odd parity bit follows, which checks the validity of the address and read/write bits. The last 8 bits correspond to the data that is written to the selected register. While receiving the current command, the slave replies with 16 bits regarding the previous command. The first bit is a dummy zero, followed by two information bits. The first information bit (I1) flags an error in the previous command, while the second bit (I0) flags a supply voltage outage. The next 4 bits are a dummy sequence "0001", followed by an odd parity bit for the data being sent. Finally, 8 data bits are transmitted.

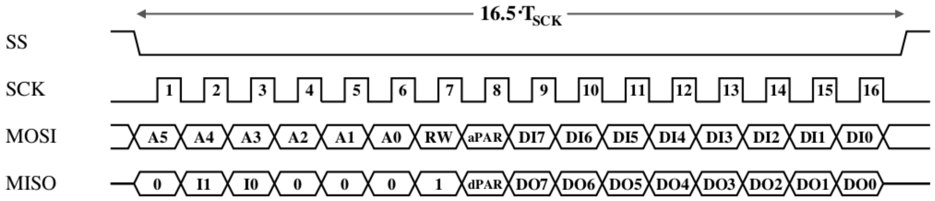

16.5·T_{SCK}
SS
SCK
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
MOSI
A5 A4 A3 A2 A1 A0 RW aPAR DI7 DI6 DI5 DI4 DI3 DI2 DI1 DI0
MISO
0 I1 I0 0 0 0 1 dPAR DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0

Figure 47: SPI timing diagram. From [12].

Seven SPI slaves are accessible within the IC. In order to differentiate between them, three additional signal lines are available for interfacing the chip communication. These lines consist of the select signals of a demultiplexer, that routes the slave select signal to the appropriate SPI slave within the IC. Therefore, in addition to the four SPI lines, the Arduino Due board uses three slave select signals, totalling seven communication lines. A custom FC-6P to FC-10P connector was created that facilitates plug and play operation, reducing the time for wiring and debugging.

7.4.1 Command parsing and IC-communication software

An Arduino Due board was chosen to handle the IC communication and command parsing. The board functions at a supply voltage of 3.3 V, making its input and output voltage levels compatible with the characterization PCB without needing additional level translation circuitry. The board is based on an Atmel SAM3X8E ARM Cortex-M3, which is a powerful 32-bit microcontroller with 512 kBytes of Flash and 100 kBytes of SRAM, operating at a maximum speed of 84 MHz. In later development stages, the board should run a voltage regulation control loop, besides handling commands, motivating the need for such a powerful microcontroller.

The communication and parsing software is written in C++. It consists of three core classes, two libraries used for sensor emulation (described in Chapter 2) and one main program. CommandParser is the class that is responsible for receiving and handling serial commands. NakuSpi is the class which handles the SPI communication. An ICReset class handles the resetting of the IC. Figure 48 presents the high-level operation of the program. In advance of handling the incoming commands, the serial and I2C communication is initialized, objects are instantiated and the I0 information bits are cleared for each SPI slave. Then, the command parsing continues unless the Arduino Due is reset.

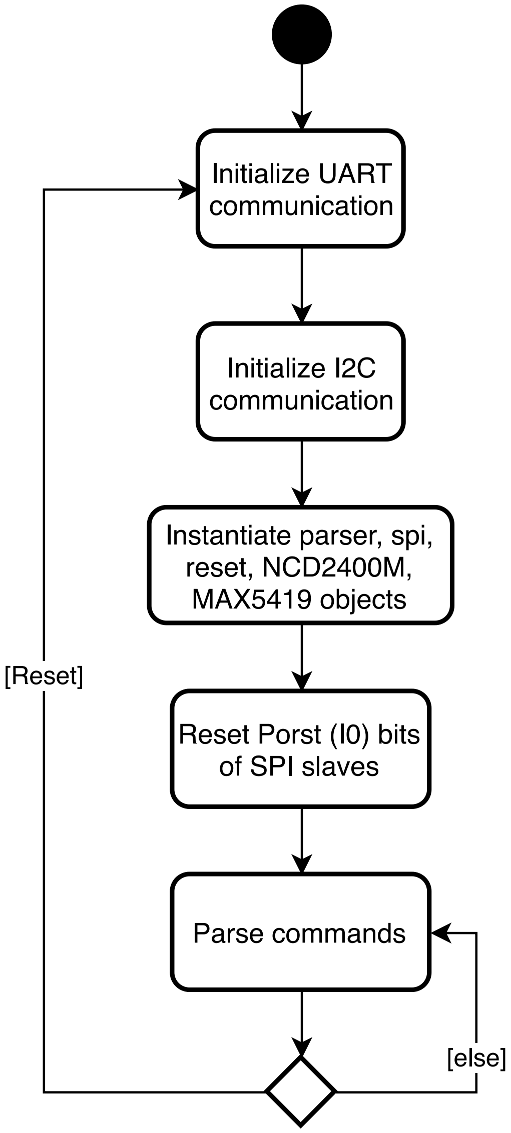

Initialize UART communication
Initialize I2C communication
Instantiate parser, spi, reset, NCD2400M, MAX5419 objects
Reset Porst (I0) bits of SPI slaves
Parse commands
[Reset]
[else]

Figure 48: Main program activity diagram.

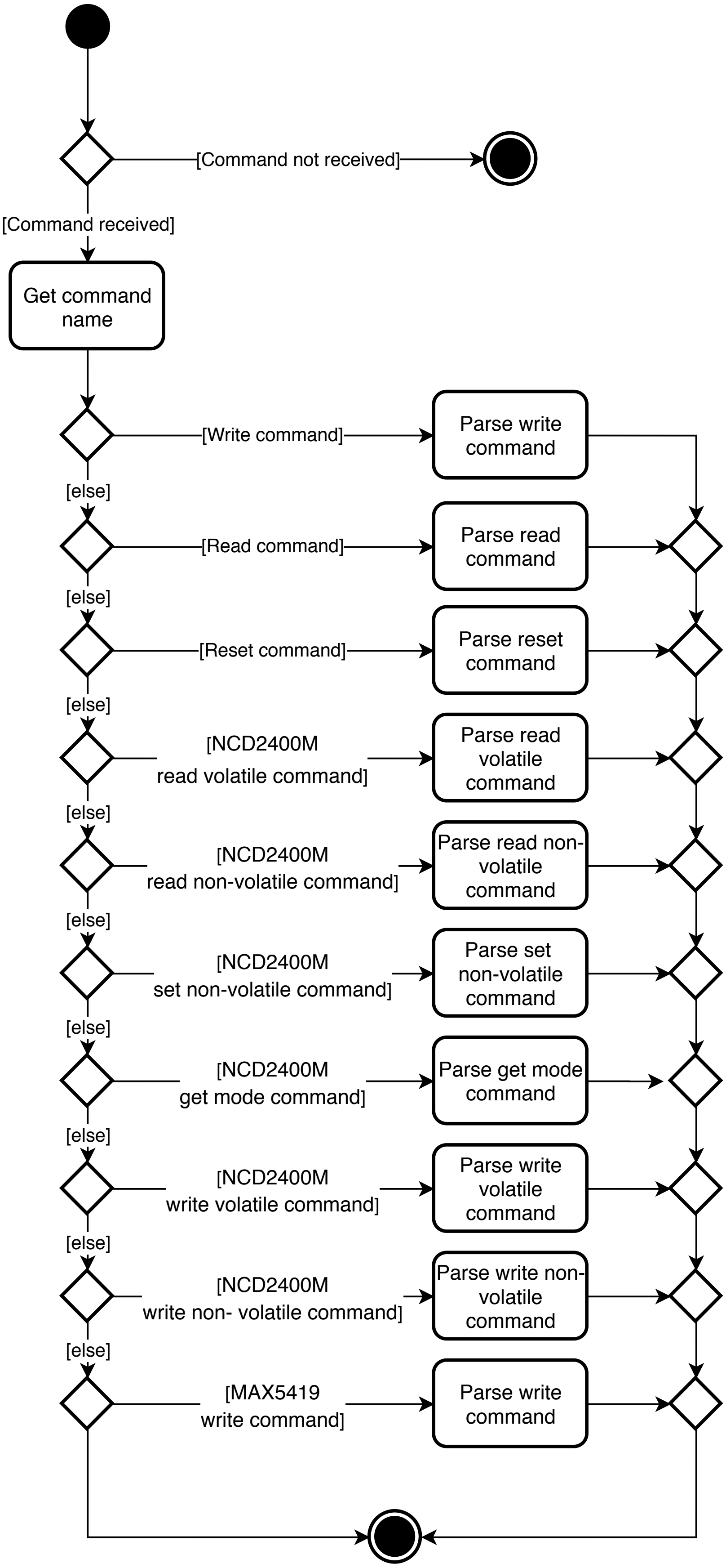

[Command not received]
[Command received]
Get command name
[Write command]
Parse write command
[else]
[Read command]
Parse read command
[else]
[Reset command]
Parse reset command
[else]
[NCD2400M read volatile command]
Parse read volatile command
[else]
[NCD2400M read non-volatile command]
Parse read non-volatile command
[else]
[NCD2400M set non-volatile command]
Parse set non-volatile command
[else]
[NCD2400M get mode command]
Parse get mode command
[else]
[NCD2400M write volatile command]
Parse write volatile command
[else]
[NCD2400M write non- volatile command]
Parse write non-volatile command
[else]
[MAX5419 write command]
Parse write command

Figure 49: Command parser activity diagram.

The CommandParser is presented in Figure 49. Once an incoming command is received, its name is extracted. Depending on the command's name, the CommandParser invokes a suitable method that performs the requested actions. Command names have to be agreed upon by both the Python interface and the Command Parser. In order to facilitate consistency, commands are identified by identical constants, which are defined in both interfaces.

The modular command parsing architecture ensures future maintainability, as the parser can be easily extended with new commands and old commands can be modified without affecting behaviors that are external to them. In further development iterations, the commands could be encapsulated by dedicated classes.

Figure 50 presents how a write command is parsed. First, the SPI slave number, number of addresses that will be written to, the list of addresses and the corresponding list of data integers to be written to each address are parsed from the command. Then, the SPI communication begins by selecting the SPI slave and writing the data for each SPI address. An error status is collected after each write operation, for debugging purposes. Once all the data was written, the program continues with a check. For each address in the list, the data is read and compared to the target data. In case of a mismatch, the process stops by sending a message that flags the insuccess of the writing operation. When all data was successfully read, the program sends a message which informs the interface that the write operation could be performed. Both messages are agreed upon by the two pieces of software, using constants.

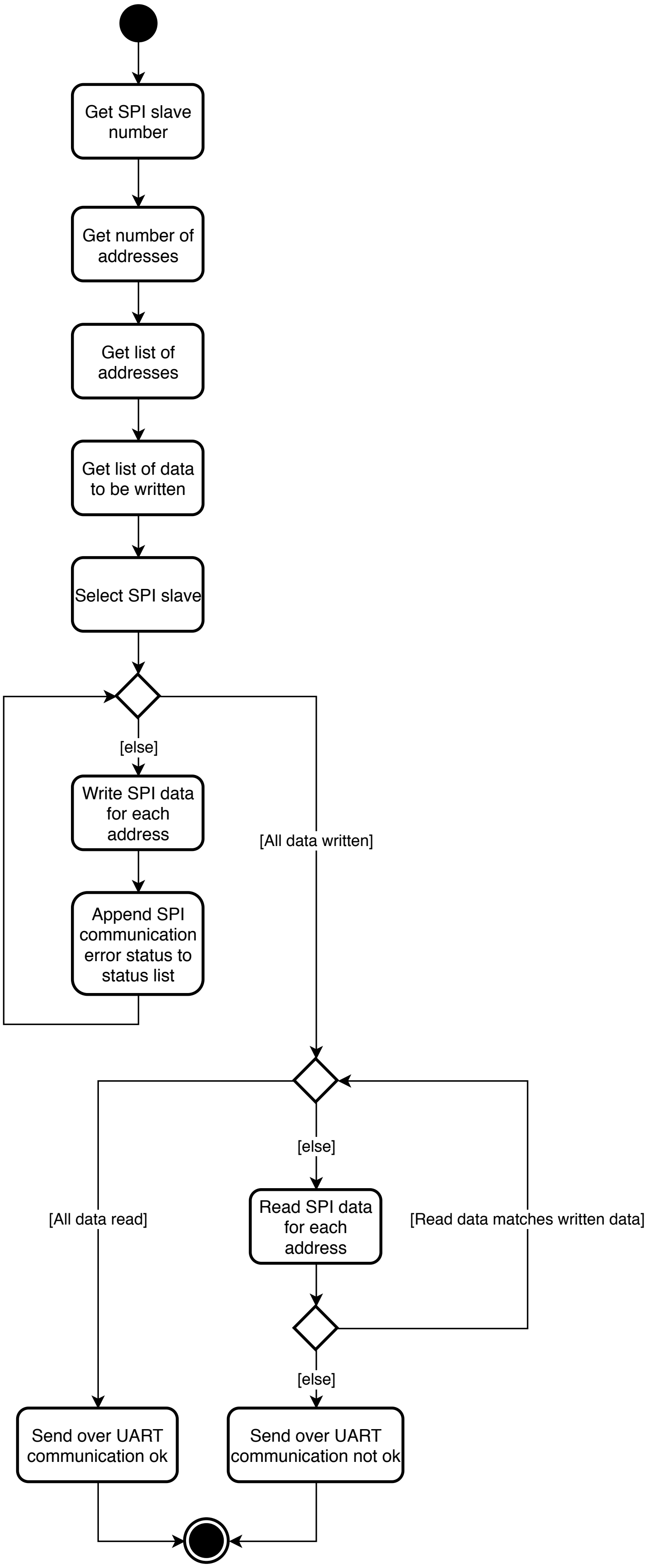

Get SPI slave number
Get number of addresses
Get list of addresses
Get list of data to be written
Select SPI slave
[else]
Write SPI data for each address
[All data written]
Append SPI communication error status to status list
[else]
[All data read]
Read SPI data for each address
[Read data matches written data]
[else]
Send over UART communication ok
Send over UART communication not ok

Figure 50: Write command activity diagram.

7.5 Project structure

When working with a medium-sized project such as this, as the software is consisting of multiple components and files, it is important to maintain an organized folder hierarchy that emphasizes the project structure. There are two main components of the interface: the IC communication software and the Python programming interface software, each having its own file hierarchy.

```
Python
│   README.md
│   .gitignore
│   ic.py
│
└───blocks
│   │   __init__.py
│   │   block.py
│   │   beta.py
│   │   disp.py
│   │   eh.py
│   │   gs.py
│   │   napro5.py
│   │   nbtx.py
│   │   regu.py
│   │   usi.py
│   │   uwb.py
│
└───comm
│   │   __init__.py
│   │   arduinoCommunication.py
│
└───commands
│   │   __init__.py
│   │   bitdata.py
│   │   command.py
│   │   readOnlyRegisterCommand.py
│   │   registerCommand.py
│   │   tuneCommand.py
│
└───constants
│   │ __init__.py
│   │ Constants.py
│   │ External.py
│
└───demos
│
└───env
│
└───registers
│   │   __init__.py
│   │   readOnlyRegister.py
│   │   register.py
│
└───tests
│   │   test_bitData.py
│   │   test_Command.py
│   │   test_ic.py
│   │   test_ReadOnlyRegister.py
│   │   test_Register.py
│   │   test_RegisterCommand.py
│   │   test_RegisterCommandWrapper.py
│   │   test_TuneCommand.py
│   │   test_TuneCommandWrapper.py
│
└───wrappers
    │   __init__.py
    │   commandWrapper.py
    │   externalWrapper.py
    │   readOnlyRegisterCommandWrapper.py
    │   registerCommandWrapper.py
    │   resetWrapper.py
    │   slowMemoryTuneCommandWrapper.py
    │   tuneCommandWrapper.py
```

Figure 51: Python interface structure.

```
command_parser
│   README.md
│   command_parser.ino
└───libraries
    |   Readme.txt
    |
    └───Constants
    |   | constants.h
    |
    └───ICReset
    |   | ICReset.cpp
    |   | ICReset.h
    |
    └───MAX5419
    |   | MAX5419.cpp
    |   | MAX5419.h
    |
    └───ncd2400m
    |   | ncd2400m.cpp
    |   | ncd2400m.h
    └───Parser
    |
    |   | commandParser.h
    |   | commandParser.cpp
    └───SPIcomm
        | nakuSPI.h
        | nakuSPI.cpp
```

Figure 52: IC Communication software structure.

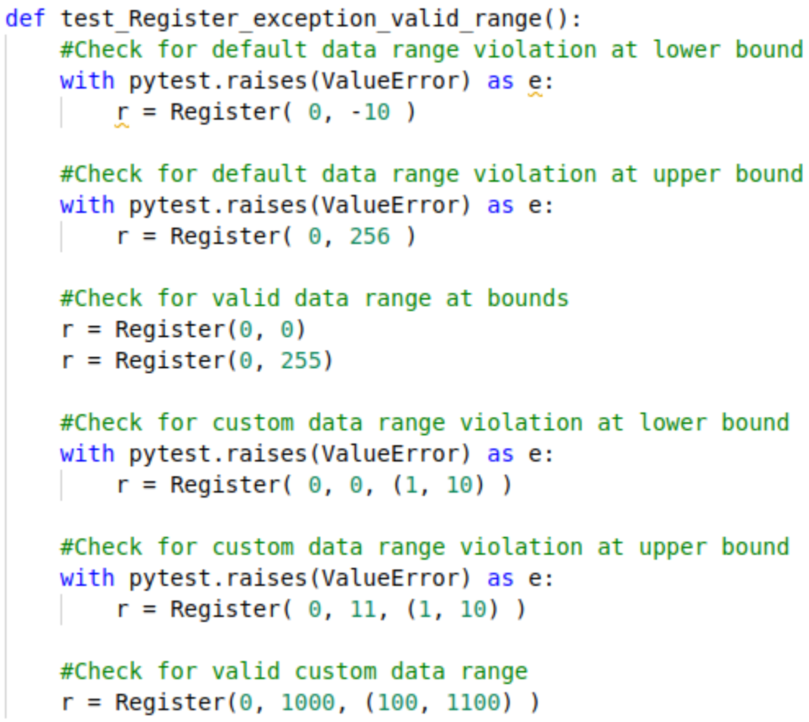
7.6 Testing and validation

7.6.1 Testing

Software testing is a crucial part of the software development life cycle. Testing assures that the software operates correctly and meets the necessary requirements, enabling the developers to catch bugs at early stages of product development. Test-driven development (TDD) is a common software engineering practice, where requirements are formulated as test cases that have to be passed by the program. Unit tests are written for each function of each module, testing both normal and abnormal execution scenarios.

As the software project becomes more complex, it is harder for the developer to keep track of all the changes and ensure that adding a new feature (or modifying a pre-existing one) does not break the functionality of the software system as a whole. Integration testing helps to avoid such situations by having a set of tests which use multiple modules together, in order to evaluate the system as a whole.

Pytest was used for testing the interface. Pytest is a framework that enables test-driven development with Python. Tests are written independent of the source code, allowing an organized directory structure for the project. By executing a simple command, developers can launch a series of complex tests. Pytest can also be integrated into code editors such as Visual Studio Code, which was used for developing this project. In case of a failure, the framework provides extensive information that helps pinpointing the software fault. Figure 53 presents a unit test for the validation of the register data.

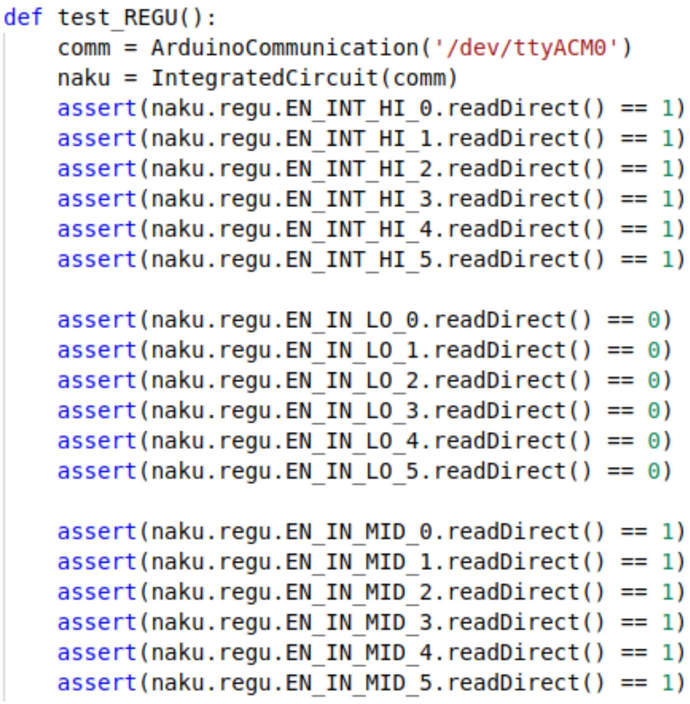


```
def test_Register_exception_valid_range():
    #Check for default data range violation at lower bound
    with pytest.raises(ValueError) as e:
        r = Register( 0, -10 )

    #Check for default data range violation at upper bound
    with pytest.raises(ValueError) as e:
        r = Register( 0, 256 )

    #Check for valid data range at bounds
    r = Register(0, 0)
    r = Register(0, 255)

    #Check for custom data range violation at lower bound
    with pytest.raises(ValueError) as e:
        r = Register( 0, 0, (1, 10) )

    #Check for custom data range violation at upper bound
    with pytest.raises(ValueError) as e:
        r = Register( 0, 11, (1, 10) )

    #Check for valid custom data range
    r = Register(0, 1000, (100, 1100) )
```

Figure 53: Unit test written in Pytest.

7.6.2 Validation

The software validation takes place as the last step of the testing process, aiming to ensure that the code meets its specifications before it is released. For this thesis, the validation consisted of comparing the Python interface's operation to the LabVIEW interface which had been made and utilized previously by the research team. Various commands were executed and checked with both interfaces for consistency. Figure 54 shows a validation test for the IC. A dedicated configuration file was written in LabVIEW and the configuration data was uploaded to the IC. Then, a validation test file was made using the Python interface, that checked that all the data was being read correctly.



```
def test_REGU():
    comm = ArduinoCommunication('/dev/ttyACM0')
    naku = IntegratedCircuit(comm)
    assert(naku.regu.EN_INT_HI_0.readDirect() == 1)
    assert(naku.regu.EN_INT_HI_1.readDirect() == 1)
    assert(naku.regu.EN_INT_HI_2.readDirect() == 1)
    assert(naku.regu.EN_INT_HI_3.readDirect() == 1)
    assert(naku.regu.EN_INT_HI_4.readDirect() == 1)
    assert(naku.regu.EN_INT_HI_5.readDirect() == 1)

    assert(naku.regu.EN_IN_LO_0.readDirect() == 0)
    assert(naku.regu.EN_IN_LO_1.readDirect() == 0)
    assert(naku.regu.EN_IN_LO_2.readDirect() == 0)
    assert(naku.regu.EN_IN_LO_3.readDirect() == 0)
    assert(naku.regu.EN_IN_LO_4.readDirect() == 0)
    assert(naku.regu.EN_IN_LO_5.readDirect() == 0)

    assert(naku.regu.EN_IN_MID_0.readDirect() == 1)
    assert(naku.regu.EN_IN_MID_1.readDirect() == 1)
    assert(naku.regu.EN_IN_MID_2.readDirect() == 1)
    assert(naku.regu.EN_IN_MID_3.readDirect() == 1)
    assert(naku.regu.EN_IN_MID_4.readDirect() == 1)
    assert(naku.regu.EN_IN_MID_5.readDirect() == 1)
```

Figure 54: Validation test written in Pytest.

7.7 Documentation

Documenting the software is an important part of the development life cycle which is often overlooked. Documentation consists of any artifact that has an explanatory role and can typically regard the following: requirements, architecture design, technical aspects and user instructions. The requirements document describes the desired software operation, user characteristics, constraints, assumptions and dependencies. The architecture design document supports the technical document at a higher level, specifying how the design shall be achieved in broader terms. Architecture design documents are typically used in large projects with multiple developers. Technical documentation consists of source code documentation, that can either accompany the software as separate files, or can be extracted directly from comments. User documentation explains how the software shall be used by the intended users.

This project's documentation focused on technical- and user-centered aspects. It is far-reaching for two reasons: the software will be maintained in the research group after this project has finished; the researchers, acting as end-users, will utilize the code for creating their own IC programming interfaces.

Python provides docstrings for the technical documentation. Docstrings are string literals that can be used in classes, modules or functions and can be accessed as attributes of the documented objects, or by calling the built-in help() function. Moreover, certain code editors can show the docstring of a piece of code that is hovered over.

End user documentation was written for aiding the researchers in setting up the interface, as well as building a custom interface for their specific needs. Users are instructed how to set up the hardware connections. A comprehensive explanation of the project structure is given. A step-by-step how-to that explains the creation of a bespoke IC interface is provided.

The Git version-control system was used for keeping track of the software components and their changes.

7.8 Summary

This chapter presented the development of the programming interface which is used by researchers to configure and program their ICs. The whole software design process was discussed, taking into consideration aspects such as the programming languages, requirements, qualities, program architecture, project structure, testing, validation and documentation. The interface can be customized for different ICs, is easily expandable with other features and can be used for controlling and configuring multiple ICs at the same time. Table 2 summarizes the software metrics which describe the interface. In addition to the data presented in Table 2, two Markdown Readme files of 157 and 97 lines of code (LOC) were written for the Python Interface and IC communication software, respectively.

Table 2: Programming interface software metrics.

Metric	**Python code**	**C++ code**
Files	58	12
Code lines	3388	606
Comment lines	319	103
Blank lines	873	88
Total lines	4580	797

The interface is deployed in Chapter 9, where the testing and measurement of a one-bit display driver are performed.

8 Measurement automation

This chapter discusses measurement automation: the process of automating electrical measurements by means of controlling laboratory equipment with computer software, instead of performing manual work. Testing and characterizing an IC requires elaborate measurements, involving large amounts of data and repetitive procedures. In Section 8.1 communication methods with measurement equipment are considered. Section 8.2 gives the motivation behind the choice of programming environment. The Virtual Instrument Software Architecture (VISA) is introduced in Section 8.3. In Section 8.4 Standard Commands for Programmable Instruments (SCPI) are presented. Automation commands for Wave Expert series oscilloscopes are discussed in Section 8.5. The instrument drivers written for this thesis are included in Section 8.6. Finally, Section 8.7 unveils the complete remote measurement setup.

8.1 Communication with measurement equipment

The communication between a computer and electronic test equipment can be achieved in several ways. A General Purpose Interface Bus (GPIB), also known as IEEE-488 is the oldest and most common interface, designed especially for controlling measurement instruments. The Universal Serial Bus (USB) is a newer standard which is commonly used for connecting peripherals to computers. It is found in many modern measurement devices, however, due to the noise sensitivity of the cables, it is not always the most suitable choice. Local area network (LAN) eXtensions for Instrumentation (LXI) is a standard that is used for measurement devices with Ethernet capabilities. The high connection bandwidth, standardized port supported by any computer and the fact that the instrument can be accessed remotely by any machine operating inside the same LAN makes LXI a good choice for this project. Instruments can be connected to a switch and then automation commands can be sent from anywhere in the LAN. If remote connection to any computer in the same LAN as the measurement equipment is possible, then measurements can be performed remotely.

8.2 Programming environment

During the past, MATLAB and LabVIEW have been used for measurement automation by the research department. However, now the department is shifting the programming environment towards Python. Therefore, Python 3 was used as the programming language for the measurement automation software.

Many instruments come with readily-available automation drivers. While this eases the work for MATLAB or LabVIEW developers, it is often not the case with Python. Automation software therefore had to be written for the measurement equipment by consulting the remote control/automation manual of each instrument. Aside from this inconvenience, using Python for establishing the measurement setup allows having an unitary programming and testing interface. It is more convenient to control both the IC and the measurement instruments using one program that imports the required APIs as modules.

8.3 Virtual instrument software architecture

VISA is a standard communication API used in the Test and Measurement (T&M) industry. Its primary use is for communicating with various instruments from a computer. VISA supports communication over multiple interfaces, such as the GPIB, VXI USB and Ethernet, offering programmers a standard way of communicating with T&M devices. VISA is sometimes called a communication driver, which is not to be confused with an instrument driver.

For this project, the PyVISA library was used, as it implements VISA API as a Python package. PyVISA is easy to utilize: once a resource (measurement device) is available, the programmer can connect to it and start sending commands. However, for making the resource available to the PC used for remote controling the device, the Measurement & Automation Explorer by NI is needed.

8.4 Standard commands for programmable instruments

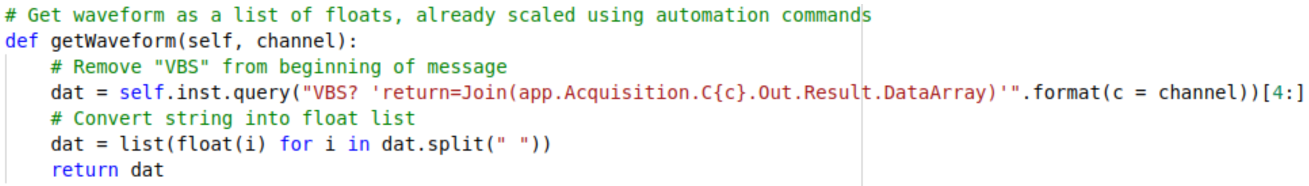
SCPI is a standard that defines commands for controlling measurement devices. The standard (available in [36]) comprises of four volumes: "Syntax and style", "Command Reference", "Data Interchange Format" and "Instrument Classes". SCPI commands can perform two operations, called a set or a query. A set operation changes some state of the instrument (for example, enables outputs), while a query operation returns a value (for example, a current reading). There are two types of command implemented by SCPI: common and subsystem. Common commands are defined in the IEEE 488.2 standard and are used to perform functions that are independent of the instrument, such as identify, status and reset. Subsystem commands perform functions that are specific to the instrument and are defined in the second volume of the SCPI standard.

8.5 Automation commands

Not all measurement devices necessarily use SCPI commands for remote control. Teledyne LeCroy define their own automation protocol (defined in [37]) for remote controlling their X-Stream digital storage oscilloscopes (DSOs). The commands used for controlling the Wavesurfer 44xs oscilloscope which was used for this thesis can be split into two types: remote control and automation commands. The remote control commands are used for achieving simple functions regarding the oscilloscope's state offering the possibility to adjust parameters of subsystems such as status, display, acquisition, as well as for sending automation commands. Automation is a separate subsystem, that performs complex functionalities, having a dedicated programmer's manual. Automation commands are written in the Visual Basic programming language, as the X-Stream instruments implement Microsoft's Component Object Model (COM) interface.

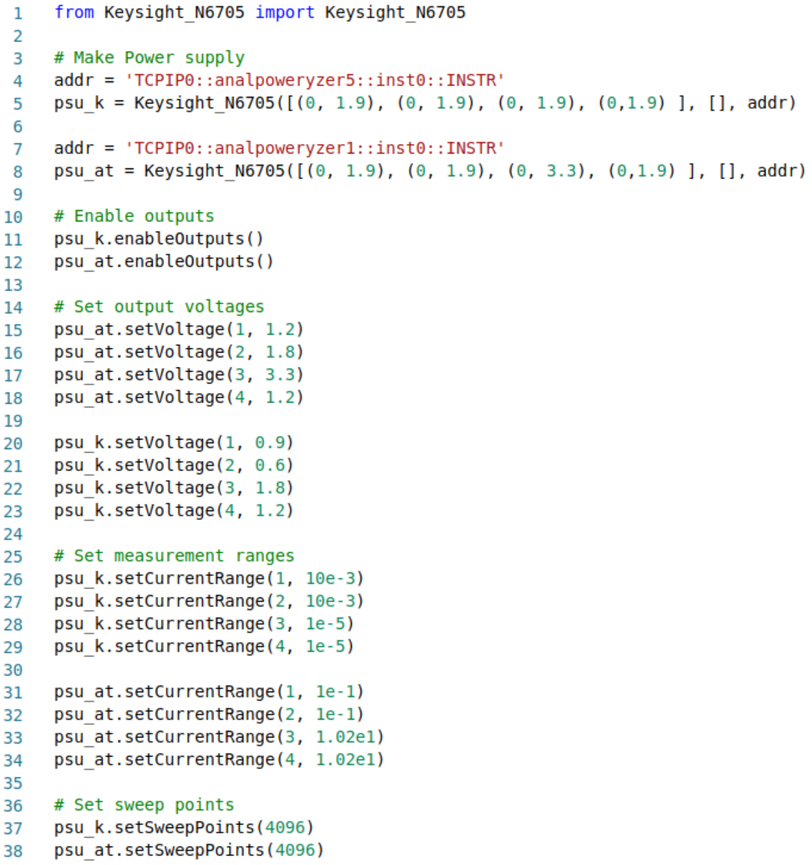
8.6 Instrument drivers

Instrument drivers are a set of software modules which enable the programmers to easily acquire data and control a specific test instrument. During this thesis work, drivers were written for Keysight N6705, Keysight B2961A and LeCroy Wavesurfer 44Xs-A instruments. A typical driver consists of a class that defines the instrument object, which takes a PyVISA resource as a constructor argument. Figure 55 presents a function that is acquiring a waveform from the LeCroy oscilloscope. Figure 56 shows a power supply initialization script.



```python
# Get waveform as a list of floats, already scaled using automation commands
def getWaveform(self, channel):
    # Remove "VBS" from beginning of message
    dat = self.inst.query("VBS? 'return=Join(app.Acquisition.C{c}.Out.Result.DataArray)'".format(c = channel))[4:]
    # Convert string into float list
    dat = list(float(i) for i in dat.split(" "))
    return dat
```

Figure 55: Waveform acquisition.



```python
from Keysight_N6705 import Keysight_N6705

# Make Power supply
addr = 'TCPIP0::analpoweryzer5::inst0::INSTR'
psu_k = Keysight_N6705([(0, 1.9), (0, 1.9), (0, 1.9), (0,1.9) ], [], addr)

addr = 'TCPIP0::analpoweryzer1::inst0::INSTR'
psu_at = Keysight_N6705([(0, 1.9), (0, 1.9), (0, 3.3), (0,1.9) ], [], addr)

# Enable outputs
psu_k.enableOutputs()
psu_at.enableOutputs()

# Set output voltages
psu_at.setVoltage(1, 1.2)
psu_at.setVoltage(2, 1.8)
psu_at.setVoltage(3, 3.3)
psu_at.setVoltage(4, 1.2)

psu_k.setVoltage(1, 0.9)
psu_k.setVoltage(2, 0.6)
psu_k.setVoltage(3, 1.8)
psu_k.setVoltage(4, 1.2)

# Set measurement ranges
psu_k.setCurrentRange(1, 10e-3)
psu_k.setCurrentRange(2, 10e-3)
psu_k.setCurrentRange(3, 1e-5)
psu_k.setCurrentRange(4, 1e-5)

psu_at.setCurrentRange(1, 1e-1)
psu_at.setCurrentRange(2, 1e-1)
psu_at.setCurrentRange(3, 1.02e1)
psu_at.setCurrentRange(4, 1.02e1)

# Set sweep points
psu_k.setSweepPoints(4096)
psu_at.setSweepPoints(4096)
```

Figure 56: Power supply initialization.

8.7 Remote measurement setup

The vast majority of the measurements were performed remotely. Figure 57 presents the remote measurement setup. The PCB with an installed IC is connected to the Ardunio Due board and to the T&M instruments. A local workstation which is in the same LAN with the T&M instruments is running the measurement automation software presented in Chapter 7 and sending commands to the instruments. A remote workstation is connected via a remote desktop connection to the local workstation, in order to launch the measurement sequences. Microsoft's Remote Desktop for Windows or KRDC for Linux operating systems can be used for establishing the remote desktop connection.

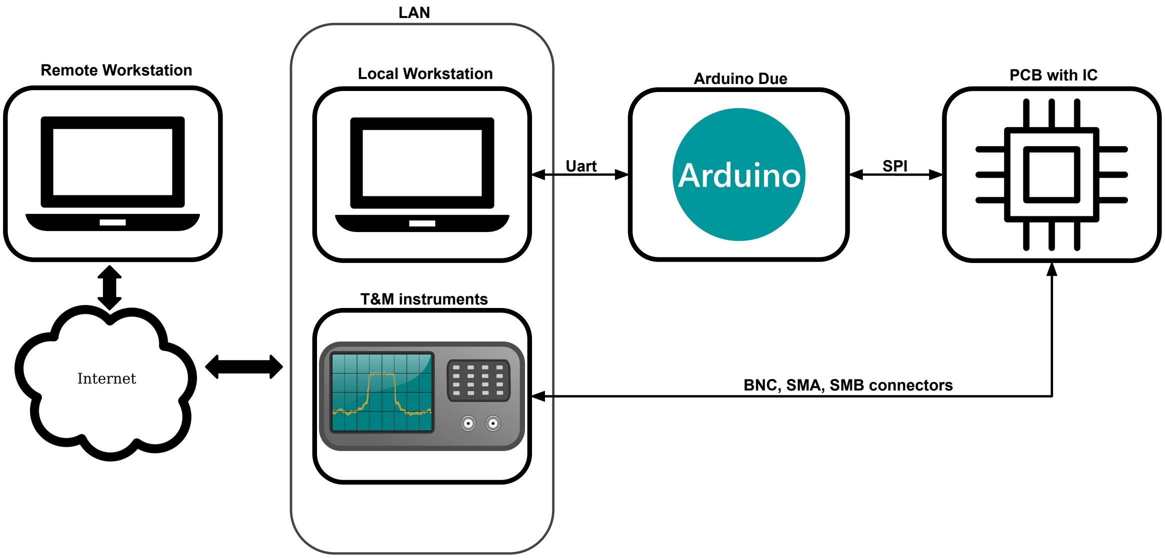


Figure 57: Remote measurement setup.

9 Display driver

This chapter presents the display driver IC block. Section 9.1 introduces electrophoretic displays and the particular device that was chosen for testing and demonstrating the driver. Section 9.2 presents the conceptual mode of operation of the driver and the charge pump that supplies the high voltage level. In Section 9.3 the PCB which was designed for connecting the chosen display to the driver is presented. Section 9.4 explores the measurement framework established for testing the display driver. Finally, Section 9.5 presents and discusses the measurement results.

9.1 Electrophoretic display

Electrophoresis is a new technology based on the manipulation of charged, colored submicron particles transitioning in a colloidal suspension. An electrophoretic display cell is made of two transparent electrodes, each protected by a transparent plate, enclosing in the middle a thin layer of died liquid. One of the electrodes is split into multiple segments and each segment needs to be connected to an individual voltage source. The other electrode is connected to a voltage source of opposite polarity. When the charged particles are attracted to the front electrode, the so-formed colored pattern becomes observable. In the opposite case, when the charged particles are attracted to the bottom electrode, the incident light is absorbed and scattered by the colloidal suspension, reavealing the color of the dye (the display's background color). [38]

Ultra-low-power displays are common peripheral devices for energy autonomous sensor nodes, often used for interacting with the user [40]. A practical usage scenario would be a cold chain monitoring system, where the product would have attached to it an autonomous sensor node with temperature measurement capability, fitted with a one-bit display. The white color of the display would indicate that the cold chain has been properly maintained, while a black color would indicate that the cold chain has been broken and the product cannot be safely consumed anymore.

Electrophoretic displays are suitable candidates for this purpose, as they do not use a backlight, this feature reducing their power consumption drastically. This means, however, that the environment where they operate needs to be well-lit. Moreover, electrophoretic displays draw current only during image updates, being able to retain the displayed image even after power is lost. They typically have only two colors, black for the text and white for the background, which does not pose a problem for a one-bit display. Refresh rates are considerably lower than in other technologies such as LCD or AMOLED, which makes them unsuitable for applications where the user feedback is expected to be fast. [39]

An E-ink SC009221 three-digit electrophoretic display was chosen for demonstrating the one-bit display driver. Its active area has a diameter of 22.65 mm. Therefore, the active area is 4.03 cm2. The display requires no current to maintain a static image, while the image update is 0.5 μA per cm2, thus 2.015 μA for this particular display. The operating supply voltage is 5 V to 15 V. Nevertheless, proper display operation has been previously reported with 3.3 V in [40].

9.2 Driver description

A driver circuit is implemented on-chip for enabling the IC to interface with low-power electrophoretic displays. The IC cannot operate the display directly from the available supplies, as their voltage is too low. Figure 58 presents the display driver diagram: it consists of two identical circuits, each circuit controllable by two data bits, programmable via the SPI interface provided by the IC. By choosing different bit configurations, the driver's output can be either set to VDD_HIGH_DISP, 0V or in a high impedance mode. A CMOS push-pull output stage with independent push and pull controls is used for driving each display pin. The PMOS transistor is controlled by the signal coming from the DISP_BITS_P<1:0> bits. However, the signal level is too low for driving the PMOS into cutoff, as the IC operates at 1.2 V levels and VDD_HIGH_DISP is 3.6 V. For addressing this issue, a level shifter (LS) is used. An inverter (INV) is employed for buffering the 1.2V digital signal coming from the IC and driving the level shifter. This ensures that the rising time of the signal is well-defined regardless of the connection parasitics. Using an inverter instead of a buffer saves area and power consumption, as the latter is accomplished by cascading two inverters. The NMOS transistor is designed to be directly controlled by the digital signals which are coming from the IC. The allowed values for the control bits of the driver are described by the ordered pairs

$$(\text{DISP\_BITS\_P<x>}, \text{DISP\_BITS\_N<x>}) \in \{(0,0),(0,1),(1,0)\}, \tag{18}$$

where x is 1 or 0, DISP_BITS_P and DISP_BITS_N are shown in Figure 58.

If the ordered pair (1,1) was controlling the driver, then VDD_HIGH_DISP would be shorted to ground, drawing a large amount of power and possibly damaging the circuit.

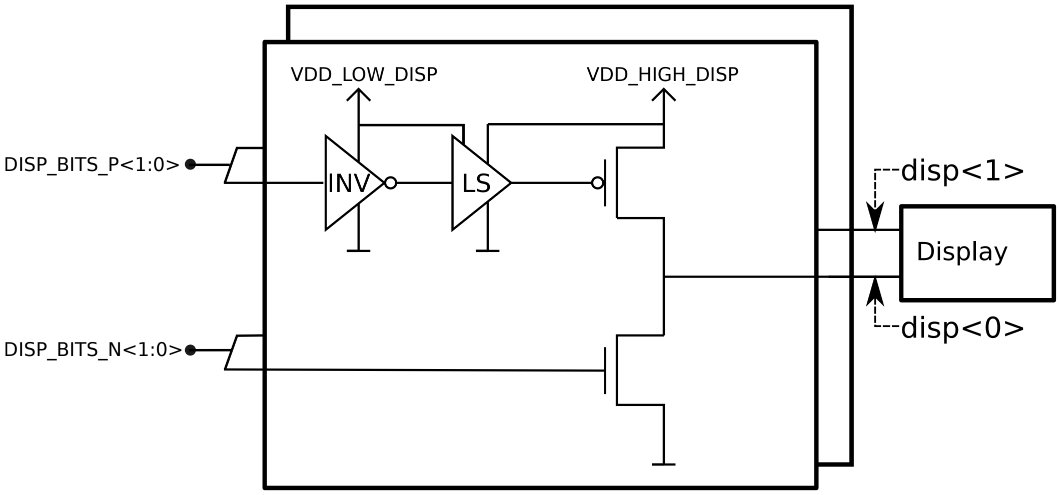


Figure 58: Display driver diagram.

The VDD_HIGH_DISP driver supply voltage is provided by a charge pump-based DC-DC converter circuit, which consists of an NOL and a switched-capacitor DC voltage tripler (CP). The VDD_LOW_DISP (1.2 V) is tripled by the charge pumps and output to VDD_HIGH_DISP (3.6 V). Figure 59 presents the conceptual diagram of the DC converter. During the testing of the driver, VDD_HIGH_DISP can be connected to an external 3.3 V supply.

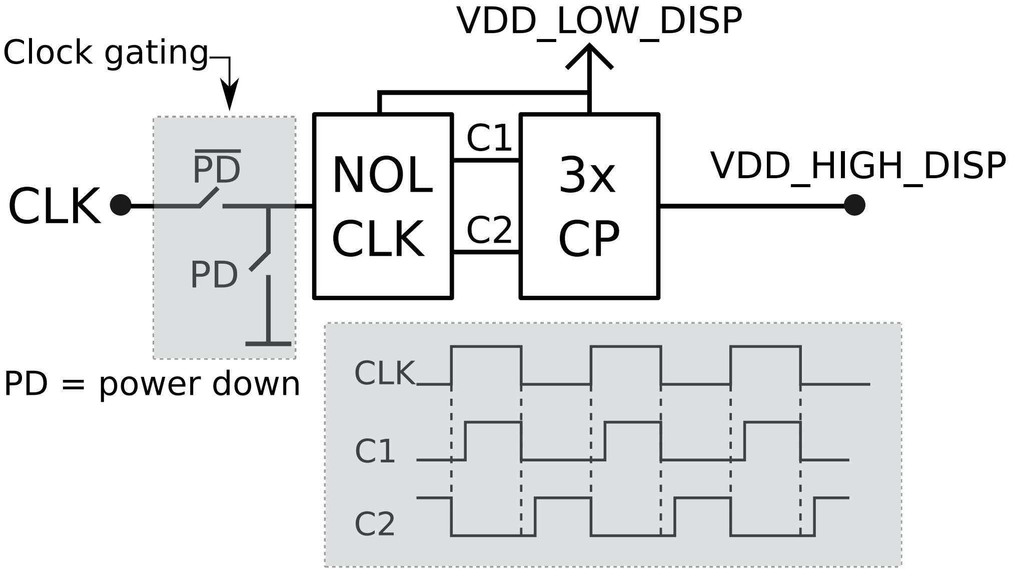


Figure 59: Charge pump diagram.

9.3 Test PCB

A PCB was developed for testing the driver with the E-ink SC009221 display. The PCB allows either individual control of the display pins, or wiring it as a one-bit display (full white or full black image) via the provided headers. Soldering the FPC cable pins for one-bit operation without the PCB would not be easily achievable.

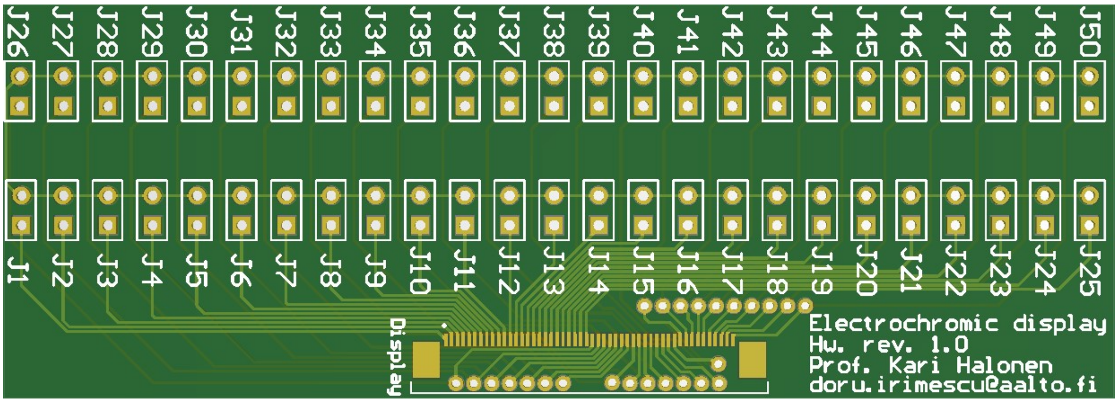

J26 J27 J28 J29 J30 J31 J32 J33 J34 J35 J36 J37 J38 J39 J40 J41 J42 J43 J44 J45 J46 J47 J48 J49 J50
J1 J2 J3 J4 J5 J6 J7 J8 J9 J10 J11 J12 J13 J14 J15 J16 J17 J18 J19 J20 J21 J22 J23 J24 J25
Display
Electrochromic display
Hw. rev. 1.0
Prof. Kari Halonen
doru.irimescu@aalto.fi

Figure 60: Display test PCB.

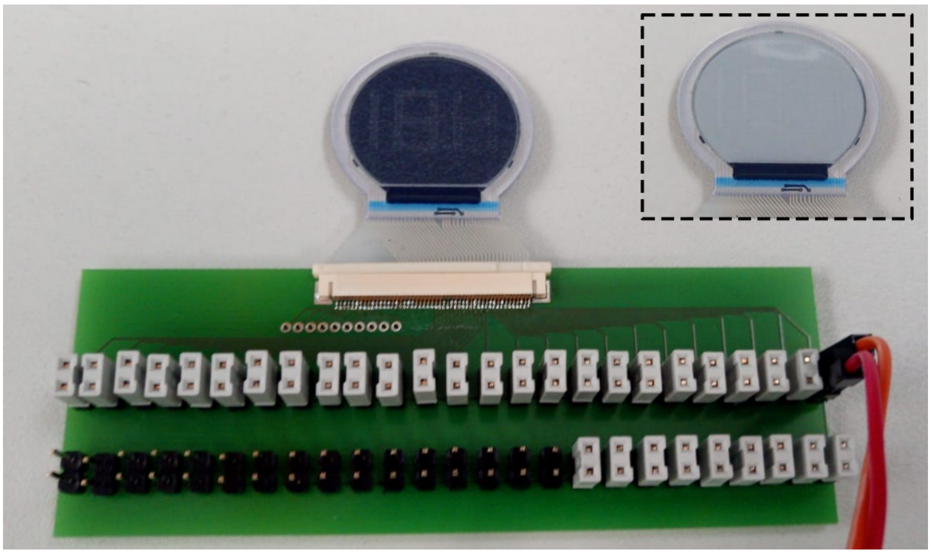


Figure 61: Assembled display test PCB.

9.4 Measurement setup

The transient current consumption of the display (or the driver) during the image update could not be measured directly. A special measurement setup had to be prepared (Figure 62): The voltage drop across a 50 kΩ sense resistor was measured between points A and B with an oscilloscope, from which the current can be calculated using Ohm's law:

$$I_{disp} = \frac{V_A - V_B}{R_{sense}}, \tag{19}$$

where I_{disp}, V_A, V_B, R_{sense} are defined in Figure 62.

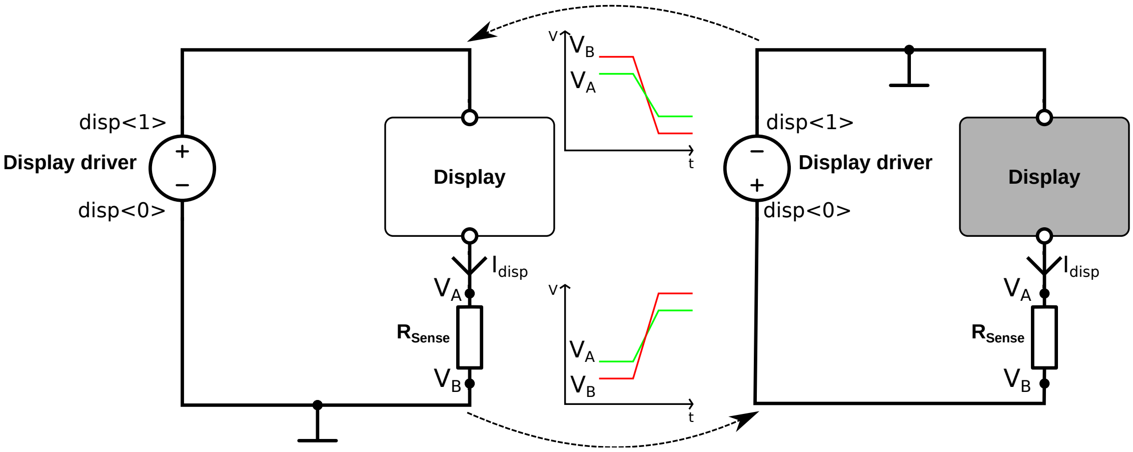


Figure 62: Display current measurement setup.

Figure 62 presents also the idealized expected waveforms during the two possible transitions: the two driver outputs are labeled disp<1> and disp<0>, as in Figure 58. When disp<1> is transitioning from high to low and disp<0> is transitioning from low to high, the display switches color from white to black. The expected measured waveforms are depicted at the center-bottom of the figure. At this point, the potential difference between the two points A and B starts as being positive, crosses zero and ends up being negative. When disp<1> is transitioning from low to high and disp<0> is transitioning from high to low, the display switches color from black to white. The expected measured waveforms are depicted at the center-top of the figure. Now, the potential difference between the A and B measurement points starts as being negative, crosses zero and ends up being positive.

At this stage, it is important to consider how the measurement will take place: if an attempt is made to measure the voltage drop over the sense resistor with one single-ended oscilloscope channel, then the probe's signal reference terminal will either be attached at point A or B. However, the signal reference terminal is connected to protective earth, as a measure of precaution taken by oscilloscope manufacturers. If another device (such as the Arduino Due) which has the ground connected to protective earth is connected to the characterization PCB, the PCB's ground will also be connected to protective earth. Thus, either the point A or B of the measurement will be connected to ground at all times, either shorting the driver, the sense resistor or the display. In order to overcome this issue, a differential measurement has to be taken: this could either be achieved with a differential probe, or with a pseudo-differential measurement [41]. The pseudo-differential measurement setup was chosen due to lack of differential probes, presented in Figure 63.

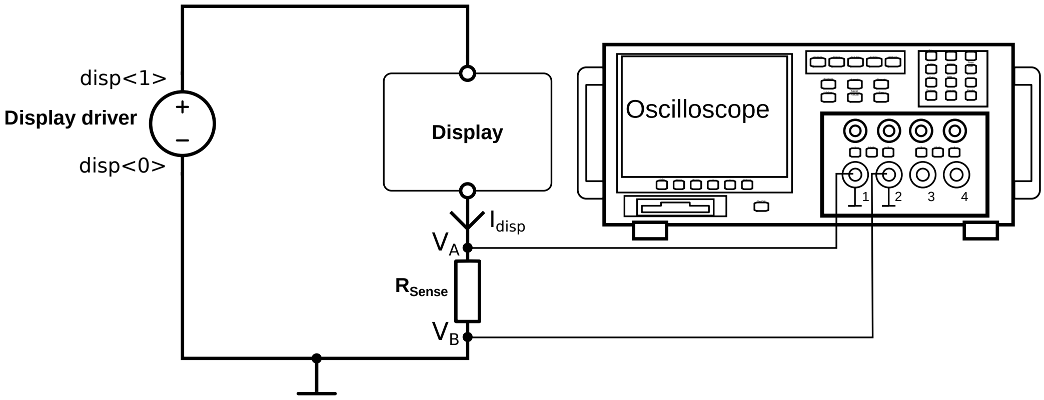


Figure 63: Display current pseudo-differential measurement setup.

9.5 Measurement results

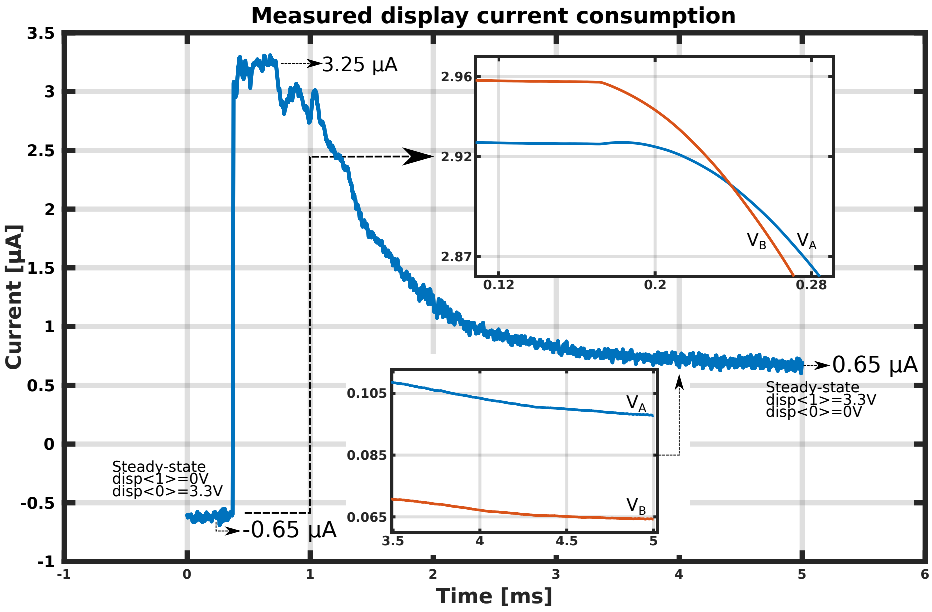


Figure 64: Display current consumption with 3.3 V driver supply voltage during black to white transition.

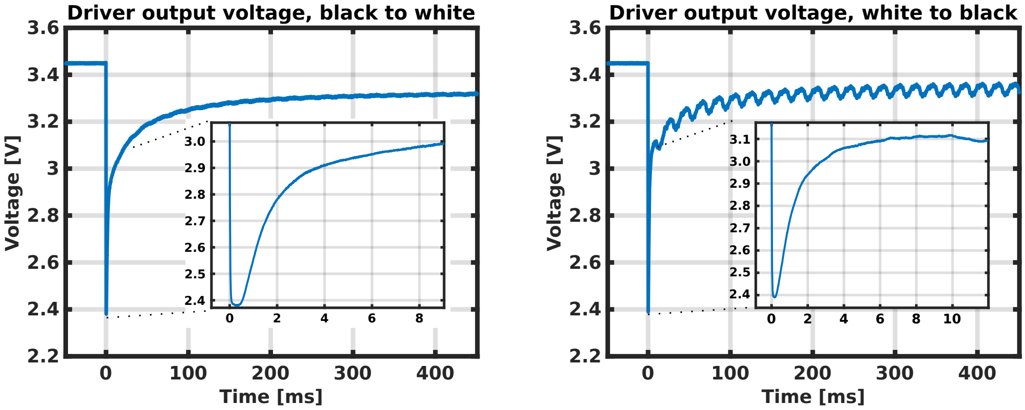


Figure 65: Measured display driver output voltages, using charge pump.

Figure 64 presents the measured display current consumption during an image update. The sense resistor which was used for the measurements has a value of 50 kΩ and the VDD_HIGH_DISP was connected to an external 3.3 V supply. The display leakage current is in the range of 0.65 μA in steady-state. However, there is no current draw when the display driver is set to high impedance mode.

Figure 65 presents the driver output voltage during two image updates, using the internal charge pump supply. The switched capacitor circuit's input clock signal frequency is 300 kHz. Oscillations were repetitively observed during white to black transitions, with a frequency of 50 Hz. This could be happening due to interferences from the mains electricity system.

Figure 66 shows the measured current consumption of the display driver and charge pump. The switched capacitor circuit's input clock signal frequency is 300 kHz. The current draw peaks during the display update, at 0.127 μA for the black to white and 0.125 μA for the white to black transitions.

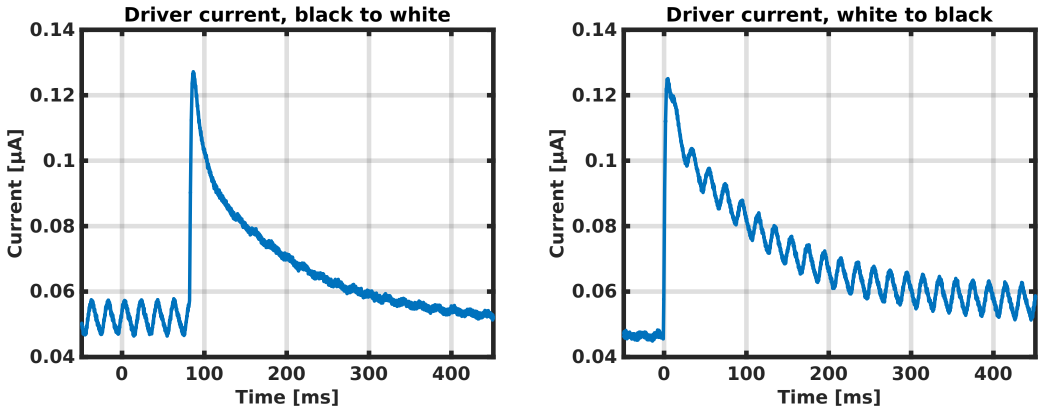


Figure 66: Current draw of driver and charge pump.

Overall, the display driver was proven fully functional and can be used to showcase the user interfacing capabilities of the IC. The PCB which was developed during this thesis work enables further prototyping of the system in scenarios such as in cold-chain management.

10 Conclusions

In this work, a platform used by the research team to test and characterize their IC was developed together with the necessary building blocks for a prototype that will showcase the main capabilities of the device. A four-layer PCB was designed and assembled and the necessary components that support the operation of the IC were selected. This included sensor emulation, impedance matching and filter design. Wireless communication between the IC and a computer was established, by using an USRP device and LabVIEW programming. A programming interface was written in Python, which enables researchers to program and configure the IC. A novel, reusable software architecture emerged, which can be used for future projects involving other ICs. Adapting the programming interface to contain all the necessary configuration parameters for the chip was a substantial part of the work. Measurement automation was deployed and a remote measurement setup was developed, which incorporated the hardware and software platform along with the measurement software to control T&M devices. Finally, the one-bit display driver was tested and measured, in order to validate the complete system.

The SPI communication interface of the IC was shown to work correctly, as the success of this thesis depended on it. The narrowband receiver and transmitter were proved to be functional. However, it was found that the transmitted data can use only 8 out of 16 bits in order to be received correctly. The one-bit display driver was tested and operated successfully.

This thesis showed the functionality of the programming interface and measurement automation system with the display driver. However, the presented setup is currently employed successfully for the evaluation of other IC blocks such as the ADC, REGU, EH and UWB transmitter.

This work laid the foundation for the prototyping of a concrete application scenario where the IC could address an industrial problem or could showcase its capabilities. Some IC blocks still need to be tested and characterized by the researchers. A plan that establishes which blocks to incorporate in the prototype needs to be developed. Next, a minimized version of the current PCB which would support only the necessary functions of the IC should be produced. Finally, using the software that was written, the IC could be configured for a particular use case.

Currently, the programming interface software benefits from automated unit tests, but as the codebase matures it will become more complex due to future iterations of improvement. Implementing a continuous integration framework could benefit future development, especially in the case where more IC programmers are customizing their own interfaces. When building a custom programming interface for a particular IC, most of the time will be spent with mapping the registers of the device to the interface. This laborious work could be greatly reduced by automating the process in the following way: a standardized file format should be planned by the researchers and used by the department by convention for describing the memory layout; then, a Python parser script could read a memory layout file and generate a bespoken programming interface suited for that particular IC.

References

[1] *Directive 2012/27/EU of the European Parliament and the Council of 25 October 2012 on energy efficiency, amending Directives 2009/125/EC and 2010/30/EU and repealing Directives 2004/8/EC and 2006/32/EC*, directive 2012/27/EU, European Parliament and Council, Nov. 14, 2012.

[2] K. Routh and T. Pal, "A survey on technological, business and societal aspects of Internet of Things by Q3, 2017", in Proc. *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, Feb. 2018, pp. 1-4.

[3] O. B. Akan, O. Cetinkaya, C. Koca and M. Ozger, "Internet of Hybrid Energy Harvesting Things", *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 736-746, Apr. 2018.

[4] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution: A Guide to Platform-Based Design*, Boston:Kluwer Academic Publishers, 2002, pp. 4.

[5] J. Salomaa, M. Pulkkinen, T. Haapala, M. Nurmi and K. Halonen, "Power management system for ultra-low power energy harvesting applications," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1086-1089.

[6] J. Salomaa, M. Pulkkinen, T. Haapala, S. S. Chouhan and K. Halonen, "Energy harvesting ASIC for autonomous sensors," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 2350-2353.

[7] M. M. Moayer, J. Salomaa, M. Pulkkinen and K. Halonen, "Ultra-Low Power Wide-Dynamic-Range Universal Interface for Capacitive and Resistive Sensors," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1-5.

[8] M. Pulkkinen, J. Salomaa, M. M. Moayer, T. Haapala and K. Halonen, "462-nW 2-axis gesture sensor interface based on capacitively controlled ring oscillators," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1-4.

[9] M. Pulkkinen, J. Salomaa and K. Halonen, "Low-Power Single-Stage Narrowband Transmitter Front-End for 433-MHz Band," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1-4.

[10] T. Haapala, T. Rantataro and K. A. I. Halonen, "A Fully Integrated Programmable 6.0–8.5-GHz UWB IR Transmitter Front-End for Energy-Harvesting Devices", *IEEE Journal of Solid-State Circuits*, vol. 55, no. 7, pp. 1922-1934, Apr. 2020.

[11] M. Pulkkinen, T. Haapala, J. Salomaa and K. Halonen, "45.2% Energy efficiency improvement of UWB IR Tx by use of differential PPM in 180nm CMOS," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 193-196.

[12] M. Pulkkinen, L. Aaltonen and K. Halonen, "SPI interface, mux-based synchronizer and DSP unit for a MEMS-based accelerometer," in Proc. *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 453-456.

[13] M. M. Moayer, J. Salomaa and K. A. I. Halonen, "A 0.39–3.56-μW Wide-Dynamic-Range Universal Multi-Sensor Interface Circuit", IEEE Sensors Journal.

[14] J.-M. Irazabal and S. Blozis, *I2C Manual*, document AN10216-01, Philips Semiconductors, Mar. 24, 2003.

[15] IXYS, *NCD2400M - Wide capacitance range, non-volatile digital programmable capacitor*, document DS-NCD2400M-R01, rev. 1, IXYS integrated circuits division, Apr. 3, 2018.

[16] *256-Tap, Nonvolatile, I2C-Interface, Digital Potentiometers*, document 19-3185, rev. 4, Maxim Integrated Products, Apr. 2010.

[17] A. Sain and K. L. Melde, "Impact of Ground via Placement in Grounded Coplanar Waveguide Interconnects", *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 6, no. 1, pp. 136-144, Jan. 2016.

[18] J. Coonrod, “Understanding when to use FR-4 or high frequency laminates”, OnBoard Technology, pp. 26–30, Sep. 2011.

[19] E. Bogatin, *Signal and Power Integrity - SIMPLIFIED* 2. edition. Prentice Hall, 2010.

[20] Anritsu *The Essentials of Vector Network Analysis* 1. edition. The United States, Anritsu Company, 2009.

[21] Wentworth, S. M. *Applied Electromagnetics* 1. edition. The United States of America, John Wiley & Sons, Inc., 2007.

[22] Steer, M. *Microwave and RF Design A Systems Approach* 2. edition. Raleigh, NC., SciTech Publishing, 2010.

[23] Zumbahlen, H. *Basic linear design* 1. edition. Analog Devices, Inc., 2007.

[24] Sedra, A. S., Smith, K. C. *Microelectronic design* 7. edition. New York Oxford, Oxford University Press, 2015.

[25] Smith, J. O., "Phase and Group Delay" in *Introduction to digital filters with audio applications*, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University 2007. Accessed: Sep. 7, 2020. [Online]. Available: https://ccrma.stanford.edu/~jos/filters/Group_Delay.html

[26] T. Haapala, T. Rantataro and K. A. I. Halonen, *A Fully Integrated Programmable 6.0–8.5-GHz UWB IR Transmitter Front-End for Energy-Harvesting Devices*, IEEE Journal of Solid-State Circuits, vol. 55, no. 7, pp. 1922-1934, Jul. 2020.

[27] Xiong, F. *Digital modulation techniques* 2. edition. Boston and London, Artech House, 2006.

[28] National Instruments, "What Is NI USRP Hardware?", Mar. 3, 2019. Accessed: Sep. 7, 2020. [Online]. Available: https://www.ni.com/fi-fi/innovations/white-papers/11/what-is-ni-usrp-hardware-.html

[29] National Instruments, "What Is NI USRP LabVIEW?", Mar. 3, 2019. Accessed: Sep. 7, 2020. [Online]. Available: https://www.ni.com/fi-fi/shop/labview.html

[30] National Instruments, "Virtual instrumentation", Mar. 3, 2019. Accessed: Sep. 7, 2020. [Online]. Available: https://www.ni.com/fi-fi/shop/labview.html

[31] S. Sheng-Ju, "Implementation of Cyclic Redundancy Check in Data Communication," in Proc. *International Conference on Computational Intelligence and Communication Networks (CICN)*, Dec. 2015, pp. 529-531.

[32] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in Proc. *International Conference on Dependable Systems and Networks*, Jul. 2004, pp. 145-154.

[33] T. Peters, "The Zen of Python" Aug. 22, 2004. Accessed: Sep. 7, 2020. [Online]. Available: https://www.python.org/dev/peps/pep-0020/

[34] S. Cass, "The Top Programming Languages 2019", Sep. 6, 2019. Accessed: Sep. 7, 2020. [Online]. Available: https://www.python.org/dev/peps/pep-0020/

[35] ISO/IEC, *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, International Organization for Standardization, Tech. Rep., 2010.

[36] SCPI Consortium. *Standard Commands for Programmable Instruments (SCPI)*. (1999). Accessed: Sep. 7, 2020. [Online]. Available: https://www.ivifoundation.org/docs/scpi-99.pdf

[37] LeCroy Corporation. *Automation Manual For Wave Expert Series Oscilloscope*. (2009). Accessed: Sep. 7, 2020. [Online]. Available: http://cdn.teledynelecroy.com/files/manuals/we-automation-manual-e.pdf

[38] A. L. Dalisa, "Electrophoretic display technology," IEEE Transactions on Electron Devices, vol. 24, no. 7, pp. 827-834, Jul. 1977.

[39] K. Bonheur, "Electrophoretic display: Advantages and disadvantages", Dec. 26, 2018. Accessed: Sep. 7, 2020. [Online]. Available: https://www.versiondaily.com/advantages-disadvantages-electrophoretic-display/

[40] T. Haapala, "Low-power impulse radio transmitter in 180 nanometer CMOS", M.S. thesis, School of Electrical Engineering, Aalto Univ., Espoo, Finland, Oct. 2015.

[41] Tektronix. *Fundamentals of Floating Measurements and Isolated Input Oscilloscopes*. (2011). Accessed: Sep. 7, 2020. [Online]. Available: https://download.tek.com/document/3AW_19134_2_MR_Letter.pdf