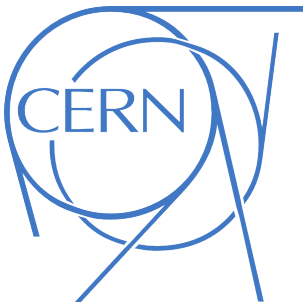


Signal generation and acquisition platform for a stretched-wire Beam Position Monitor test bench

—

Master's Thesis

Henrik Dötsch
68028

At the Department of Electrical Engineering and Information Technology
Hochschule Karlsruhe - University of Applied Sciences

—

European Organization for Nuclear Research (CERN)

CERN-THESIS-2023-017
09/03/2023

Reviewer:	Prof. Dr.-Ing. Franz Quint
Second Reviewer:	Prof. Dr.-Ing. Manfred Litzenburger
Supervisor:	Michal Krupa

06.03.2023

Abstract

The goal of this thesis was to develop new electronics for an existing test bench to calibrate BPMs. The existing electronics were thereby replaced by a state of the art Xilinx MPSoC. More precisely the Zynq UltraScale+ RFSoC ZCU111, that features eight high speed ADCs and DACs inside its chip.
The test-bench works by creating a signal at a variable frequency that is outputted on a small wire. This wire is threaded through a BPM and the received signal at each electrode is measured. To obtain the Sensitivity of a BPM the wire is moved to different positions inside the BPM and a measurement is performed. The measurements are normalised using a Delta-Sigma-Normalisation that will result in the required characteristics for the BPM, the Sensitivity and the Offset.
The board is working as a measurement device performing a s_{21}-parameter measurement. Therefore a signal is generated inside the chip at an adjustable frequency of up to 1 GHz. This is achieved using a CORDIC algorithm to calculate the required sine values on the go instead of storing them inside a LUT. The received signal is processed using the Goertzel algorithm to calculate exactly one bin of a DFT, namely the frequency bin at which the sought frequency lies. Signal generation and acquisition are performed inside the FPGA of the ZCU111.
To handle the communication with the LabView software, a simple SCPI parser has been developed and implemented over the USB interface.
Out of the measurements with the new and old setup, it can be clearly seen, that the instrument developed in this thesis outperforms the old VNA that has been used to take measurements before. Additionally it adds the possibility to perform eight electrode measurements simultaneously, while before only two were possible at a time. This makes the test bench more intuitive as well as speeds up the measurement process by a factor of at least 2.5.

Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung einer neuen Messelektronik für einen bereits bestehenden Prüfstand zur Kalibrierung von BPMs. Die vorhandene Elektronik, bestehend aus einem VNA, wurde dabei durch einen hochmoderne Xilinx MPSoC ersetzt, genauer gesagt den Zynq UltraScale+ RFSoC ZCU111, der mit acht high speed ADCs und DACs ausgestattet ist.
Der Prüfstand funktioniert, indem er ein Signal mit variabler Frequenz erzeugt, das über einen kleinen Draht ausgegeben wird. Dieser Draht wird durch einen BPM gefädelt und das empfangene Signal an jeder Elektrode wird gemessen. Um die Empfindlichkeit eines BPM zu erhalten, wird der Draht zu verschiedenen Positionen innerhalb des BPM bewegt und eine Messung durchgeführt. Die Messungen werden mit Hilfe einer Delta-Sigma-Normierung normalisiert, die zu den erforderlichen Kenndaten für die BPM, die Empfindlichkeit und den Offset führt.
Das Board arbeitet als Messgerät, das eine s_{21}-Messung durchführt. Dazu wird im FPGA ein Signal mit einer einstellbaren Frequenz von bis zu 1 GHz erzeugt. Dieses wird mit Hilfe eines CORDIC-Algorithmus erzeugt, der die Ausgangswerte im laufenden Betrieb erzeugt anstelle sie aus einer LUT auszulesen.// Das empfangene Signal wird mit dem Goertzel-Algorithmus verarbeitet, um genau einen Bin einer DFT zu berechnen, nämlich das Frequenz-Bin, auf dem die gesuchte Frequenz liegt. Die Signalerzeugung und Signalerfassung erfolgt innerhalb des FPGA des ZCU111.
Für die Kommunikation mit der LabView-Software wurde ein einfacher SCPI-Parser entwickelt und über die USB-Schnittstelle implementiert.
Aus den durchgeführten Messungen mit dem neuen und dem alten Aufbau geht eindeutig hervor, dass das in dieser Arbeit entwickelte Gerät den alten VNA, der zuvor für die Messung verwendet wurde, übertrifft. Zusätzlich bietet es die Möglichkeit acht Elektrodenmessungen gleichzeitig durchzuführen, während vorher nur zwei gleichzeitig möglich waren. Dies macht den Prüfstand intuitiver und beschleunigt den Messvorgang um einen Faktor von mindestens 2,5.

Contents

List of Figures

List of Tables

List of Acronyms

ACLR Adjacent Channel Leakage Ratio. 20, 21

ADC Analog-Digital-Converter. i, iii, xiii, 12, 20–22, 28, 29, 31, 33, 34, 44–46, 48–51, 61, 67, 73

ALICE A Large Ion Collider Experiment. 2

AMBA ARM Advanced Microcontroller Bus Architecture. 20

APU Application Processing Unit. x, 16, 17, 54

ASCII American Standard Code for Information Interchange. 28

ATLAS A Toroidal LHC Apparatus. 2

AXI Advanced eXtensible Interface. 20, 39, 44, 45

BPM Beam Position Monitor. i, iii, ix, xi, 3–7, 9–12, 32, 34, 51, 65–67, 71, 73

CERN Conseil Européen Pour La Recherche Nucléaire. ix, 1, 2

CMS Compact Muon Solenoid. 2

CORDIC Coordinate Rotational Digital Computer. i, iii, x, 23–25, 37–42, 46, 48, 73

DAC Digital-Analog-Converter. i, iii, x, xiii, 12, 20–22, 31–34, 37, 39, 41, 46, 48, 49, 51, 61, 67

DFT Discrete Fourier Transform. i, iii, 25–28, 43–46, 49, 73

DHCP Dynamic Host Configuration Protocol. 58

DSP Digital Signal Processing. 18, 46

ENOB Effective Number of Bits. 20, 21

FAT32 File Allocation Table 32. 54, 60

FFT Fast Fourier Transform. 25

FIFO First In First Out. 18, 44–46

FMC FPGA Mezzanine Card. 20

FPGA Field-Programmable Gate Array. i, iii, ix, 12, 15, 18, 37, 39, 41, 43, 45, 46, 48, 53, 57–60, 62, 73

FPU Floating Point Unit. 16, 17

FSBL First Stage Bootloader. 54

FSK Frequency-Shift-Keying. 25

GPIB General Purpose Interface Bus. 28

GPIO General-Purpose-Input-Output. 39

GUI Graphical User Interface. 11

IIR Infinite Impulse Response. x, 26, 27

IP Intellectual Property. 20, 44, 63

IP Internet Protocol. 28, 58

LHC Large Hadron Collider. xv, xvi, 1, 2, 12, 73

LHCb LHC Beauty. 2

Linac4 Linear Accelerator 4. 2

LUT Look Up Table. i, iii, 18, 24, 46

MAC Media Access Control. 58

MPE Media Processing Engine. 16

MPSoC Multi-Processor System On Chip. i, iii, ix, 15, 16

MSB Most-Significant-Bit. 38

NSD Noise-Spectral Density. x, 20–22, 31, 33

PL Programmable Logic. 15–18, 20, 43, 46, 48, 54

PLL Phase-Locked-Loop. 46

PS Processing System. 2, 15, 16, 20, 37, 39, 43–46

PSB Proton Synchrotron Booster. 2

PU Pick-Up. ix, 3, 4, 9–12, 33, 65, 67

QAM Quadrature Amplitude Modulation. 21

RAM Random-Access Memory. 16, 18, 46, 73

RF Radio-Frequency. 1–3, 16, 22, 62

RFDC Radio-Frequency Data Converter. 46, 48, 63

RFSoC Radio Frequency System On Chip. i, iii, ix, 12, 15, 16, 18, 20, 49, 53, 54, 65, 67, 69, 71–73

ROM Read-only Memory. 18

RPU Real-Time Processing Unit. x, 16–18

SCPI Standard Commands for Programmable Instruments. i, iii, xiii, 12, 28, 29, 61

SFDR Spurious Free Dynamic Range. 20, 21

SMA SubMiniature version A. 20, 48

SNR Signal-to-Noise Ratio. x, 20–22, 31, 33

SoC System-On-Chip. 15, 53

SPS Super Proton Synchrotron. 2

TCP Transmission Control Protocol. 28

USB Universal Serial Bus. i, iii, 12, 28, 54, 56, 58–62, 64

VNA Vector-Network-Analyzer. i, iii, 10–12, 28, 65, 69, 71–73

Chapter 1

The Cern Laboratory

1.1 Introduction

After the Second World War ended, European science was no longer world class. A handful of visionary scientists from Europe and North America imagined the creation of a physics research facility. Their vision was to stop the brain drain to America and to establish unity in post-war Europe. Finally, in June 1953, the final draft of the CERN Convention was agreed upon and signed by 12 new Member States. In July 1955 CERNs first foundation stone was laid in the Franco-Swiss border near Geneva in Switzerland [1].
Since its birth in 1953, CERN has been responsible for many major achievements in physics. This includes the discovery of *W and Z bosons*[1] in 1983, the invention of the *World Wide Web* in 1989, the particle accelerator *Large Hadron Collider* in 2008 and the discovery of the *Higgs boson*[2] in 2012. CERN's contributions to society are not constrained to High-Energy Physics. CERN also develops state-of-the-art technologies that are applied in areas such as health, safety, Industry 4.0 and much more.

1.2 The accelerator complex

CERN operates the world's largest and most powerful particle accelerator, the Large Hadron Collider (LHC). It is composed of a 27 kilometer long ring of superconducting magnets with multiple accelerating structures that boost the particle's energy [2]. Inside the accelerator, two high-energy beams travel, in two separate vacuum chambers, at close to the speed of light before they are made to collide. In order to avoid collisions with gas molecules inside the accelerator, the beam pipes are filled with a vacuum as empty as interstellar space. The trajectories of the particle beams are controlled by a strong magnetic field maintained by superconducting electromagnets. These electromagnets use a current of over 11 kA to produce a magnetic field of 8.3 T which is 100.000 times more powerful than the Earth's magnetic field. In order to keep the magnets in their superconducting state, they have to be cooled down to 1.9 K (−271.3 °C) [3]. To accelerate particles, accelerators are fitted with resonant structures containing a strong modulating electromagnetic field known as RF cavities. Charged particles injected into this

[1] W and Z bosons mediate the weak force [1]
[2] Higgs boson is the quantum particle of the Higgs field. Particles that interact with the Higgs field get mass [1]

field receive an electrical impulse that accelerates them. Figure 1.1 shows the CERN accelerator complex which consists of multiple interconnected accelerators, each gradually changing the energy of accelerated particles.

The journey of the proton beams starts in the Linear Accelerator 4 (Linac4) which accelerates negatively-charged hydrogen ions[3] up to the energy of 160 MeV at which they enter the Proton Synchrotron Booster (PSB). The ions are stripped of their electrons during their transition from the Linac4 to the PSB where the protons are further accelerated to 2 GeV before injection to the Processing System (PS). The PS brings the beam energy up to 26 GeV and sends them to the Super Proton Synchrotron (SPS) where they are pushed to 450 GeV. The particles are finally transferred to the two rings of the LHC. The two beams injected into the LHC circulate in opposite directions and the RF cavities increase their energy to 6.7 TeV – more than 14 times their injection energy. At this point, the two beams are ready for collisions inside the four detectors (ALICE, ATLAS, CMS and LHCb) where the collision energy reaches 13.4 TeV [4].

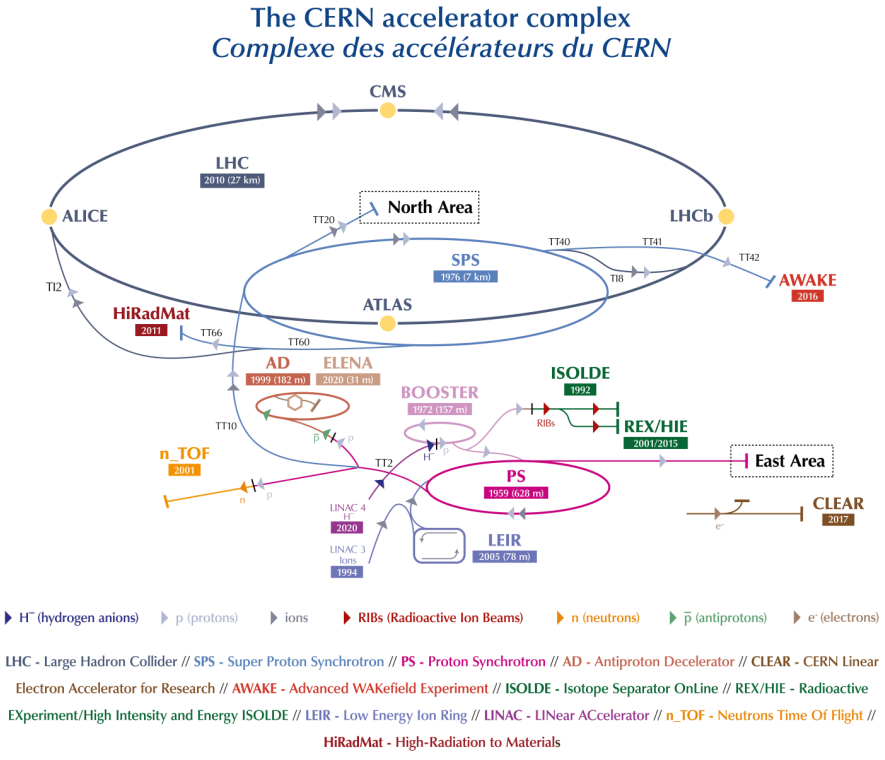


Figure 1.1: The CERN accelerator complex [4]

[3]Hydrogen ions (H^-) are hydrogen atoms with an additional electron

1.3 Beam Position Monitors

The present thesis describes work done on a calibration test bench for Beam Position Monitors (BPMs) which are non-destructive diagnostic devices widely used in particle accelerators. A BPM (also referred to as a Pick-Up (PU)) is equipped with metal electrodes that generate signals induced by the electric field of the particle beam. Because the electric field of a bunched beam[4] is time-dependent, the BPM electrodes produce time-varying signals the bandwidth of which can reach several gigahertz and often requires processing using RF techniques. The BPMs are used to observe the transverse beam position, also called the center of charge of the beam. The power of the signal generated by a BPM electrode is a function of the distance between the beam and the electrode. As the beam gets closer to an electrode, the power of the generated signal increases. Therefore, with four BPM electrodes installed crosswise at the vacuum chamber wall, it becomes possible to calculate the beam position by comparing the signals generated by the opposite electrodes [5]. This calculation is typically performed by a dedicated read-out system which acquires and processes the signals generated by the pickup electrodes and computes the beam position such that it can be further used. Figure 1.2 shows the general structure of a BPM system installed in an accelerator.

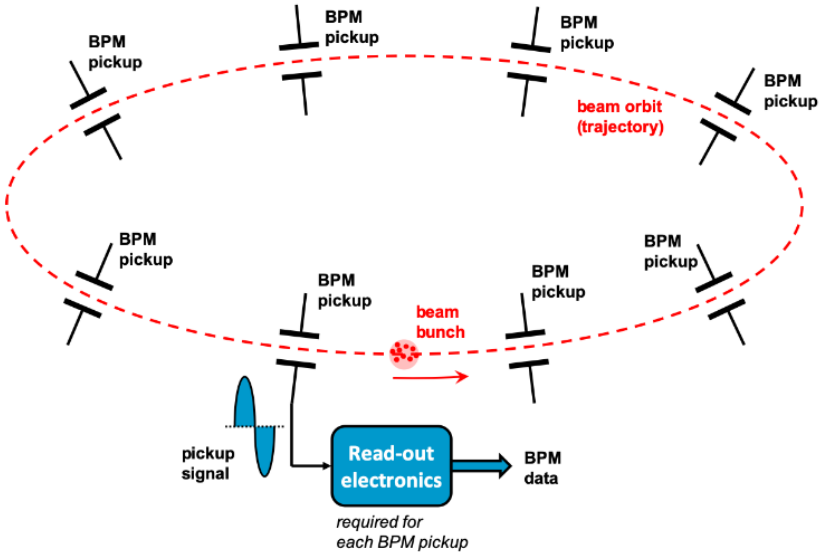


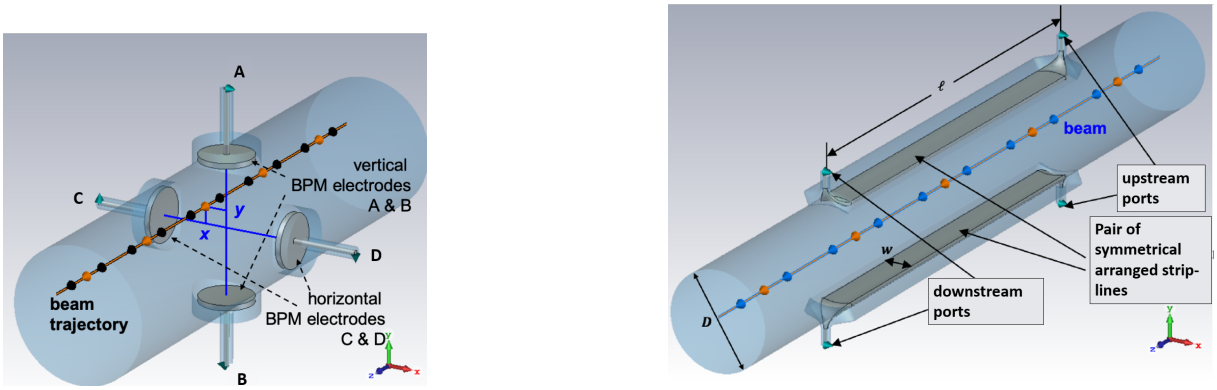
Figure 1.2: BPM system [6]

In ring accelerators, it is possible to average the BPM data over many revolutions of the beam around the ring whereas in a linear accelerator, this can be done over many consecutive pulses. Doing this for each BPM separately provides a high-resolution measurement of the beam's orbit (in ring accelerators) or trajectory (in linear accelerators), which can be used for the alignment of the beam or creating an orbit feedback system to automatically maintain the beam at the desired transverse position.

[4] In a bunched beam particles are distributed into pulses (bunches)

Button BPM

Two popular BPM types are shown below in figure 1.3. The left-hand side depicts a button BPM whereas the right-hand side shows a stripline BPM which in the figure is equipped with beam-sensing electrodes only in the vertical plane, but other designs can also feature electrodes for horizontal measurement.



(a) Button BPM

(b) Stripline BPM

Figure 1.3: Different types of BPMs [6]

A "button" PU consists of four round metal electrodes which are arranged symmetrically along the horizontal and vertical axes of the vacuum chamber. The horizontal and vertical offset of the beam's centre of electric charge with respect to the mechanical center of the vacuum chamber is the transverse beam position or displacement. This offset has to be monitored with high resolution, accuracy and repeatability, which is the ultimate goal of a BPM system. Button-style BPMs are the most widely-used pick-ups because of their relative simplicity, robustness and compact design. Their characteristics are reproducible at a reasonable cost. Button-style BPMs can be designed in different sizes and dimensions for their specific usages. The beam-sensing button electrodes couple capacitively to the beam's electric field through the capacitance C_b between the electrode and the grounded vacuum chamber. Such an arrangement forms a high-pass filter with the load impedance R_l. A characteristic equivalent circuit of a single button electrode can be seen in figure 1.4 [6].

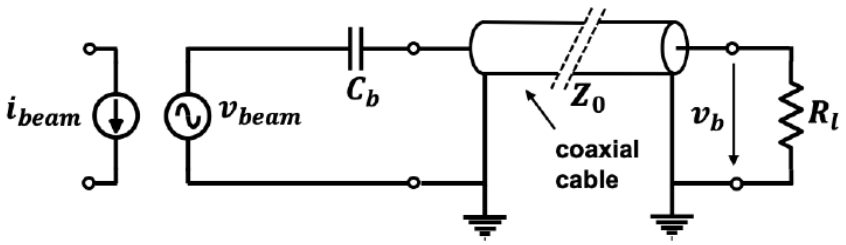


Figure 1.4: Equivalent circuit of a button electrode [6]

To detect the position of the particle beam in the vacuum chamber, the BPM couples to the electric field of the particle beam. This field displaces an electric charge on the beam-facing side of the electrodes which generates a current that can be measured as a voltage over the load resistor R_l. From the relative magnitude of the voltages in the 4 electrodes, the absolute position of the particle beam can be calculated using a set of calibration coefficients.

To understand the measurement principle of a button-BPM, it is useful to first analyse the difference between stationary and moving charges. Figure 1.5 shows the electric field lines of a stationary point charge in open space (left-hand side) and of a stationary point charge enclosed between two infinitely large conductive plates, which are an idealised representation of the beam-facing sides of the vacuum chamber.

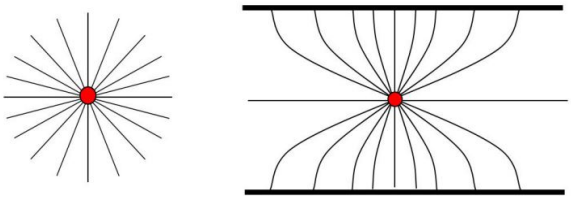


Figure 1.5: Electrical field of a stationary point charge in open space and between two conductive plates [6]

If the point charge moves at a certain speed in a certain direction, the electrical field lines will compress depending on the velocity of the particle, as shown in figure 1.6 for a particle displaced towards one of the conductive plates.

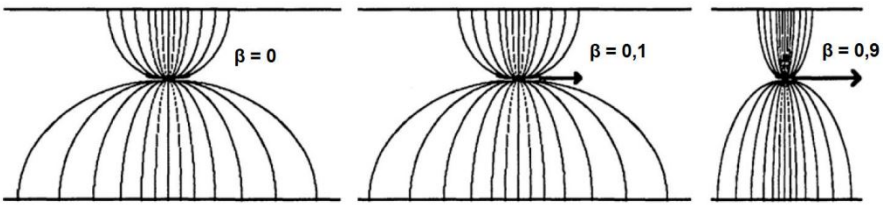


Figure 1.6: Electric field of a moving point charge [6]

The velocity of particles in an accelerator is often described using the relativistic factor β which is the speed of a particle normalized by the speed of light in vacuum: $\beta = \frac{v}{c}$. With increasing β, the longitudinal component of the electric field (i.e. the component parallel to the direction of motion) is contracted and for a particle travelling very close to the speed of light ($\beta \approx 1$) all the electric field lines are perpendicular to the direction of motion.

The electric field of a particle in the centre of a vacuum chamber with a round cross-section is symmetrical, as seen on the left-hand side of figure 1.7. If a particle is at an offset d_z from the vacuum chamber centre, the electrical field surrounding it is not anymore symmetric and it induces different signals in the opposing electrodes (marked as A and B in figure 1.7).
Button BPMs couple to the changing electric field of a particle beam inside a vacuum chamber. Since in most high-energy accelerators, the particles move virtually at the speed of light, only the transverse electric field lines are present. Since the measured particle beam is usually bunched, i.e. the beam particles are longitudinally grouped into so-called bunches, and therefore its electric field is time-variant, the output signal of a button BPM electrode is contained within relatively high frequencies. The signal generated by each electrode is proportional to the total electric charge of the beam and inversely proportional to the distance between the beam and the electrode.
To determine the two-dimensional position of a particle beam, four electrodes are typically used with two

for the horizontal (x) plane and two for the vertical (y) plane. The beam induces a stronger signal on the electrodes it is closer to. This characteristic will be used to determine the position of the particle beam.

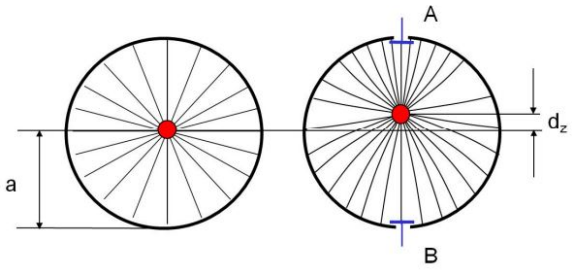


Figure 1.7: Cross section of a circular vacuum chamber with a centered particle and an offset particle [6]

To remove the dependence on the beam charge from the signals generated by BPM electrodes, they are typically analysed in pairs. The most common way to process BPM outputs is the so-called Delta-Sigma normalization in which the sensitivity S_x[5] and a known offset of the position of the charge in one plane can be calculated according to:

$$\begin{aligned} x &= \frac{1}{S_x} \cdot \frac{U_1 - U_3}{U_1 + U_3} + \delta_x \\ y &= \frac{1}{S_y} \cdot \frac{U_2 - U_4}{U_2 + U_4} + \delta_y \end{aligned} \tag{1.1}$$

The voltages U_1 and U_3 are measured by opposing electrodes that get the x-plane position. The same is done for the y-plane position.

For a beam close to the mechanical center of the vacuum chamber applying Delta-Sigma normalization to the button BPM signals gives precise results. However, if the beam moves far away from the center, the signals of the opposing electrodes change non-linearly. This behaviour can be seen in figure 1.8. A special correction algorithm is necessary to deal with this non-linearity.

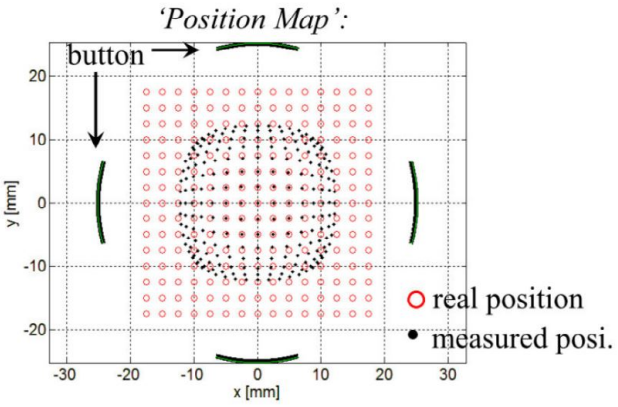


Figure 1.8: Position Map of a Button-BPM [6]

[5] This is the proportionality factor between the beam displacement and the signal strength. It is defined by the geometry of a given pick-up.

Stripline BPM

A stripline BPM has a structure similar to a button BPM. It consists of two or four stripline-like electrodes of length l and width w. These striplines are transmission lines with characteristic impedance Z_0, which usually corresponds to $50\,\Omega$. Electrically, each stripline electrodes and the particle beam form a 4-port network, similar to a directional coupler. However, for a stripline BPM, two ports correspond to the particle beam itself. Figure 1.9 explains the basic functionality of a stripline BPM in the time domain. It is assumed, that the signal in the stripline propagates at the same velocity as the particle beam [6]. The beam travels from left to right in the top part of the image.
At the time $t = 0$ a positively charged beam passes the upstream port of the electrode. This launches a pulse-like signal on the electrode. The intensity of the pulse is proportional to the beam position in the beam-pipe. Since the induced signal sees a transmission line, with a characteristic impedance of $Z_0 = 50\,\Omega$ in both directions, it splits into two parts of equal amplitude and travels in both directions, along the electrode towards the downstream port and through the upstream port.
At $t = \frac{l}{c_0}$ the beam passes the downstream port and induces a negative pulse in the stripline electrode. This pulse splits up as well and travels in both directions. At the same time the induced positive pulse from the upstream port arrives. Since the amplitudes of the induced pulses are equal, positive and negative pulse annihilate each other in the load impedance Z_0 at the downstream port. What remains is one positive and one negative pulse.
At the time $t = 2\frac{l}{c_0}$ the negative pulse leaves the upstream port and follows the positive pulse on the transmission line. The two pulses of different sign and separated by $2\frac{l}{c_0}$ can be detected.
This property of the stripline BPM is commonly referred to as directivity. Compared to the button BPM the stripline BPM is more expensive, complicated and fragile [6].

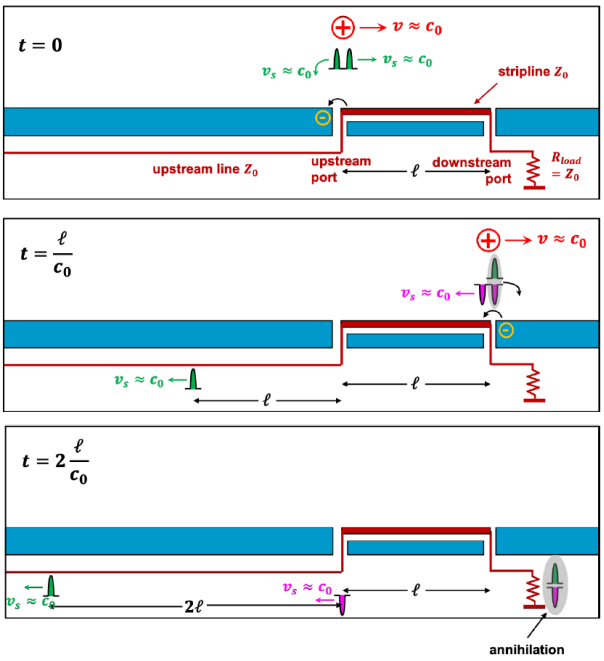


Figure 1.9: Operational principle of a stripline BPM [6]

Chapter 2

Project Description

The objective of the project described in this thesis is to upgrade an already existing test bench. Before installing a BPM in an accelerator, one needs to fully know and understand its properties and characteristics. Therefore a stretched-wire test bench was developed in the BP section of the Beam Instrumentation Group at CERN. The test bench makes it possible to measure a PU in the laboratory and determine its characteristics. One of the principle measurements performed is the determination of the offset between the mechanical center, where each electrode has an equal distance to a point in space, and the electrical center, where the received signal on each electrode is equal. These two centers can differ due to manufacturing tolerances of the button electrodes. As the measurement instruments used previously in the test bench had some limitations, a new platform is to be developed that generates a calibration signal, acquires and processes the measurements.

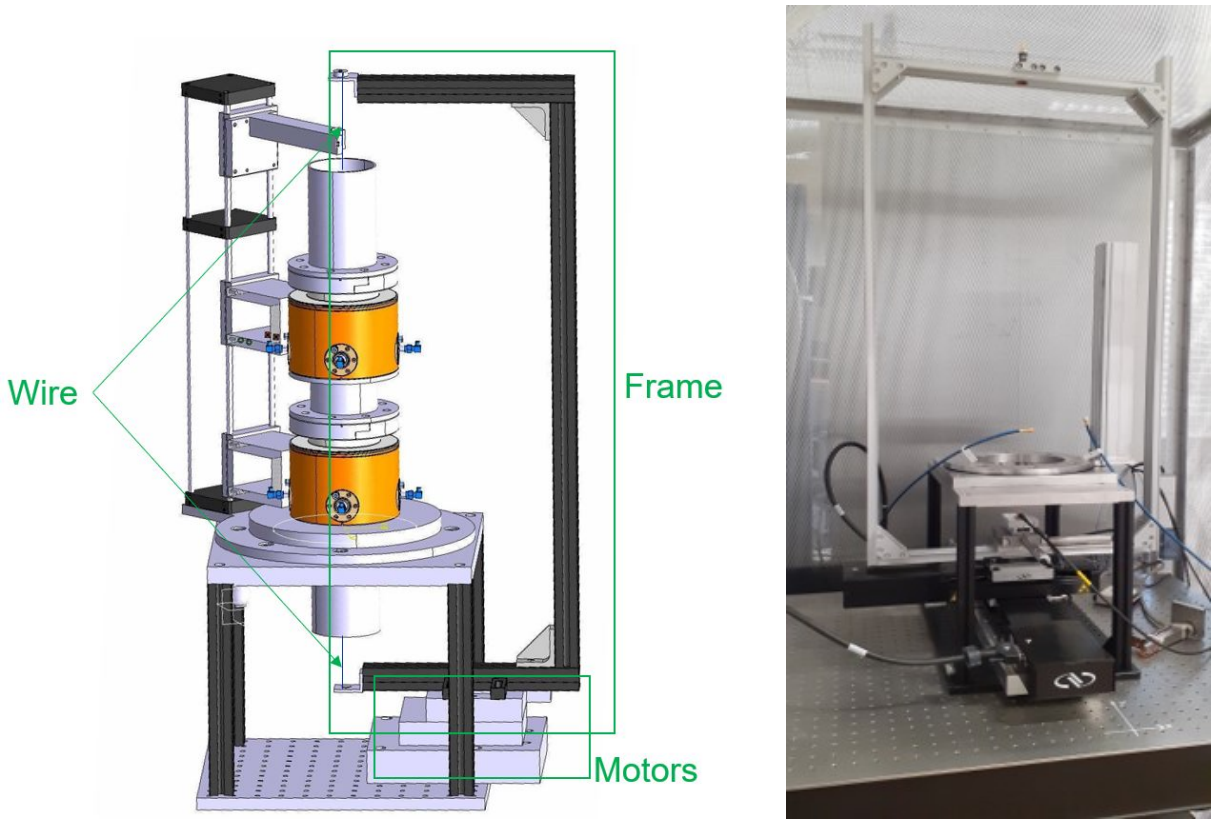
2.1 Current stretched-wire test bench

To observe the transverse position of the beam inside an accelerator one makes use of the principle that an electrode closer to the beam receives more signal than the one further away. The stretched-wire test bench emulates the beam by threading a wire through the BPM and sending an electric signal on it. Since the wire will behave as an antenna, it will emit electromagnetic waves that can be received by the BPM electrodes. The current test bench has been built by Franck Guillot-Vignot and consists of three different parts:

- Mechanical structure
- Readout electronics
- Controls software developed in LabView

The mechanical structure is built out of a frame that holds the two endpoints of the stretched wire. This frame is mounted on a movable platform, which can be moved around with a precision of $\pm 0.6\,\mu m$. Additionally, there is a fixed platform on which the BPM under test is mounted. The motors displacing

the wire frame are controlled by the LabView software. The entire mechanical structure can be seen in figure 2.1. The left-hand side shows a 3D model of the test bench, while the right-hand side shows a photograph of the actual structure. Additionally, the test bench is installed on a seismically-isolated optical table to mechanically decouple it from floor vibrations and is surrounded by a Faraday cage in order to reduce its susceptibility to any external electronics noise.



(a) 3D model of test bench

(b) Actual test bench

Figure 2.1: Structure of stretched-wire test bench

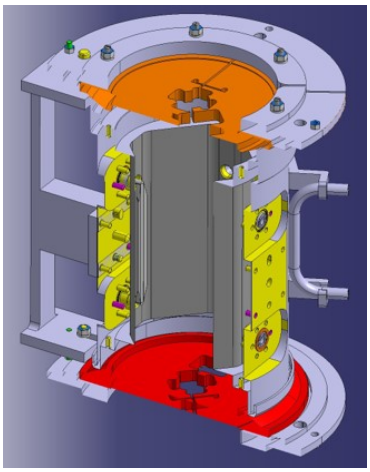
In the originally used setup, the power received by BPM electrodes was measured for two electrodes at a time with a Vector-Network-Analyzer (VNA) which also generated an excitation signal at a set frequency to the wire. It is worth mentioning, that this measurement can be performed with a narrow-band excitation frequency since the sensitivity of a BPM measuring relativistic beams is independent of the excitation frequency. The VNA computed the s_{21} parameter and forwarded it to the control software. Since the used VNA for the data acquisition can only measure a pair of electrodes at a time, measurements for BPMs with 4 or more electrodes were time-consuming since cables had to be switched as well as calibration and the measuring process had to be repeated. This increased the time during which the operator is actively occupied with the test bench.
The control software was developed in LabView and is currently being updated to make it more user-friendly. The following functionalities are present in the current version of the software. The main program manages five sub-categories:

- Capacitive tool management
- Manage PU

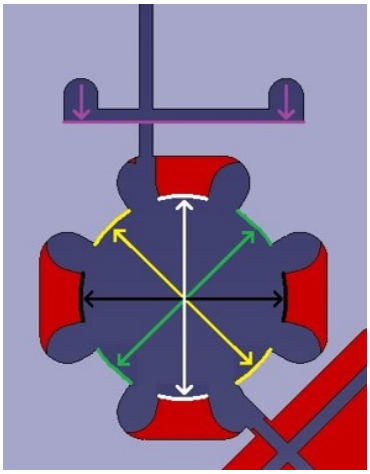
- Calibration software
- Linearity measurement
- Create measurement positions

All of them fulfill different tasks in the measuring process.

Capacitive tool management is used to align the wire with the mechanical center of the BPM. The capacitive tool is placed at the top and bottom opening of the PU and the wire frame is moved until an equal capacitance is measured against each opposing surface. The bottom tool is rotated by 45° compared to the top tool. The movement of the wire recorded by the motors can then be used to calculate the centre position. Figure 2.2 shows a close up of the tool and a pair of them installed in the test bench.



(a) Capacitive tool attached to a PU



(b) Two capacitive tools seen from above

Figure 2.2: Capacitive tool to find the center position

Manage PU lets the user create and manage different PUs. The user-defined parameters include shape, size, measurement frequency, amplifier settings, and others.

Calibration software manages the calibration of the entire test bench such as measuring and compensating the individual cable losses. It also adds the possibility of using an optical fork instead of a capacitive tool to align the wire in the geometrical center of the bench.

Linearity measurement drives the wire to different positions and performs measurements. The measured and calculated values (Delta-Sigma Normalization) are stored in a CSV file and plotted by the GUI of the software. Figure 2.3 illustrates the plots.

Create measurement positions allows the user to create a pattern of positions to which the wire will be displaced for measurements by the VNA. The position vector can have different forms as shown in figure 2.4.

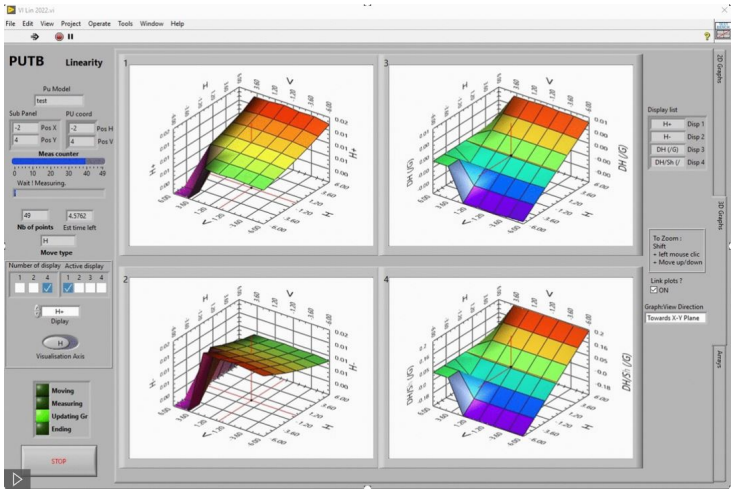
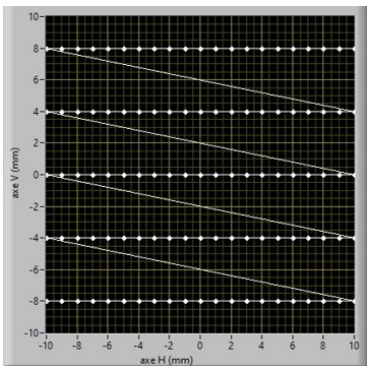
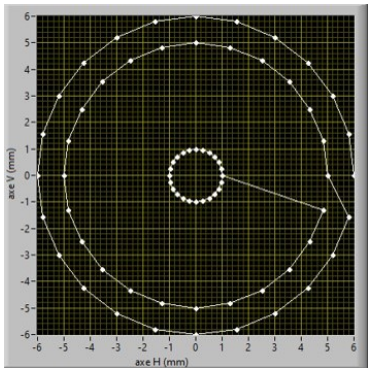


Figure 2.3: Screenshot of the LabView program while measuring a PU



(a) Linear Pattern



(b) Circular Pattern

Figure 2.4: Position Patterns

2.2 Goal of the Project

The test bench has shown some limitations originating from the used VNA as discussed in section 2.1. Therefore, a new platform is to be developed that generates the excitation signal as well as acquires and processes the signals generated by the PU electrodes. The output is to be sent to the LabView program. The project is based on the RFSoC ZCU111 platform by Xilinx that was already tested at CERN as a proof-of-principle data acquisition platform for future LHC BPMs. The FPGA of the ZCU111 can drive an internal DAC with a sinusoidal signal of variable frequency to be sent to the test bench wire as well as acquire and process data from up to eight electrodes simultaneously through the internal ADCs. Furthermore, a suitable algorithm to calculate the power and phase of each signal is to be implemented. The processor of the ZCU111 can use a USB or Ethernet connection to communicate with the LabView software running on an external PC. The board should be configured as a measuring instrument and exchange data as well as control commands using the SCPI standard.
The project should be an upgrade for the previously existing stretched-wire test bench and allow the operator to measure multiple planes of a BPM at the same time. This will result in a major time saving for BPMs with 4 or more electrodes, at least halving the measurement time.

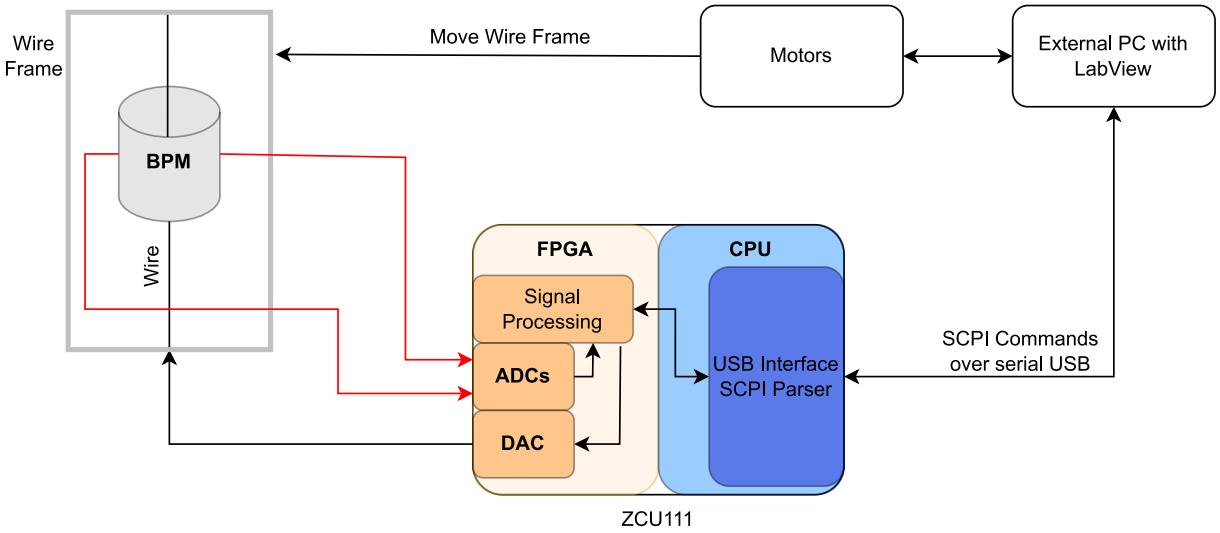


Figure 2.5: Principle structure of project description

A drawing of the principle structure has been created to make the project description clearer and to enforce a clear distribution of tasks which have to be implemented on the ZCU111. The structure is illustrated in figure 2.5.

Chapter 3

Xilinx RFSoC ZCU111

3.1 General Information

The Xilinx Multi-Processor System On Chip (MPSoC) is an evolution of the Xilinx Zynq SoC. It consists of a Processing System (PS) and a Programmable Logic (PL) with the PL being functionally equivalent to an FPGA. Figure 3.1 shows a high-level comparison of the three device types with the MPSoC being the most sophisticated of them [7].

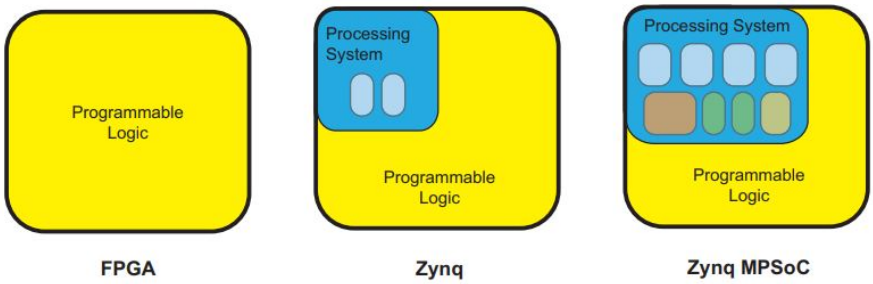


Figure 3.1: High Level Comparison of FPGA, Zynq and Zynq MPSoC [7]

Details of the Xilinx Zynq will not be discussed in the present thesis which will instead focus on the Zynq UltraScale+ MPSoC and Zynq UltraScale+ RFSoC.
The first MPSoC was announced in 2015 after Zynq devices had been widely adapted across the industry. The embedded FPGA and processor solution had been well-established and the need for an enhanced PS and a larger PL grew.

The rationale for coupling a PS with PL is to provide a dedicated, optimised resource for running the software-based components of an embedded system, in particular operating systems and software applications, while retaining all of the benefits and flexibility of an FPGA (in particular, its great parallelism and reconfigurability) [7].
Figure 3.2 shows the ZCU111, a Zynq UltraScale+ RFSoC evaluation board with its many connectors

and components. The following sections will go into detail about the functionalities and capabilities of the PS, PL and RF data converters. The difference between a MPSoC and RFSoC is the addition of high-speed RF data converters that can be accessed by the PL.

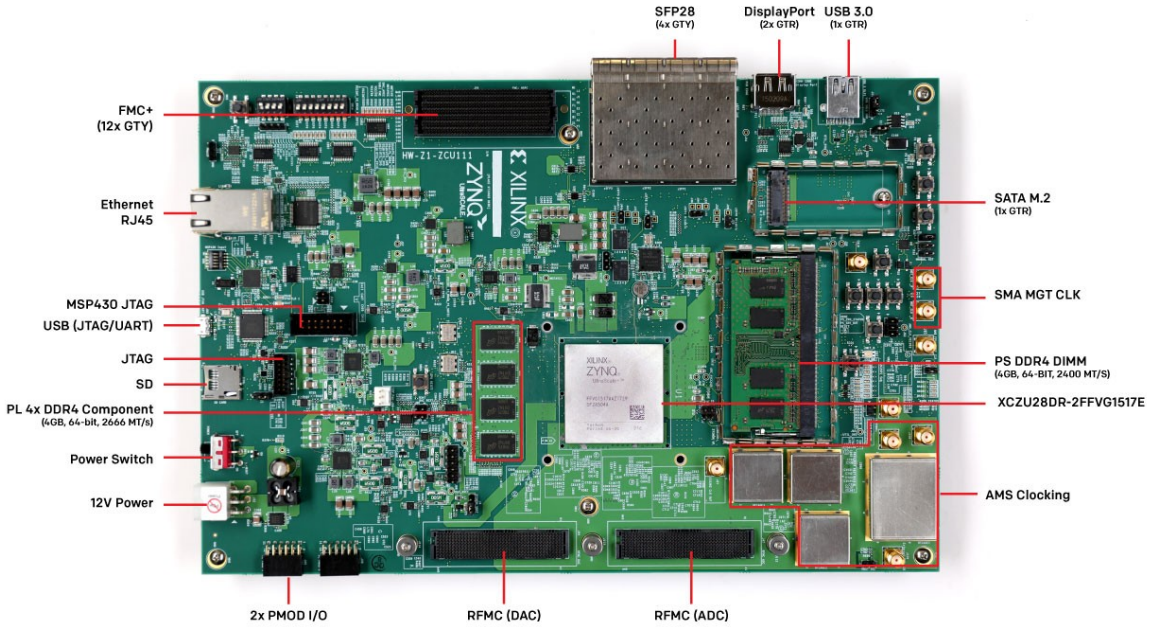


Figure 3.2: Xilinx Ultrascale+ RFSoC ZCU111

3.2 Processing System

The PS architecture of the ZCU111 consists of two different processing systems, the Application Processing Unit (APU) and the Real-Time Processing Unit (RPU). Both include additional interfaces, controllers and memories which will be briefly explained. The PS is attached to a 4 GB of Random-Access Memory (RAM).

Application Processing Unit

The APU houses four Arm Cortex-A53 Multi-Processing Cores. Each of them contains the following computational units: Floating Point Unit (FPU), NEON Media Processing Engine (MPE) and Crypto Extensions, Memory Management Unit, dedicated L1 cache, shared L2 cache and a Snoop Control Unit. A simplified version of the APU architecture can be seen in figure 3.3.

The FPU and the NEON MPE are responsible for computational tasks. While the FPU is designed to perform floating point arithmetic, the NEON instruction set accelerates signal processing algorithms and functions to speed up e.g. deep learning algorithms for which the ZCU111 is also commonly used for.

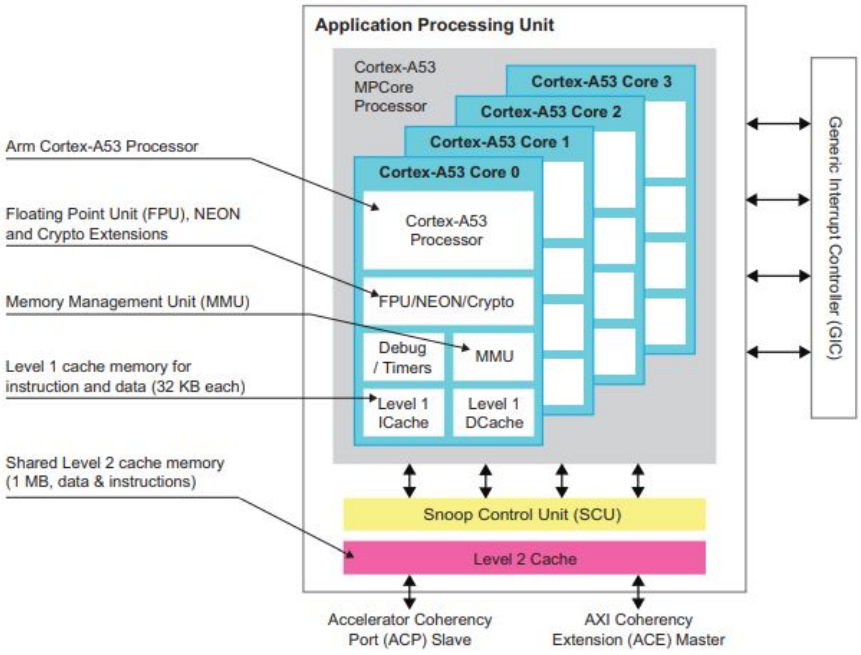


Figure 3.3: Simplified Block Diagram of the APU [7]

The memory management Unit is used to translate addresses between the virtual and physical address spaces.
Level 1 cache memory is separate for each core, so that they can locally store data and instructions to retrieve them quickly. The L2 cache is shared between the cores for additional storage space for instructions or data. Since it is shared it can also be used by the cores to communicate between each other without the use of external memory.
The Snoop Control Unit manages the access to the L2 cache and manages the transactions between the APU and the PL over the Accelerator Coherency Port.
The Crypto Extension is a dedicated encryption and decryption tool for well know cryptographic standards like Advanced Encryption Standard or Secure-Hash algorithms [7].

Real-Time Processing Unit

The RPU contains two Arm Cortex-R5 real-time processor cores for real-time applications. It provides low latency and deterministic performance. Figure 3.4 illustrates a simplified structure of the RPU.

A single Cortex-R5 core contains, similarly to the APU, a Floating Point Unit (FPU) for single and double precision arithmetic. The Level 1 memory of the RPUs includes three Tightly Coupled Memories that provide predictable instruction execution and predictable data/store timing in the RPU processors. This is needed to ensure instructions are executed on schedule and makes the RPU processing deterministic. Furthermore, two 32 KB caches are implemented for local storage of data and instructions to improve processing performance as well as a Memory Protection Unit to manage access requests to the L1 memory. In principle, the RPU can be used in two different configuration modes of the dual-core Cortex-R5, Split-mode and Lock-step mode. In the Split-mode, each core works independently which allows the RPU to

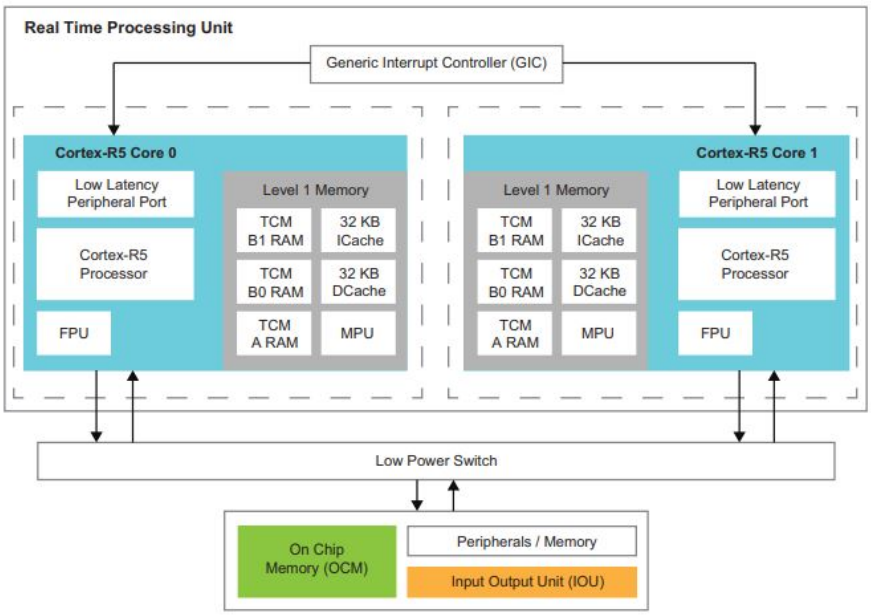


Figure 3.4: Simplified Block Diagram of the RPU [7]

work at its maximum performance. The Lock-step mode, on the other hand, lets both cores perform exactly the same tasks. This provides redundancy since the output of both cores should be exactly the same and allows errors to be detected. The Lock-step mode is also called a safety mode as it makes detection of errors efficient [7].

3.3 Programmable Logic

The PL is a crucial component of the Zynq UltraScale+ ZCU111, as it provides hardware acceleration for computationally intensive arithmetic. It is implemented using a 16 nm Kintex Ultrascale+ FPGA fabric consisting of Slices and Configurable Logic Blocks. Each of these blocks is positioned next to a switch matrix so that it can route data to other PL blocks. The different elements and the structure of the PL can be seen in figure 3.5. An individual Configurable Logic Blocks is composed of a slice that holds all the sequential and combinatorial logic circuits. Each slice is composed out of 8 6-input Look Up Tables (LUTs), 16 Flip-Flops and additional routing resources. Adjacent slices can be connected together via carry logic to build large arithmetic circuits. Figure 3.6 illustrates the composition of a Configurable Logic Block and its components [7].
The RFSoCs PL side is equipped with a 4 GB RAM. These 4 GB are organised into Block RAM tiles, that can be configured as conventional RAM, Read-only Memory (ROM) and First In First Out (FIFO) buffers. Another memory element inside the PL are the UltraRAM tiles. These are possible replacements for external memories such as static RAM and offer a larger memory space in a single tile than a BlockRAM. For fast arithmetic operations, dedicated Digital Signal Processing (DSP) slices are implemented within the PL to meet tight timing constraints. These DSP48E2 slices offer high-speed hardware multipliers for efficient signal processing.

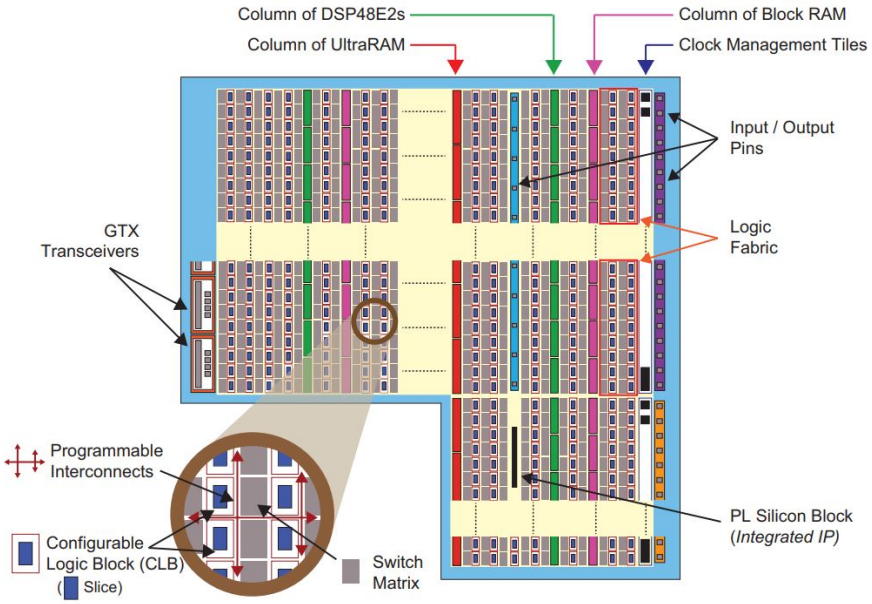


Figure 3.5: Zynq MPSoC Programmable Logic [7]

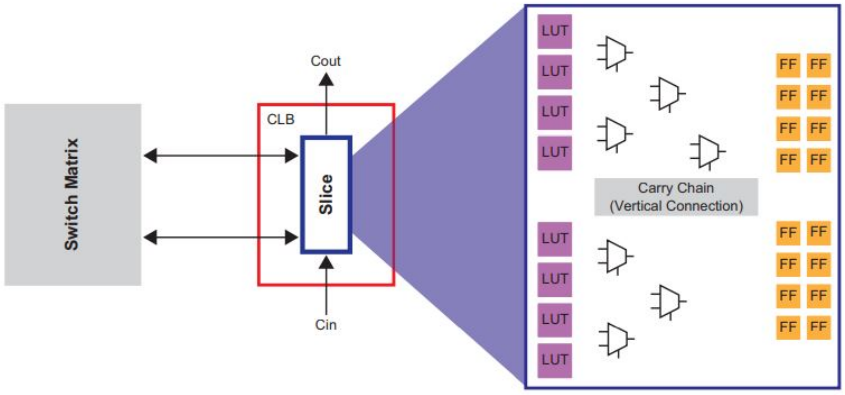


Figure 3.6: Configurable Logic Block with connections and internal components [7]

3.4 Processing System and Programmable Logic Interfaces

Since the RFSoC offers software and hardware co-processing in a single chip, there need to be dedicated hardware interfaces that drive the communication and transfer of data and instructions between the PS and PL. The basis of most of these interfaces is the ARM Advanced Microcontroller Bus Architecture (AMBA) open standard. There are six interfaces, extensions and buses within the AMBA of which five are featured in the RFSoC device.
The most interesting one for this project is the Advanced eXtensible Interface (AXI) standard, used in embedded systems to connect processing elements with IP[1] blocks whilst providing high bandwidth and low latency communication [7].
Since the RFSoC uses the fourth release of the protocol it will be referred to as AXI4. Within the AXI4 standard there are three protocols that have different advantages and use cases.

- **AXI4** – the most sophisticated of the three protocols which is used for memory-mapped[2] connections between processing elements and IP blocks. It can perform single-beat as well as burst transfers and has dedicated channel resources for transmitting its data. The AXI4 protocol is suitable for sending large quantities of information from or to main memory.
- **AXI-Lite** – a simple memory-mapped protocol with reduced handshake signals, resulting in fewer resource allocations than the full AXI4 protocol. Since only single-beat transfers are allowed, this protocol is used mainly for low bandwidth communication with IP blocks and control registers.
- **AXI4-Stream** – direct point-to-point data transfers of unrestricted size. It is made for sending large quantities of data between a source and destination in a device therefore no address channel is required.

3.5 RF data converters

Since the DACs and ADCs play an important role in this project for generating and measuring RF signals it is important to look at their characteristics in more detail.
The ZCU111 hosts 8 12-bit ADCs sampling at up to 4.096 GSPS and 8 14-bit DACs sampling at up to 6.554 GSPS. The manufacturer's datasheet „DC and AC Switching Characteristics (DS926)“ lists common parameters for the ADCs and DACs, like the Adjacent Channel Leakage Ratio (ACLR), Noise-Spectral Density (NSD) and Spurious Free Dynamic Range (SFDR). These parameters are shown in tables 3.1 and 3.2, respectively.
The DACs can be used in two different output modes, 20 mA and 32 mA mode. Both modes provide a different maximum output power of the DAC. In the 20 mA mode, the maximum output power is limited to 1 dBm into 100 Ω which is the same as the maximum allowed input power of the ADCs.
Both types of data-converters have a 100 Ω differential termination on the chip. The converters are interfaced through a custom add-on card developed by the Beam Instrumentation Group at CERN. The card matches the differential 100 Ω input of each data converter to a single-ended 50 Ω channel. The card uses specialized RF-FPGA Mezzanine Card (FMC) on the ZCU111 side and more common SMA connectors on the user side.

Since parameters like Signal-to-Noise Ratio (SNR) and Effective Number of Bits (ENOB) take the whole Nyquist bandwidth of a data converter into account, they are not that commonly used for converters

[1] An Intellectual Property (IP) core is a functional module that is implemented in the PL
[2] „refers to a protocol that issues an address, together with the specified transaction. The data is either stored or read from memory using the given address, which can only be provided by the master“ [7]

Table 3.1: ZCU111 ADC parameters

Parameter	Info	typical	worst
ACLR	64QAM with F_{in}= 3.5 GHz and B=18 MHz	−63 dBc	-
NSD	NSD across the first Nyquist zone F_{in}=240 MHz @ −1 dBFs	−150 dBFs/Hz	−147 dBFs/Hz
SFDR	Excluding second and third harmonic with F_{in}=240 MHz @ −1 dBFs	83 dBc	77 dBc

Table 3.2: ZCU111 DAC parameters

Parameter	Comment	typical	worst
ACLR	F_c=240 MHz, single 3.84 MHz QPSK	−82 dBc	−78 dBc
NSD	F_{out}=240 MHz CW at 0 dBFs	−158 dBFs/Hz	−151 dBFs/Hz
SFDR	F_{out}=240 MHz CW at 0 dBFs	84 dBc	80 dBc

in direct sampling applications where most of the time only a small bandwidth is of interest. These parameters are usually used for conventional send/receive architectures where the mixing happens in the analog domain and only slow-sampling data converters are needed.
Nevertheless, these parameters can be calculated using the Noise-Spectral Density (NSD) of the data converters to get a better understanding of the components.

The SNR is defined as:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \cdot \log\left(\frac{P_{\text{Fundamental signal}}}{P_{\text{Noise over Nyquist bandwidth}}}\right) \tag{3.1}$$

The noise over a certain bandwidth can be described as the NSD integrated over the same bandwidth, hence over the whole Nyquist bandwidth for a sampling frequency f_s the SNR normalized to the full scale (FS) is:

$$\mathrm{SNR}_{\mathrm{dBFs}} = P_{\text{Fundamental signal (dBFs)}} - (\mathrm{NSD}_{\mathrm{dBFs/Hz}} + 10 \cdot \log\left(\frac{f_s}{2})_{\mathrm{Hz}}\right) \tag{3.2}$$

Assuming that the input signal covers the entire full scale, i.e. $P_{\text{Fundamental signal}} = 0\mathrm{dBFs}$, the equation simplifies to:

$$\mathrm{SNR}_{\mathrm{dBFs}} = -\mathrm{NSD}_{\mathrm{dBFs/Hz}} - 10 \cdot \log\left(\frac{f_s}{2}\right)_{\mathrm{Hz}} \tag{3.3}$$

Alternatively, the SNR for a data converter over the whole Nyquist bandwidth can be calculated as:

$$\mathrm{SNR} = \mathrm{ENOB} \cdot 6.02\,\mathrm{dB} + 1.76\,\mathrm{dB} \tag{3.4}$$

Equation 3.3 and 3.4 can be combined to calculate the ENOB:

$$\mathrm{ENOB} = \frac{-\mathrm{NSD}_{\mathrm{dBFs/Hz}} - 10 \cdot \log\left(\frac{f_s}{2}\right)_{\mathrm{Hz}} - 1.76\,\mathrm{dB}}{6.02\,\mathrm{dB}} \tag{3.5}$$

The NSD can be extracted from the tables 3.1 and 3.2 for the ADC and DAC, respectively. For the calculations, sampling frequencies of 4.096 GSPS for the DAC and 2.048 GSPS for the ADC are used. This leads to $\mathrm{ENOB_{ADC}} \approx 9.66$ and $\mathrm{ENOB_{DAC}} \approx 10.49$ over the whole Nyquist bandwidth.
Figure 3.7 illustrates the relationship between SNR and NSD for an ideal 12-bit ADC at $f_s = 4$ GSPS. For narrowband applications, such as the project described in this thesis. It is, therefore, useful to consider the NSD to compare different RF-sampling data converters with different sampling frequencies to evaluate which device has the lowest frequency-band specific noise [8].

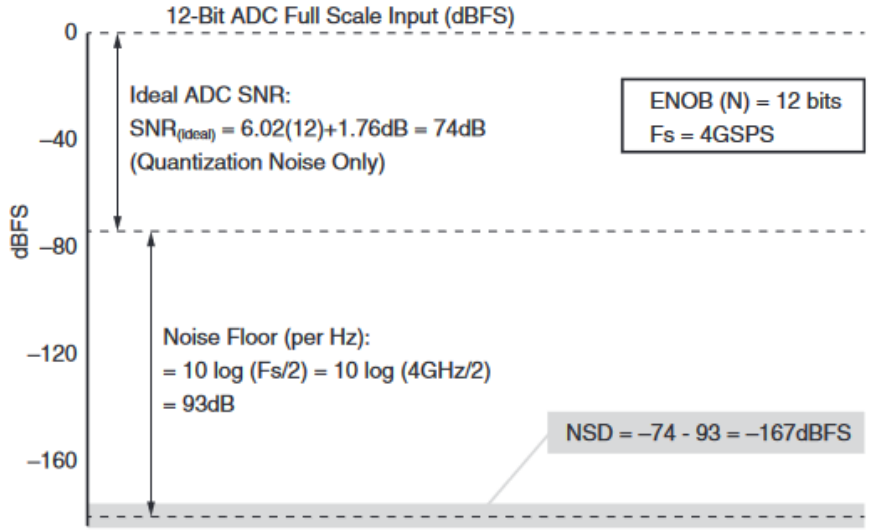


Figure 3.7: Comparison of the SNR and NSD over a wide frequency spectrum [8]

Chapter 4

Theoretical Background

4.1 Cordic Algorithm

The Coordinate Rotational Digital Computer (CORDIC) is a hardware-efficient iterative method that uses rotations to calculate a wide range of elementary functions.
Suppose a point in a coordinate system is (x_0, y_0) and rotate it by an arbitrary angle θ. A new point (x_1, y_1) is obtained, the coordinates of which can be calculated as:

$$\begin{aligned} x_1 &= x_0 \cdot \cos(\theta) - y_0 \cdot \sin(\theta) \\ y_1 &= x_0 \cdot \sin(\theta) + y_0 \cdot \cos(\theta) \end{aligned} \tag{4.1}$$

For $x_0 = 1$ and $y_0 = 0$ the rotated point is:

$$\begin{aligned} x_1 &= \cos(\theta) \\ y_1 &= \sin(\theta) \end{aligned} \tag{4.2}$$

This proves that the trigonometric functions $\sin(\theta)$ and $\cos(\theta)$ can be calculated by simply rotating a vector (x_0, y_0).
The CORDIC algorithm provides a hardware-efficient method to calculate those trigonometric functions. The algorithm is considered hardware efficient as it does not use multipliers and relies only on additions/subtractions and shift registers for multiplication and division.
Equation 4.1 can be rewritten as:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = cos(\theta) \begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{4.3}$$

with the number of multiplications remaining the same as before.
The CORDIC algorithm exploits two fundamental ideas to achieve a rotation without any multiplications. Firstly, a rotation by an arbitrary angle θ_d is equal to the sum of several smaller rotations by an angle of θ_i with $i = 0, 1, \cdots n$. Furthermore, the smaller angles θ_i are chosen such that $\tan(\theta_i) = 2^{-i}$ for $i = 0, 1, \cdots n$. Consequently, the multiplications by $\tan(\theta_i)$ in 4.3 are replaced by a bitwise shift which is much faster than a regular multiplier.
Satisfying both assumptions simultaneously ($\theta_d = \sum_{i=0}^{n} \theta_i$ while $tan(\theta_i) = 2^{-i}$) becomes possible with

the increase of n as the equations converge and the accuracy of the rotation is increased.
The remaining multiplication by $\cos(\theta)$ in equation 4.3 acts as a scaling factor for the rotation. The rotation by an angle θ will be correct, but the values (x_1, y_1) will still be wrong by a known factor.
The possible rotations for $\tan(\theta_i) = 2^{-i}$ up to $i = 5$ are shown in table 4.1.

Table 4.1: Angles of $\arctan(2^{-i})$

i	$\arctan(2^{-i})$
0	45°
1	26.565°
2	14.036°
3	7.125°
4	3.576°
5	1.79°

Since the process is iterative, equation 4.3 can be used multiple times. For example, a rotation by $\theta = 32.471°$ can be achieved in three steps as $32.471° = 45° - 26.565° + 14.036°$:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \cos(45°)\cos(-26.565°)\cos(14.036°) \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2^{-1} \\ -2^{-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{4.4}$$

The scaling factor can be ignored for the intermediate steps and applied once at the end. Since the rotational angles get closer and closer to zero, $\cos(\theta_i)$ will tend towards unity. Moreover, the rotation direction of an intermediate step does not affect the scaling factor since $\cos(-\theta) = \cos(\theta)$. Therefore it converges into a single constant:

$$K = \Pi_{i=0}^{n} cos(\theta_i) = 0.6072 \quad \text{where} \quad \theta_i = \arctan(2^{-i}) \tag{4.5}$$

Neglecting the scaling factor K, the following equations describe the CORDIC algorithm:

$$\begin{aligned} x[i+1] &= x[i] - \sigma_i 2^{-i} y[i] \\ y[i+1] &= y[i] + \sigma_i 2^{-i} x[i] \end{aligned} \tag{4.6}$$

where $\sigma_i \in \{+1, -1\}$ determines the direction of rotation which will be discussed now.
At each iteration step i, the overall achieved so far must be compared to the final required rotation. Let's consider a desired rotation angle of 60°. Since the starting angle is 0°, it follows that $\sigma_0 = 1$ to rotate counter-clockwise. The second iteration step starts at an angle of 45° which is smaller than 60° so another counter-clockwise rotation has to be performed ($\sigma_1 = 1$). At the next step, the accumulated rotation angle is $45° + 26.565° = 71.565°$ which is greater than 60°. Therefore, the rotation must be clockwise and $\sigma_2 = -1$. This procedure continues until n iteration steps are completed. Equation 4.6 can be modified with an extra expression $z[i]$ that keeps track of the current rotation.

$$\begin{aligned} x[i+1] &= x[i] - \sigma_i 2^{-i} y[i] \\ y[i+1] &= y[i] + \sigma_i 2^{-i} x[i] \\ z[i+1] &= z[i] - \sigma_i \arctan(2^{-i}) \end{aligned} \tag{4.7}$$

where $z[0]$ is initialised with the desired final rotation angle. The sign of $z[i+1]$ decides the direction of the next rotation.
The CORDIC algorithm requires only n values of $\arctan(2^{-i})$ to be stored in a LUT. Figure 4.1 shows an implementation of the CORDIC algorithm in hardware for one iteration [9].

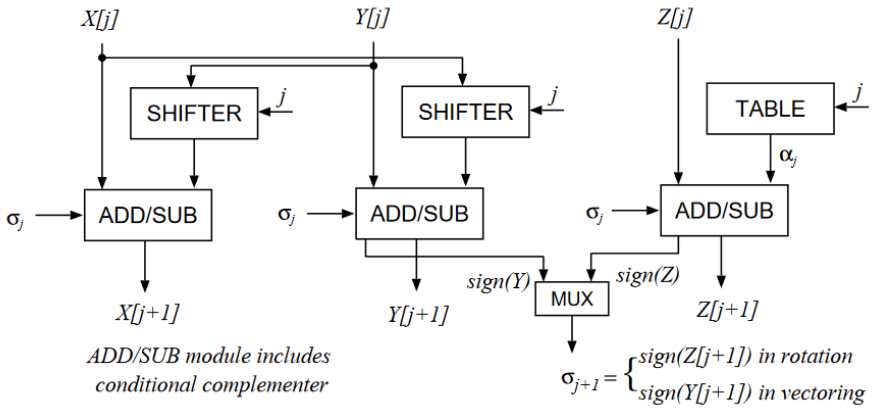


Figure 4.1: CORDIC implementation in hardware for one iteration [9]

An important limitation of the CORDIC algorithm is that it cannot rotate by arbitrary angles. The maximum possible rotation is given by:

$$\sum_{i=0}^{\infty} \arctan(2^{-i}) \approx 99.88^{\circ} \tag{4.8}$$

Therefore, not all sine and cosine values for a fixed starting position of $(x, y) = (1, 0)$ can be calculated. However, working with greater angles is possible when the initial values (x, y) are adapted accordingly. For examples, $\sin(135^{\circ})$ can be calculated using the starting position of $(x, y) = (-1, 0)$ and a rotation by 45°.

4.2 Goertzel Algorithm

The power of a sine wave of a certain frequency can be calculated from the Discrete Fourier Transform (DFT). Since for the project described in this thesis, the power at only a single frequency is of interest, the whole DFT does not need to be calculated. It is sufficient to compute only a single DFT bin.
The algorithm used to fulfill this need is called Goertzel-Algorithm or Goertzel-Filter. It has been used in many applications before, such as Frequency-Shift-Keying (FSK) demodulation. The algorithm reduces the number of necessary multiplications compared to a Fast Fourier Transform (FFT) and is therefore particularly useful when only a few frequencies must be computed. Goertzel's algorithm can be derived by converting the DFT equations into an equivalent form as a convolution, which can be implemented as a digital filter [10].

The DFT equation can be rewritten as follows:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}} \\ &= \sum_{n=0}^{N-1} x[n] e^{j\frac{2\pi kN}{N}} e^{-j\frac{2\pi nk}{N}} \\ &= \sum_{n=0}^{N-1} x[n] e^{j\frac{2\pi((N-n)k)}{N}} \\ &= \Big(\big((x[0]e^{j\frac{2\pi k}{N}} + x[1])e^{j\frac{2\pi k}{N}} + x[2]\big) + \cdots + x[N-1]\Big) e^{j\frac{2\pi k}{N}} \end{aligned} \tag{4.9}$$

In the last equation, a recursive pattern can be recognized:

$$y[n] = e^{j\frac{2\pi k}{N}} y[n-1] + x[n] \tag{4.10}$$

where $y(-1) = 0$. This method is also called the first-order Goertzel Filter and the desired output $X[k]$ will be the Nth sample (since the input data stream $x[n]$ has only N-1 samples, the last sample will be $x[N] = 0$). As a complex multiplication is used for every sample, which makes it computationally demanding, the equation can be modified further by transformation into the z-Domain. $H(z)$ represents the z-Transform of equation 4.10:

$$H(z) = \frac{1}{1 - e^{j\frac{2\pi k}{N}} z^{-1}} \tag{4.11}$$

Multiplying the nominator and denominator by $1 - e^{-j\frac{2\pi k}{N}} z^{-1}$ yields:

$$H(z) = \frac{1}{1 - e^{j\frac{2\pi k}{N}} z^{-1}} = \frac{1 - e^{-j\frac{2\pi k}{N}} z^{-1}}{1 - ((e^{-j\frac{2\pi k}{N}} + e^{j\frac{2\pi k}{N}})z^{-1} - z^{-2})} = \frac{1 - e^{-j\frac{2\pi k}{N}} z^{-1}}{1 - (2\cos(\frac{2\pi k}{N})z^{-1} - z^{-2})} \tag{4.12}$$

The respective difference equation is:

$$y[n] = x[n] - x[n-1]e^{-j\frac{2\pi k}{N}} + 2\cos\left(\frac{2\pi k}{N}\right) y[n-1] - y[n-2] \tag{4.13}$$

with the starting values of $x[-1] = y[-1] = y[-2] = 0$. This system can be described using the state variables:

$$s[n] = x[n] + 2\cos\left(\frac{2\pi k}{N}\right) s[n-1] - s[n-2] \tag{4.14}$$

while the output is given by [11]:

$$y[n] = s[n] - e^{-j\frac{2\pi k}{N}} s[n-1] \tag{4.15}$$

From the transfer function $H(z)$ one can extract the zeroes and poles. The Goertzel Filter has a single zero at $z = e^{-j\frac{2\pi k}{N}}$ and conjugate poles at $z = e^{j\frac{2\pi k}{N}}$. The zero/pole pair at $z = e^{-j\frac{2\pi k}{N}}$ cancels each other out and leaves the transfer function with just one pole. The magnitude response shows a single resonance at the normalized frequency of $\frac{2\pi k}{N}$, which is expected [12].
The pole lying exactly on the unit circle affects the stability of the filter, especially in low-precision arithmetic. This issue can be addressed either by a variation of the Goertzel Filter, called the Reinsch Algorithm or by using sufficient precision to the calculation.

The constructed Infinite Impulse Response (IIR) filter structure is shown in Figure 4.2.

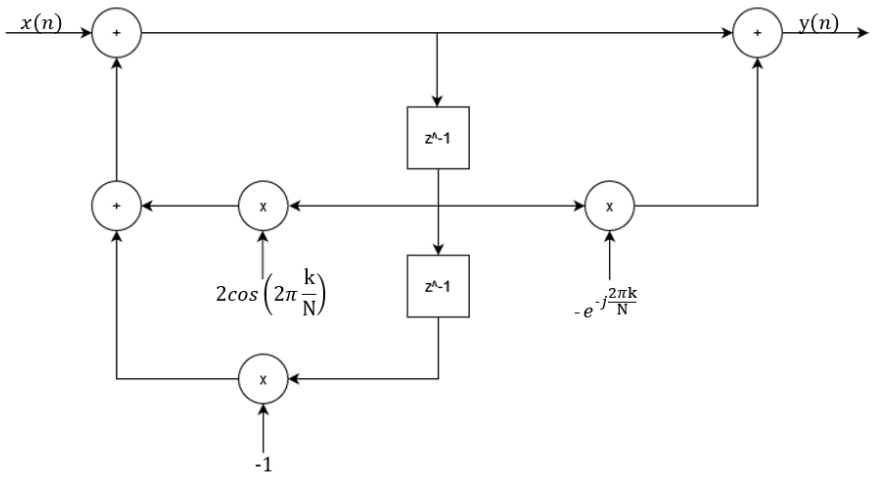


Figure 4.2: IIR Filter implementation of Goertzel Filter

4.3 Periodogram

The periodogram is an estimate for the power spectrum of a signal. It is part of non parameterised methods to estimate the spectral components of a signal and is based on the DFT.
It is known that the power spectral density of a signal can be calculated by Fourier transforming its autocorrelation function. The biased autocorrelation function of a sampled signal $x[n]$ is calculated according to:

$$\hat{l}^b_{xx}[k] = \frac{1}{N}\sum_{n=0}^{N-1} x^*[n]\cdot x[n+k] = \frac{1}{N}\sum_{n=0}^{N-1} x^*[n-k]\cdot x[n] \tag{4.16}$$

Equation 4.16 resembles a convolution. To reduce computational power a fast convolution method can be used to transform the signal into the Fourier domain and replace the convolution by a multiplication.

$$\begin{aligned}\hat{l}^b_{xx}[k] &= \frac{1}{N}x[k] * x^*[-k] \\ &= \frac{1}{N}\mathrm{IDFT}(X[n]\cdot X^*[n]) \\ &= \frac{1}{N}\mathrm{IDFT}(|X[n]|^2)\end{aligned} \tag{4.17}$$

The spectral power density can now be estimated by Fourier transforming the estimated autocorrelation function $\hat{l}^b_{xx}[k]$.

$$\begin{aligned}L^b_{xx}[n] &= \mathrm{DFT}(\hat{l}^b_{xx}[k]) \\ &= \frac{1}{N}\mathrm{DFT}(\mathrm{IDFT}(|X[n]|^2))\end{aligned} \tag{4.18}$$

This is known as the periodogram. In general, for time sampled signals, it takes the following form:

$$Per_{xx}[f] = \frac{1}{N}\cdot|X[f]|^2 = \frac{1}{N}\cdot X[f]\cdot X^*[f] \tag{4.19}$$

Using the common definition of the DFT, its values are calculated as:

$$X[n] = \sum_{k=0}^{N-1} x[k] e^{-j2\pi \frac{kn}{N}} \tag{4.20}$$

where an additional scaling factor N is present compared to the continuous Fourier transform. This scaling affects also the periodogram the values of which are scaled by a factor N compared to the power of the analog signal in the bandwidth between two DFT bins.
To compensate for this, the power calculation must be corrected:

$$P[f] = \frac{1}{N} Per_{xx}[n] = \frac{|X[f]|^2}{N^2} \tag{4.21}$$

The calculated periodogram returns the power contained within a whole frequency bin, rather than a single frequency, the width of which is equal to $\frac{f_s}{N}$ [13].

4.4 Standard Commands for Programmable Instruments

Standard Commands for Programmable Instruments (SCPI) is an instruction set for standardized communication with measurement instruments such as oscilloscopes or multimeters. It is commonly used to automate measurements with laboratory equipment.
SCPI is defined in the IEEE-488.2 specification „Standard Codes, Formats, Protocols, and Common Commands“ and provides a common syntax and format that can be used with all instruments. The commands are ASCII text strings which are divided into groups and subgroups. Commands are a single or a series of more keywords that are combined. A keyword is usually written in the format „AVERage“ and consists of an uppercase and lowercase part. The uppercase part is mandatory while the lowercase part of the keyboard is optional. The keywords can be combined with a colon, e.g. „AVERage:FACTor“. The command „AVERage:FACTor16“ would set the average factor to a value of 16. An addition of a question mark at the end of the command creates a query to read values from the instrument. The command „MEASure:VOLTage:DC?“ would return a measured DC voltage.
There are some commands which are mandatory to be implemented in a SCPI device such as the identification command „*IDN?“ which returns the manufacturer, serial number and further useful information about the device. These commands always start with an asterisk.
To detect the end of a command, usually the termination character „\n“ is concatenated at the end of the string.
Since SCPI specification defines only the syntax of the instructions, the transport protocol can be chosen arbitrarily. Originally it was created for GPIB communication, but nowadays is commonly used also with serial ports, Ethernet (over TCP/IP) and USB.
The current project implements 12 main commands with their respective sub-commands which provide all required functionalities. These commands are summarised in table 4.2. The commands are the same as for the VNA used in the original test bench setup. For that specific instrument, a command and sub-command is not separated by a „:“, therefore this is also not implemented in this project.
The implemented commands make it possible to perform a single measurement and read the data, as well as to perform multiple measurements and return the average values. Therefore the „MODE“command lets the operator decide between a continuous and triggered mode. Additionally, some limitations have to be set in the software, to only allow a maximum output frequency of 1 GHz, a maximum of 8 ADCs and averaging over not more than 1000 values.
With „FORMAT“one can set the output scale to linear/logarithmic with the phase in degrees or to a complex format in which real and imaginary part of the calculation is returned.

Table 4.2: List of implemented SCPI commands

SCPI Command	**Sub-command**	**Description**
*IDN?	;	Asks for the Identification of the instrument
PRES	;	Resets the device to a predefined state ->In order to work again some parameters have to be retransmitted
CHAN	x;	Selects the active channel x = {1,2,3,4,5,6,7,8}
	NB;	Sets the number of active ADCs (max. 8)
	NB?;	Asks how many ADCs are active
AVERFACT	x;	Sets the averaging factor for the active channel x=[1-1000]
	?;	Asks for the averaging factor for the active channel
AVERO	x;	Enables averaging for the active channel x = {ON, OFF}
	?;	Asks if averaging for the active channel is enabled
AVERREST	;	Resets the average for the active channel
FORMAT	LINM;	Sets the polar magnitude to linear format
	LOGM;	Sets the polar magnitude to log format
	COMP;	Sets the output format to complex
OUTPMARK	;	Outputs measurement for the active channel
FREQ	x;	Sets the frequency of operation in Hz max. 1000 MHz (x is float)
	?;	Asks for the set frequency of operation
WAITTIME?	;	Asks for the time Labview has to wait for the instrument to complete measurements Returns [ms]
MODE	CONT;	Sets the mode to continuous measurement
	TRIG;	Sets the mode to single-time measurement
MEAS	;	Perform a single measurement Read Values with "OUTPMARK;"
CAL	;	Calibrates the active channel with AVERFACT values for averaging
	?;	Asks if calibration on the active channel has been performed
	??;	Asks if calibration for all channels has been performed
RFOUT	?;	Asks if the output of the instrument is on or off
	ON;	Enables the output so that measurements can be performed
	OFF;	Disables the output to that no measurements can be performed

Chapter 5

Simple Python Simulation

Firstly, to get familiar with the project and its background, a small and simple python simulation was created. The simulation contains:

- A sampled sine wave generator, which simulates the DAC
- Amplifiers and attenuators
- Variable amplifiers/attenuators, which simulate the (x, y) position of the wire
- Amplifiers before the ADCs
- Power calculation using Periodogram or Welch's method
- Delta-Sigma Normalization

The python code for every function can be found in the appendix.

5.1 Components

5.1.1 Sine wave generator

The sine wave generator produces a sine wave of variable frequency and amplitude. In addition, the DAC quantises the signal so that it uses the quantisation levels of the real DAC. The maximum output voltage of the DAC in 20 mA mode is ± 0.5 V with an output power of 1 dBm. With a 14-bit resolution, the DAC has a quantisation noise of $\approx 61\,\mu$V. To increase simulation accuracy, noise giving SNR as defined in equation 3.3 will be added to represent the quantisation noise. The SNR for this DAC setup will therefore be 57.88 dB over the whole sampling bandwidth for the most conservative NSD.
Additional DAC parameters are the sampling rate, which is chosen to 4.096 GSPS, as well as the length of the sine wave in seconds.
Figure 5.1 shows the frequency domain output of the DAC for a 600 MHz sine wave that is 1 µs long with an output amplitude of 0 dBFs which results in 1 dBm output power on a 100 Ω termination.
Additionally, the signal is upsampled to represent an analog signal.

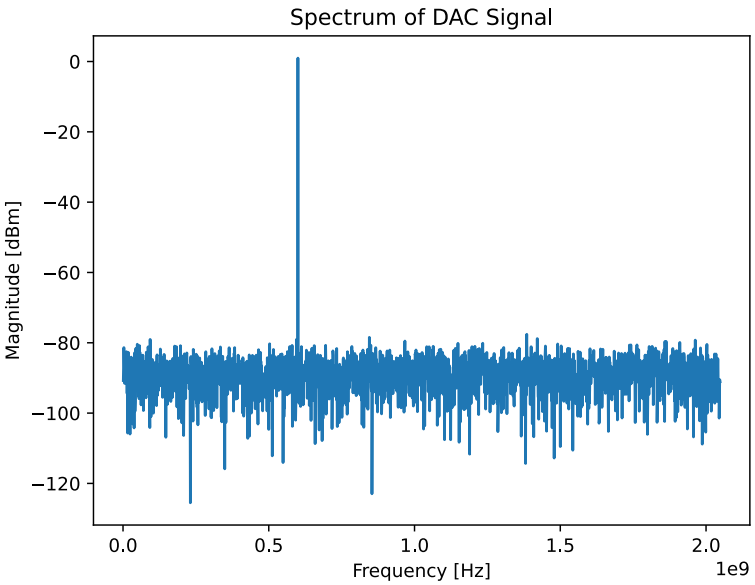


Figure 5.1: Spectrum of the DAC output

5.1.2 Amplifier and attenuator

The next component in the simulated chain is an amplifier that boosts the signal power. It was decided that its output peak voltage V_p should be 12 V. Since the DAC output peak voltage is 0.5 V, it needs to be amplified by a factor of 24 which translates to a 27.6 dB gain. As there is no ideal amplifier, noise is added to the signal. The added noise is equivalent to the „Equivalent input noise voltage $[\frac{\text{nV}}{\sqrt{\text{Hz}}}]$“ and was chosen to be $e_{in} = 10\frac{\text{nV}}{\sqrt{\text{Hz}}}$. The signal bandwidth was set to $BW = 1\,\text{MHz}$.
The produced noise can be calculated according to:

$$e_{out} = G_{in} \cdot e_{in} \cdot \sqrt{BW} \quad = 24 \cdot 10\frac{\text{nV}}{\sqrt{\text{Hz}}} \cdot \sqrt{1\text{MHz}} = 240\,\mu\text{V} \tag{5.1}$$

Therefore, the amplifier adds white noise with an RMS value of 240 µV which is 50000 times smaller than the new peak voltage of 12 V.

Additionally, an attenuator will be added to model the relatively poor coupling between the wire and the BPM electrodes. These losses are set to −40 dB.

5.1.3 Variable amplifier/attenuator

The variable amplifier/attenuator is constructed to emulate the position of the wire pulled inside a PU. When the wire's position changes, the power generated by each of the pick-up electrodes increases or decreases. To create characteristics similar to a real PU, the variable amplifier/attenuator changes the power in an exponential fashion. It was decided, that the signal power should change by a factor of two for the maximum wire offset. Since the maximum positions are ±10 mm the following equations were added to amplify/attenuate the position dependence. Each electrode signal is multiplied with the derived scaling factor in order to change its amplitude accordingly.

$$\begin{aligned} H_p &= \sqrt{2^{\frac{x}{10}}} \qquad H_n = \sqrt{2^{\frac{-x}{10}}} \\ V_p &= \sqrt{2^{\frac{y}{10}}} \qquad V_n = \sqrt{2^{\frac{-y}{10}}} \end{aligned} \tag{5.2}$$

10 mm is the maximum possible offset of the wire. The lowercase letter denotes the positive or negative direction of the wire. If $x > 0$ then H_p will increase and H_n will decrease. The same applies to the vertical pick-up electrodes.

5.1.4 ADCs

Before the signals of the four electrodes get forwarded to the ADC, they are again amplified so that the full input range of the ADCs is used. The full scale input of an ADC is $1\,\mathrm{V_{PPD}}$ or 1 dBm on a 100 Ω termination. For a maximum offset of 10 mm the signal power is doubled which means it increases by 3 dB. Hence, taking into account the DAC output amplifier, limited coupling between the wire and the PU, and leaving a 3 dB margin for the wire position change, the power into the ADC should be equal to −2 dBm. The gain of the ADC amplifiers can therefore be calculated as:

$$\begin{aligned} -2\mathrm{dBm} &= 1\mathrm{dBm} + 27\mathrm{dB} - 40\mathrm{dB} + G_{\text{ADC amplifier}} \\ &=> G_{\text{ADC amplifier}} = 10\mathrm{dB} \end{aligned} \tag{5.3}$$

These amplifiers will add additional noise to the signal which will be equal to the equation 5.1 just with a different gain factor.
The ADC will then sample the incoming signal at 2.048 GSPS. The resolution of one 12-bit ADC is therefore ≈ 244 µV. Additionally, the ADC will add quantisation noise with a SNR of 56.9 dB for a NSD of −147 dBFs/Hz.

5.1.5 Power Calculation and Normalization

In the simulation, the signal power is simply calculated as the periodogram of the signal sampled by the ADC. Since the time length of the signal is 1 µs that results in 2048 sampling point and therefore in a frequency resolution of $\Delta f = 1\mathrm{MHz}$. The signal is located at 600 MHz, so it falls directly onto a frequency bin of the periodogram with a bandwidth of Δf.
With the calculated power the forward transmission s_{21} can be determined for each of the electrodes, by subtracting the input power of 1 dBm, that the DAC provides. A $\frac{\Delta}{\Sigma}$-normalization is used to calculate the position in the vertical and horizontal plane. The normalization is calculated according to:

$$\begin{aligned} x &= \frac{1}{S_x} \cdot \frac{U_{\text{right}} - U_{\text{left}}}{U_{\text{right}} + U_{\text{left}}} + \delta_x \\ y &= \frac{1}{S_y} \cdot \frac{U_{\text{up}} - U_{\text{down}}}{U_{\text{up}} + U_{\text{down}}} + \delta_y \end{aligned} \tag{5.4}$$

where U_x stands as the forward transmission in a linear scale, S_x and S_y are the sensitivity of the BPM which are needed to retrieve the actual (x, y) position out of the normalisation calculation. S_x and S_y can be calculated if the amplification/attenuation of the signal per millimeter is known. As this was specified in section 5.1.3, it is the slope around zero for Delta-Sigma normalisation. In the implemented simulation, $S_x = S_y = 0.034657\text{mm}^{-1}$.
The constants δ_x and δ_y correct offsets created by unequal attenuation of cables, connectors, the electrodes on its own and other factors. Since there are no such component imperfections in the simulation, the offset corrections are $\delta_x = \delta_y = 0$.

5.2 Results

Table 5.1 summarises the parameters used in simulations.

Table 5.1: Python simulation parameters and values

Parameter	Value	Description
$f_{s_{dac}}$	4.096 GHz	Sampling frequency of the DAC
$f_{s_{adc}}$	2.048 GHz	Sampling frequency of the ADC
f	600 MHz	Frequency of the generated signal
t	1 µs	Time duration of the generated signal
a	0.5 V	V_p of the generated signal
$n_{bits\ DAC}$	14 bit	Vertical resolution of the DAC
$n_{bits\ ADC}$	12 bit	Vertical resolution of the ADC
$V_{max\ DAC}$	0.5 V	Full scale output of the DAC
$V_{max\ ADC}$	0.5 V	Full scale input of the ADC
x	0 mm	Position of the wire in x-plane
y	0 mm	Position of the wire in y-plane
G_{in}	27 dB	Gain of the DAC output amplifier
G_{out}	10 dB	Gain of the ADC input amplifier
e_{in}	$10\frac{\text{nV}}{\sqrt{\text{Hz}}}$	Equivalent input noise of the amplifiers
BW	10 MHz	Bandwidth of the signal

The values of f, x, y and t are not fixed and can be changed to explore different settings of the simulation. For the first simulation, the values in table 5.1 were applied. Since $(x, y) = (0, 0)$, the received power should be equal at all four electrodes in an ideal world. As the data converters and amplifiers add noise to the signal the received power will be slightly different across all four ADCs as can be seen in table 5.2.

Table 5.2: Simulation results of a single simulation where $(x, y) = (0, 0)$

Parameter	[linear]	[dB]
s_{21up}	0.7054	−3.0313
s_{21down}	0.70543	−3.031
s_{21left}	0.7055	−3.03
$s_{21right}$	0.7054	−3.0313

The wire position position calculated according to equation 5.4 is:

$$\begin{aligned} x_{single} &= 28.8542\text{mm} \cdot \frac{0.7054 - 0.7055}{0.7054 + 0.7055} = -0.002\text{mm} \\ y_{single} &= 28.8542\text{mm} \cdot \frac{0.7054 - 0.70543}{0.7054 + 0.70543} = -0.00061\text{mm} \end{aligned} \tag{5.5}$$

Compared to the real value, the deviations are in the order of tens of µm. Figure 5.2 shows positions calculated for 100 simulations.

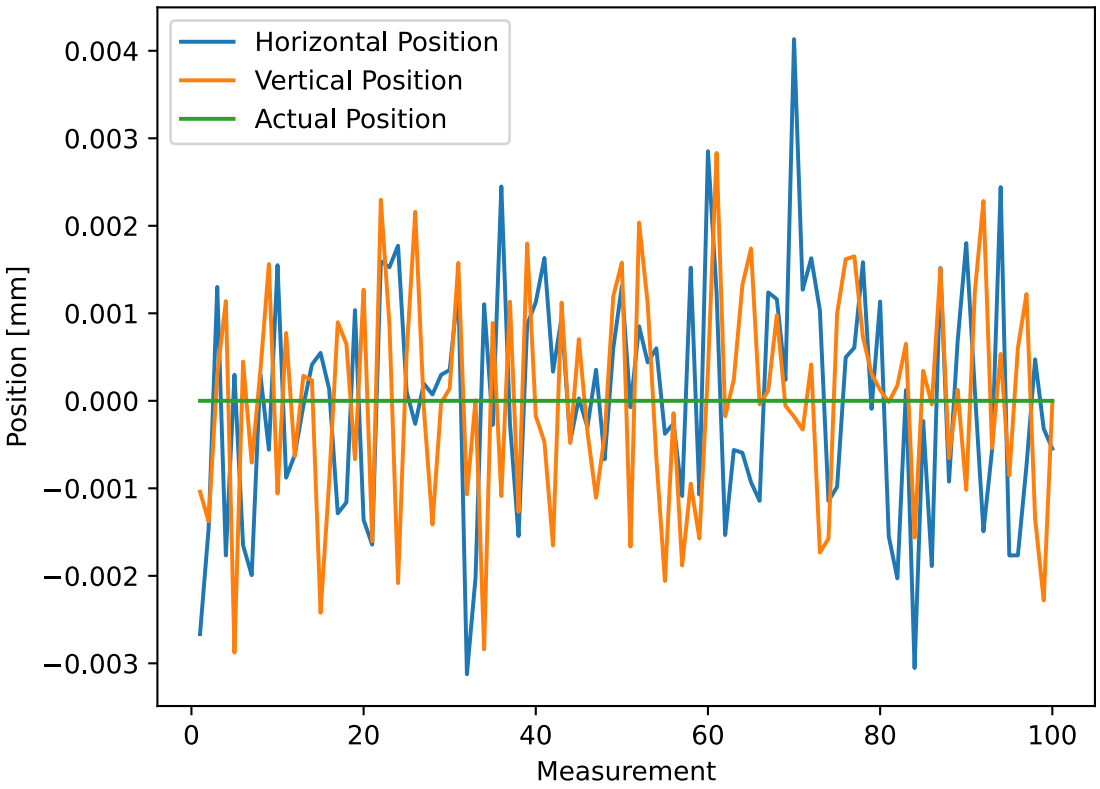


Figure 5.2: 100 position calulcations

The figure shows that the calculated positions are spread around the true position because of the noise in the transmission chain. Improving the results could be achieved by taking N measurements and calculating the average position to increase the precision.
The next simulation used the original parameters but the number of measurements was increased to 100 and their results were averaged for each electrode individually. Table 5.3 shows the obtained results.

The position can again be calculated:

$$\begin{aligned} x_{avg100} &= 28.8542\text{mm} \cdot \frac{0.70541 - 0.705416}{0.70541 + 0.705416} = -0.00019\text{mm} \\ y_{avg100} &= 28.8542\text{mm} \cdot \frac{0.705417 - 0.705412}{0.705417 + 0.705412} = 0.0001\text{mm} \end{aligned} \tag{5.6}$$

Table 5.3: Simulation results after averaging over 100 simulations where $(x, y) = (0, 0)$

Parameter	[linear]	[dB]
s_{21up}	0.705417	-3.03109
s_{21down}	0.705412	-3.03115
s_{21left}	0.705416	-3.03109
$s_{21right}$	0.70541	-3.0312

After comparing a single simulation with averaging over 100 values, averaging reduces the measurement error and gives a better result as a trade-off for measurement duration.

The simulation were also run for the wire position of (x, y) to $(x, y) = (-3, 7)$ to investigate large wire displacements. The calculated power of the signals is shown in table 5.4.

Table 5.4: Simulation results for $(x, y) = (-3, 7)$

Single Measurement			Averaged Measurement	
Parameter	[linear]	[dB]	Mean [linear]	Mean [dB]
s_{21up}	0.8991	-0.9241	0.8991	-0.9239
s_{21down}	0.5535	-5.138	0.0.55345	-5.1384
s_{21left}	0.78277	-2.127	0.0.78271	-2.12799
$s_{21right}$	0.63575	-3.9343	0.63575	-3.9343

The wire position was once more calculated according to equation 5.4.

Table 5.5: Wire position results for $(x, y) = (-3, 7)$

Parameter	Value
x_{single}	-2.986mm
y_{single}	6.865mm
x_{avg100}	-2.99mm
y_{avg100}	6.866mm

As seen in table 5.5, a slight improvement in the wire position estimation precision is achieved also for $(x, y) = (-3, 7)$ by simply averaging the forward transmission and then calculating the position of the wire.
Another interesting factor that can be seen here is, that for large wire displacements the calculated position moves away from the real position. This is the case because the Delta-Sigma normalisation results not on a straight line but a in curve defined in equation 5.2, for an exponential change in power. As the curve flattens toward the edges, calculated values become smaller than the actual position. This can also be witnessed in figure 1.8, where the non-linearities are shown. In some applications, the beam position calculated using the Delta-Sigma must be further linearised using specially derived polynomials.

Chapter 6

Implementation in Xilinx Vivado/Vitis

In the Implementation chapter, the algorithms from sections 4.1 and 4.2 will be discussed together with their implementation in Verilog to work on the ZCU111 FPGA.

6.1 Firmware in Vivado

Vivado Design Suite is a Xilinx software platform for synthesis and analysis of hardware description language designs on FPGAs.

6.1.1 CORDIC algorithm implementation

In general, the CORDIC implementation in Verilog consists of two different blocks, a phase calculator which calculates the current angle of rotation θ and the actual CORDIC block that calculates the value of $sin(\theta)$ and $cos(\theta)$.
The phase calculator gets a phase increment and a phase offset as 20-bit input values. The offset lets one choose the initial phase and the increment sets, in combination with the clocking frequency, the output frequency of the calculated sinus.
The CORDIC block calculates $sin(\theta)$ as described in section 4.1. It uses 20-bit registers for the calculation and outputs a 16-bit value. The number of iterations is fixed to 16 and they are implemented as pipeline stages. The angles of $arctan(2^{-i})$ are stored as parameters in 19-bit registers where $4\_0000_h \hat{=} 90°$. A reset signal can set all internal registers to 0_h to restart the calculation. Since there are 16 pipeline stages the output will only be valid after 16 clock cycles. Additionally, the amplitude is provided in decibels relative to full-scale (dBFs) to the CORDIC block. The output of the module is then connected to the DAC input which produces the analog signal.
The combination of the blocks with the PS and the DAC is shown in figure 6.1.

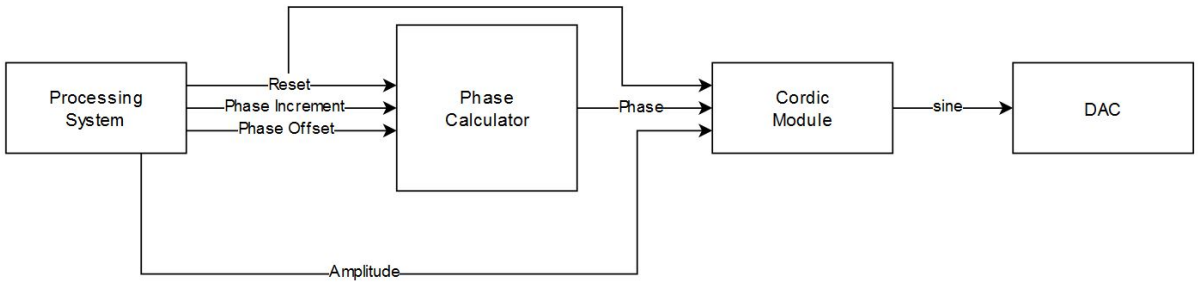


Figure 6.1: Block diagram of the CORDIC Module

Calculating the phase increment which sets a specific frequency can be done using the same equation as for direct digital synthesis:

$$\text{pinc} = \frac{f_{out}}{f_s} \cdot 2^n \tag{6.1}$$

where n describes the bit width of the calculated phase, f_s is the frequency the system is running at, f_{out} the wanted output frequency, and $pinc$ becomes the phase increment. With the system parameters of $n = 20$ and $f_s = 256\,\text{MHz}$ the resolution of the system can be determined. This amounts to $\approx 244.14\,\text{Hz}$ when $pinc = 1$ is chosen.
After a reset, the phase calculator will continuously add $pinc$ to a register and send its output to the CORDIC module. Since it is a simple addition of bits to a fixed length register, an overflow will not affect the calculation but is necessary for it to work properly. The CORDIC module pipelines the calculations and receives a new phase value at every clock cycle. The calculation results are stored in registers after each iteration pipeline stage.
As explained in equation 4.1, one cannot rotate by an angle $> 99.88°$, which is why the initial quadrant must be determined. Since the phase is constructed in signed arithmetic the first two bits of it determine the corresponding quadrant.
When the rotation direction of the pointer is known its starting position must be determined. For the sine, the phase MSB value is checked and the amplitude is maintained or multiplied by -1. The starting position for the sine is always $y = \pm amplitude$.
The cosine calculation is similar. The rotation direction will be the same as for the sine, but the starting position is obtained by logical XOR of the first two bits of the phase giving $x = \pm amplitude$.
The starting point can be either $(x, y) = (-amplitude, -amplitude)$ or $(x, y) = (amplitude, amplitude)$. With clockwise and counter-clockwise rotations of up to 99.88°, every value for sine and cosine can be reached. Figure 6.2 shows the possible starting points in bold blue arrows and their possible rotations.

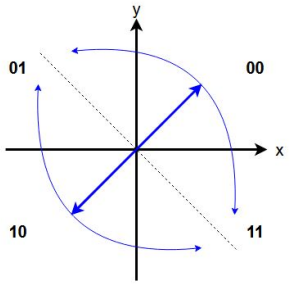


Figure 6.2: Rotations in CORDIC Hardware Implementations

As shown in section 4.1, the rotations of the CORDIC algorithm have to be scaled by a factor of 0.6072. This scaling can be ignored since the real sine and cosine values are not important. Nevertheless, the maximum value must never overflow the registers. Since the CORDIC calculation uses a 20-bit signed amplitude as an input one can calculate the maximum input amplitude as $0x7FFFF \cdot 0.6072_{10} = 318347.066_{10} \hat{=} 0x4DB8B$. Since the FPGA uses integer arithmetic for CORDIC calculations, small rounding errors occur and the max amplitude that can be set by the user is slightly higher than the calculated one and can be set to $0x4DBA4$.
In figure 6.3 one can see that an amplitude of $0x4DBA5$ for a 10 MHz signal will produce overflows in the output and thereby create unwanted frequency components.

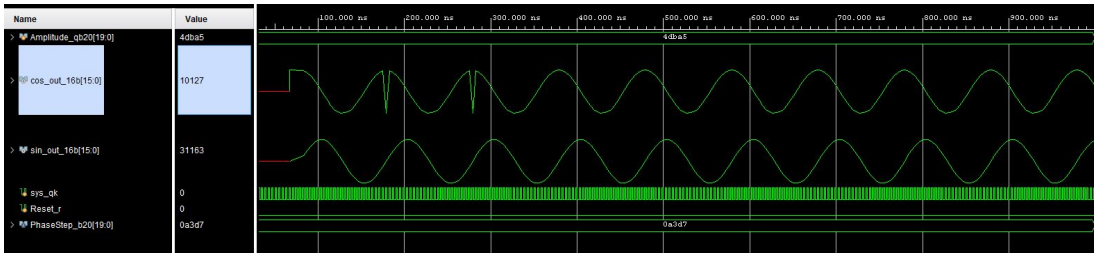


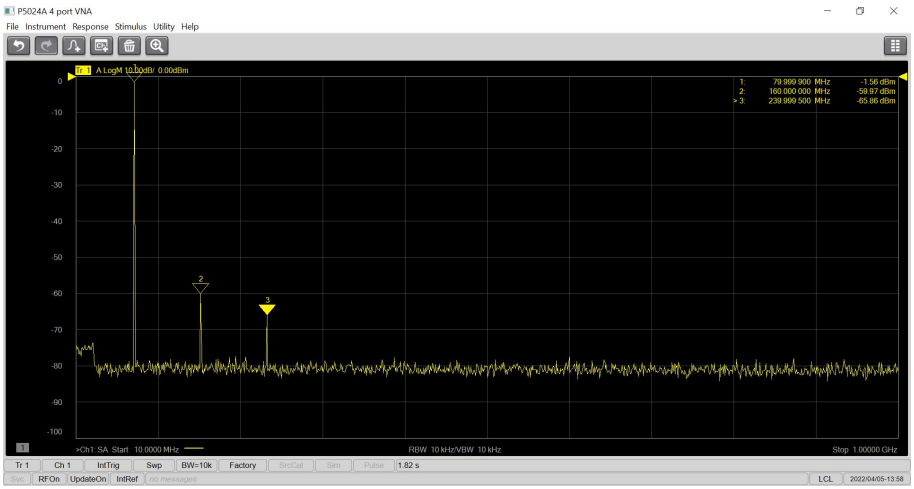
Figure 6.3: Simulation of the CORDIC module with an amplitude of $0x4DBA5$

The rest of the calculations are implemented as sequential logic where each iteration is performed in one clock cycle according to equation 4.7.
An excerpt of one iteration can be seen in figure 6.4 that resets all registers for this iteration to zero if the reset signal is applied or performs a calculation depending on the current sign of the phase and updates the phase register itself.

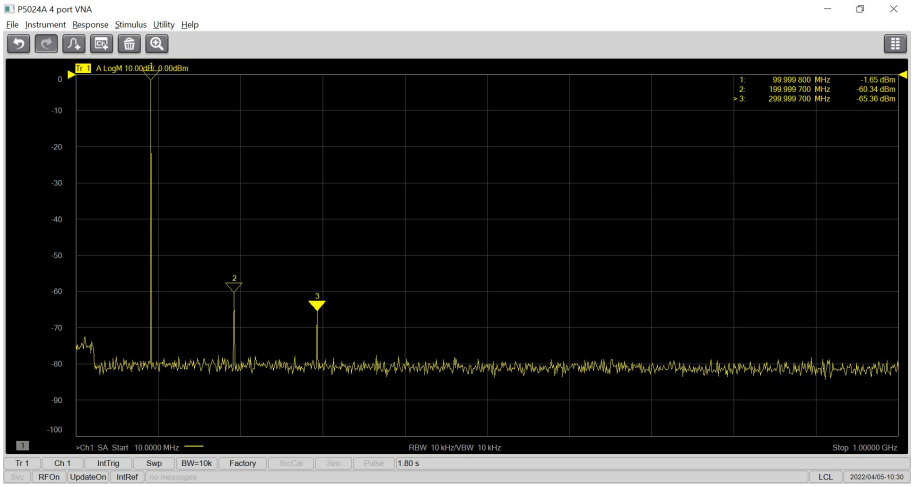
```
always @(posedge clk) begin
    if (rst == 1'b0) begin
        phase_acc_1 <= 20'h00000;
        pre_cos_1 <= 20'h00000;
        pre_sin_1 <= 20'h00000;
    end
    else begin
        phase_acc_1 <= # dly phase_acc_0[18] ? phase_acc_0 + atan_1 : phase_acc_0 - atan_1;
        pre_cos_1   <= # dly phase_acc_0[18] ? pre_cos_0 + {{1{pre_sin_0[19]}}, pre_sin_0[19:1]} : pre_cos_0 - {{1{pre_sin_0[19]}}, pre_sin_0[19:1]};
        pre_sin_1   <= # dly phase_acc_0[18] ? pre_sin_0 - {{1{pre_cos_0[19]}}, pre_cos_0[19:1]} : pre_sin_0 + {{1{pre_cos_0[19]}}, pre_cos_0[19:1]};
    end
end
```

Figure 6.4: Excerpt of one CORDIC Iteration in Verilog

The first test of the signal generation was performed with one CORDIC module and its phase calculator connected to a DAC according to figure 6.1. The calculation for *pinc* was performed in the PS and transmitted using an AXI-Lite interface to GPIOs, which then held the value in a register. The phase calculator and CORDIC-Module were clocked at 256 MHz derived from the DACs sampling frequency of 2.048 GHz. Since the module only produces values at 256 MHz, the DAC interpolates by a factor of eight so that its sampling frequency is reached.
With the clock of the module runs at 256 MHz, according to the sampling theorem, only frequencies up to $\frac{256\,\text{MHz}}{2} = 128\,\text{MHz}$ can be generated.
To validate if the right frequencies are produced, the DAC was connected to a spectrum analyzer to observe the power spectral density. Figure 6.5 shows the output spectrum of a generated sine at 80 MHz, in the upper figure, and 100 MHz, in the lower figure.



(a) Single CORDIC with $f_{out} = 80$MHz



(b) Single CORDIC with $f_{out} = 100$MHz

Figure 6.5: Spectra for single CORDIC module with different output frequencies

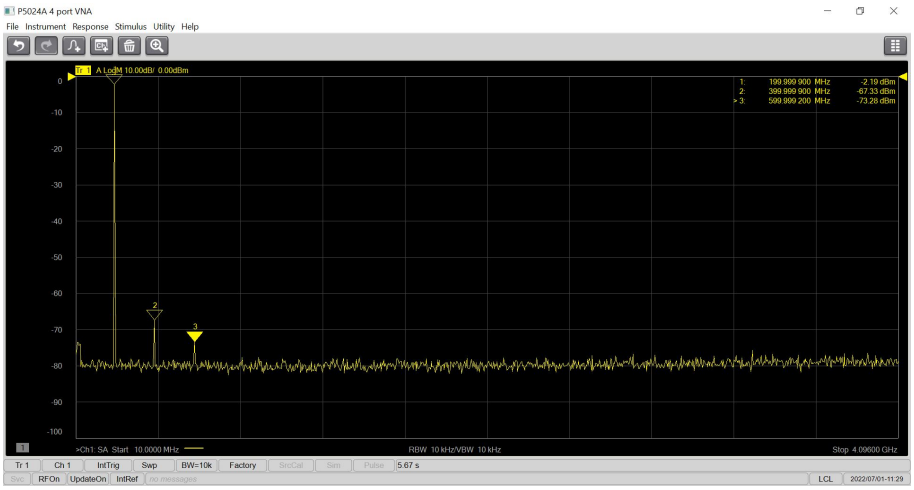
The module produces a strong fundamental frequency for 80 MHz at −1.56 dBm with a visible second and third harmonic at around −58 dBm and −64 dBm respectively. Since the maximum amplitude is provided without any overflow, it is expected that the output power of the DAC is close to 4 dBm but the measured value is around 5.56 dB lower. First of all, because of the impedance matching to 50 Ω, only half of the power of the DAC can be measured at the spectrum analyzer. This is where the first 3 dB are lost. Secondly due to losses of the balun board which performs the impedance matching, and the analog bandwidth of the DAC the measured power is lower. Additionally, there is also some attenuation of cables and connectors.
The same pattern is visible for a generation of 100 MHz with the fundamental frequency at −1.65 dBm and the harmonics at −60.34 dBm and −65.36 dBm respectively. However, there is an additional frequency component at $f_{dac} - f_{out} = 1968\,\text{MHz}/1948\,\text{MHz}$. This component is not seen in figure 6.5 since the spectrum analyzer sweep was limited to 1 GHz. This spurious component is created because a sampled signal, which the DAC produces, will always have a periodic spectrum that is mirrored every half of the sampling frequency.
To eliminate the components out of the first Nyquist band, an analog low-pass filter should be applied at the DAC output to attenuate them far below the noise level. If there is no steep enough low-pass at hand, one can also increase the DAC sampling frequency to push the periodic continuations onto higher frequencies where they will be attenuated more by a less aggressive low-pass filter.
With this setup, using only one CORDIC module at 256 MHz, generation of fundamental frequencies higher than 128 MHz is not possible. To achieve higher fundamental frequencies at the DAC output, the hardware inside the FPGA has to be changed. Multiple CORDIC modules and phase calculators running in parallel with known phase offsets and a fixed phase increment are used. The phase increment will be calculated as described in equation 6.1 but for frequencies exceeding f_s the increment will exceed the phase size in bits. That is where multiple parallel modules are necessary. Since the DAC can take multiple values at once, for example, 16 values at 256 MHz and output them one by one at 4.096 GHz, the CORDIC modules can run at a lower clock speed than the sampling frequency and still achieve a high sampling rate at the DAC. Firstly, the phase increment is calculated for the high output frequency, so that the increment is greater than the phase width. With this phase increment, the offset for each module is calculated as:

$$\text{poff}_i = i \cdot \frac{pinc}{N_{\text{Modules}}} \mod 2^n \tag{6.2}$$

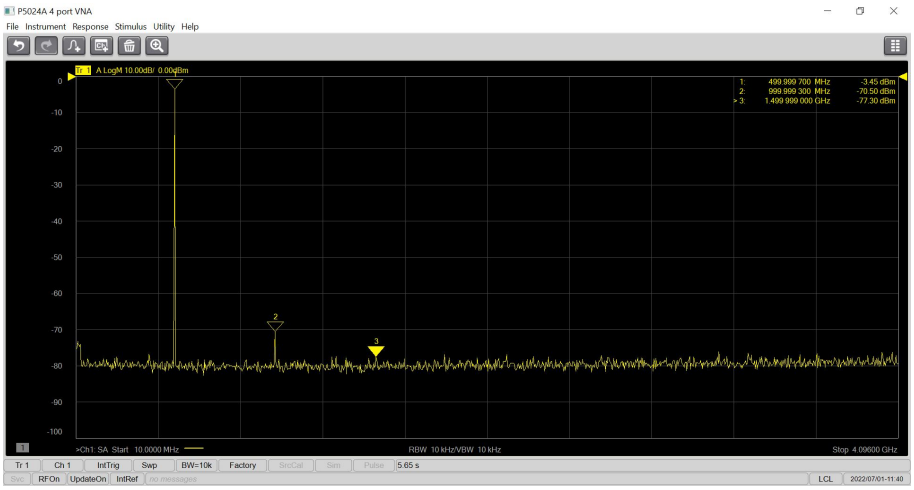
where i is the i-th module. The actual phase increment that can be implemented in the FPGA has to be smaller than the phase width of 2^n. Since sine and cosine rotate on a circle, an overflow in the phase register is the same as a change from 359° to 0°. The new phase increment can be then calculated as:

$$\text{pinc}_{\text{new}} = \text{pinc} \mod 2^n \tag{6.3}$$

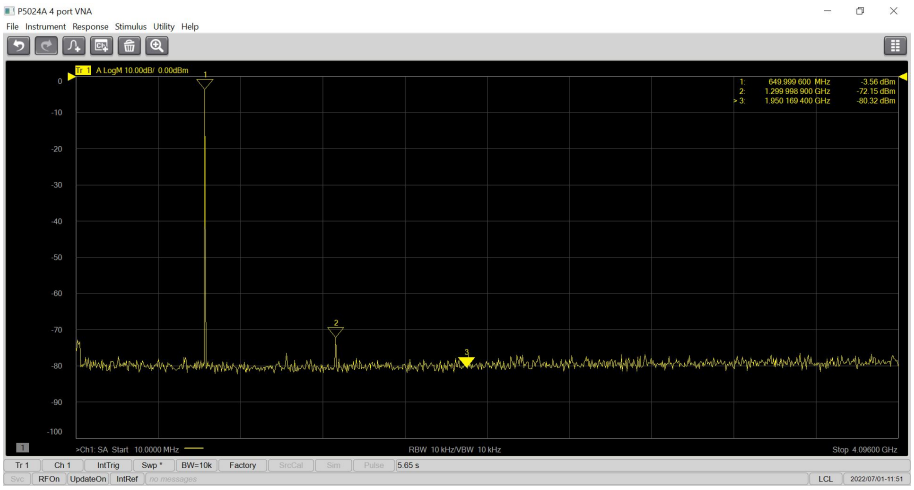
These values are assigned to the input of each module to provide the DAC with all values simultaneously. The outputs of the CORDIC modules are concatenated to create one large bus which will be the input to the DAC. This change was also implemented to push the periodic continuations of the spectrum to higher frequencies in order to not use a very steep low-pass filter. The basic structure stays the same as illustrated in figure 6.1 only that there are now 16 blocks working in parallel at 256 MHz which results in a DAC sampling frequency of 4.096 GHz.
Three different output frequencies for this new setup are shown in figure 6.6. These frequencies are 200 MHz, 500 MHz and 650 MHz. In the analog signal path, there was a low-pass filter with a cutoff frequency of $f_{\text{cutoff}} = 650\,\text{MHz}$ installed to attenuate harmonics and the mirrored frequency components. As seen in the measurements with the spectrum analyzer, the output power of all three measurements differ. For higher frequencies, the power decreases because of the frequency dependence of cables and baluns. Additionally, for the 200 MHz measurement, the spectrum analyzer measurements are only off by 100 Hz. This error increases slightly for higher frequencies so that for the 500 MHz and 650 MHz measurement the error is roughly 400 Hz.
Since using higher frequencies is not foreseen, no measurements were taken for $f_{out} \geq 700\,\text{MHz}$.



(a) Multi CORDIC with $f_{out} = 200\,\text{MHz}$



(b) Multi CORDIC with $f_{out} = 500\,\text{MHz}$



(c) Multi CORDIC with $f_{out} = 650\,\text{MHz}$

Figure 6.6: Spectra for different CORDIC output frequencies

6.1.2 Goertzel algorithm implementation

The algorithm is also implemented in Verilog on the hardware side inside the FPGA logic. From the equations derived in chapter 4.2 the equations 4.14 and 4.15 are of interest. Equation 4.15 will provide the magnitude and phase of a specific frequency component, but as known from the derivation only the Nth output value is valid. This can be used to save computing resources and only calculate them at the end of the process. Hence, using equation 4.15 is necessary only after N iterations and initially only equation 4.14 is needed to calculate the states $s[n]$ and $s[n-1]$.
In equation 4.14, $s[n-1]$ is multiplied by $2\cos(\frac{r\pi k}{N})$ which is fixed for a specific frequency. As the factor stays constant during the calculation process for a specific frequency, it can be calculated in the PS and passed over to the PL. This way the cosine can be easily calculated and adjusted. The cosine value is then stored in a 32-bit register. Since the value of $\cos(x)$ never exceeds ± 1, a floating point arithmetic has to be implemented on the FPGA side. The choice fell on a 30-bit fixed-point arithmetic where the decimal point gets shifted 30 bits to the left. The register is divided into a part to the left of the decimal point, where the bits have their usual value of 2^i, and a part to the right of the decimal point, where the bits are assigned a value of 2^{-i}. Floating values can therefore be represented with a fixed precision of $2^{-30} \approx 0.931 \cdot 10^{-9}$. As seen in equation 4.15, $s[n-1]$ is multiplied by a complex number which makes it hard to implement in the PL. Luckily, with the use of Euler's formula, the term can be split up into real and imaginary parts. This complex value will then represent the DFT value of the frequency bin scaled by N (the number of values used in the calculation). The two resulting calculations for real and imaginary part are:

$$\begin{aligned} \Re(y_k[n]) &= \cos(2\pi\frac{k}{N})s[n-1] - s[n-2] \\ \Im(y_k[n]) &= \sin(2\pi\frac{k}{N})s[n-1] \end{aligned} \tag{6.4}$$

For the calculation of the imaginary part, $\sin(2\pi\frac{k}{N})$ is needed. Since it is also a constant for a specific frequency bin, just as $\cos(2\pi\frac{k}{N})$ in the calculation for the real part and in equation 4.14, it will be provided to the FPGA in the same way as the cosine, as a 30 bit fixed-point floating arithmetic. Before the two values of equation 6.4 are sent to the PS for further processing, they are divided by N to scale them to their true value. This is done by downshifting the register since only powers of 2 are used as the number of values that contribute to the calculation.

The implementation of the receiving chain inside the FPGA looks similar to the block diagram illustrated in figure 6.7.

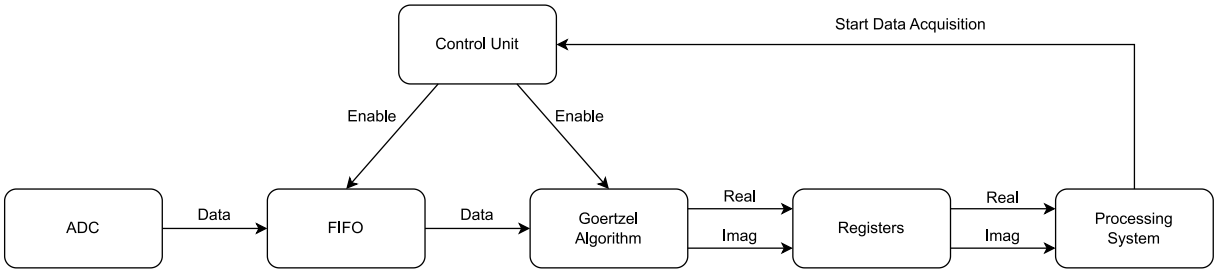


Figure 6.7: Processing chain for the Goertzel algorithm

The processing chain contains an ADC, a FIFO to store the incoming data, the actual Goertzel algorithm, a control unit to supervise the process, and some registers to provide the values over the AXI-Lite interface to the PS. Additionally, the PS provides the needed parameters as discussed earlier.
The first stage in the chain is the ADC which samples 12-bit values at $fs_{adc} = 2.048\,\text{GHz}$ and outputs a 96-bit value every 256 MHz. This value is a concatenation of the last 8 values sampled by the ADC. These values get transmitted to a FIFO, which can store up to 262144 12-bit values. This results in a measurement time of $262144 \cdot \frac{1}{2.048\,\text{GHz}} = 128\,\mu\text{s}$. The measurement time affects the frequency resolution of the DFT. With 128 µs the frequency bin resolution of the DFT is: $\Delta f = \frac{1}{128\,\mu\text{s}} = 7812.5\,\text{Hz}$. The FIFO also changes the clocking domain, switching from an input read clock of 256 MHz to an output write clock of 64 MHz which is the one used by the Goertzel algorithm. Moreover, there is another block that splits the 96-bit value that contains 8 ADC samples into 8 segments of 12-bits, which can be processed by the algorithm one by one. This data width converter block is not shown in figure 6.7. The Goertzel algorithm will produce real and imaginary part for the desired frequency bin and store them in a register that the PS can access. The values are stored until a new measurement is performed. In order to manage the whole acquisition process, a control unit was developed to reset, enable and manage the different states of the acquisition routine. Knowing that 262144 values are processed by a 64 MHz clock the calculation time is $262144 \cdot \frac{1}{64\,\text{MHz}} = 4.096\,\text{ms}$.
The ADC, FIFO and data width converter are Xilinx IP blocks that have been configured to the need of this project and their code cannot be changed. The remaining blocks, on the other hand, are self-written and will be explained in more detail.
The control block operates in the 256 MHz clock domain and has four different states to control the acquisition and calculation process. Figure 6.8 illustrates the different states and their purpose.

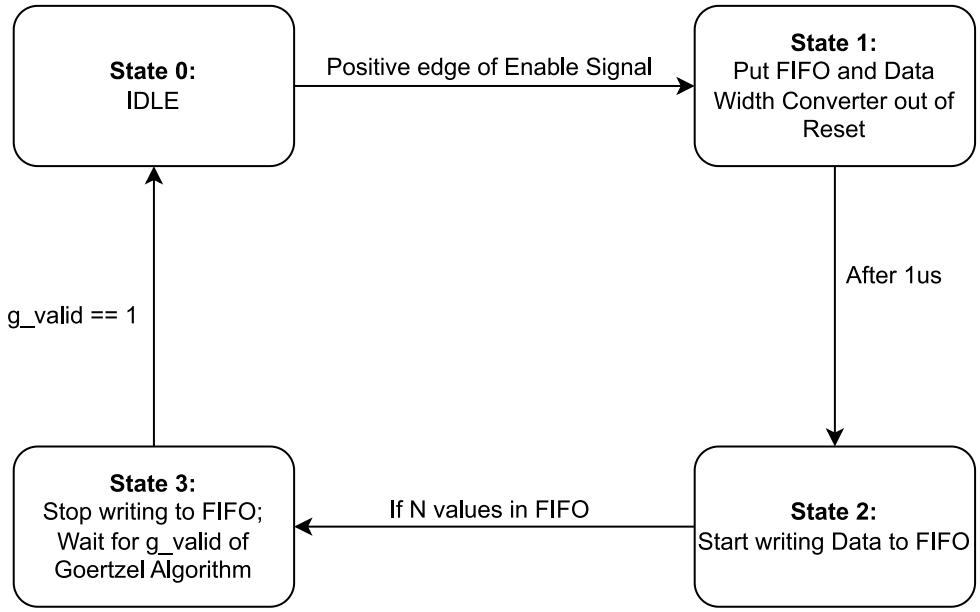


Figure 6.8: Flow chart of the Control Unit

When no measurements are being taken, the control unit is IDLE and waits for a positive edge of a start signal coming from the PS. Once the edge is detected, all used blocks are put out of reset so that their functionality can be used. The block waits for 1 µs for the FIFO and the data width converter to properly exit their reset state. After the wait time, state three is entered and the Goertzel Algorithm module is put out of reset. The ADC is allowed to write data to the FIFO. After N values have been written to the FIFO, the control unit stops the writing process and waits for a valid signal coming from the Goertzel

algorithm block. Once this signal is received, the control unit goes back to the IDLE state and sends a signal to the PS stating the calculation has finished.
The block responsible for the Goertzel algorithm stays in reset until the ADC starts writing to the FIFO. Starting from there, it waits for a valid signal of the data width converter to start the calculation. The registers holding the calculated values must have an appropriate size.
The ADC produces 12-bit signed values, that represent voltages ranging between $-0.5\,\mathrm{V}$ and $0.5\,\mathrm{V}$. The calculation is performed over 262144 samples. Therefore, the calculated value will be scaled by $262144 = 2^{18}$. Additionally, the 12-bit value of the ADC needs to be converted to a voltage by dividing by $4096 = 2^{12}$. This can be done at the beginning of the calculation for each sample or it can be treated as an additional scaling factor and applied at the end. Since it is easier to do the scaling once, the two factors are combined to 2^{30} and applied at the end of the calculation. As all values are powers of two, this can be easily achieved by shifting the result by 30 bits. Knowing that the maximum input to the ADC is 1 dBm, one can calculate back a maximum magnitude of the DFT bin of $0.5\,\mathrm{V}$ for a $100\,\Omega$ system. Multiplying this by the scaling factor of 2^{30} shows that 28-bits are needed to hold the integer part of the calculation. Since the floating part is chosen to be 30-bits wide, this leads to a result width of 58 bits. Moreover, a multiplication of two fixed point values shifts the decimal point upwards to two times its original value. One multiplication between two 64-bit fixed point values with the decimal point at bit 30 will produce a 128-bit value with the decimal point at bit 60. As there is sufficient space in the FPGA, the used width for the calculation is set to 128-bit values.
In principle, the Goertzel block simply executes 4.15 N times and afterward computes the real and imaginary part of the DFT bin according to equation 6.4. As long as the data width converter provides a high signal to the Goertzel unit, it performs its calculation. When the valid signal drops back to zero, an additional calculation is performed because the input value is not instantly processed but stored in a register and processed only during the following clock cycle. Once all the state calculations are performed, the real and imaginary parts are calculated. The output of the Goertzel block consists of the scaled real and imaginary parts. The maximum value for one of the two values will always be smaller than $0.354\,\mathrm{V}$. This can be taken into account by the output registers which, to avoid wasting any bits for the integer part, provide only the bits after the decimal point as an output in a 32-bit register.
The Goertzel unit indicates the end of calculations by writing a high signal to the control unit, which forwards it to an AXI register accessible from the PS.

6.2 FPGA utilisation

All blocks implemented inside the FPGA take up register space and use different resources. After the implementation is run in the Vivado Design Suite, it lists the utilisation of different resources of the FPGA. For example the number of used LUTs, block RAM and DSP slices. Figure 6.9 shows the resource allocation of the FPGA for this project in which eight ADCs can perform the calculation simultaneously.

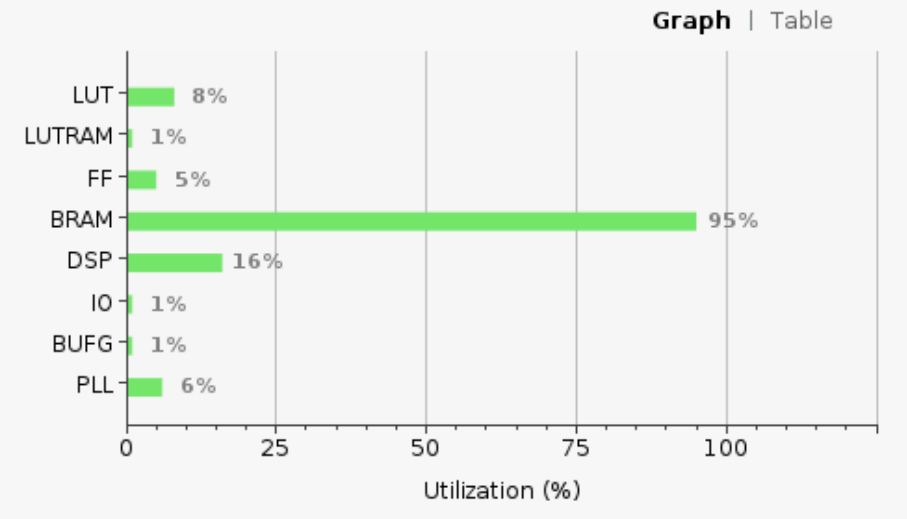


Figure 6.9: FPGA resource utilisation

It is clear that the block RAM is utilised the most because all the ADC values need to be stored temporarily in a FIFO in order to perform the Goertzel algorithm. Since the storage space is kept inside the FPGA, almost all of the block RAM is used. The DSP slices are the second most utilised as they are needed for multiplications used by the Goertzel algorithm inside the FPGA.

6.3 BareMetal Application

For first testing and validation of the FPGA implementation, a small BareMetal application was developed. BareMetal programming operates without the abstraction of an operating system. A BareMetal application interacts with the system at a hardware level and is built for the specific hardware, which in the present project is the PL. It aims to run simple programs efficiently without the usage of operating systems such as Linux or Windows. The whole application is written in the programming language C.
In this application, the registers for the CORDIC and Goertzel blocks will be set up according to the frequency the CORDIC is programmed to output. Additionally, the values produced by the DFT calculation can be read by the PS. Figure 6.10 illustrates the flowchart of the BareMetal application.
The program is organised linearly to perform the specific task of taking one or multiple measurements of one ADC. One DAC will output a sine wave at a specific frequency and one ADC will receive this signal and perform a power calculation for the set frequency. At first, the sampling clocks of the Radio-Frequency Data Converters (RFDCs) have to be set up correctly. This is achieved with C files which program the RFDC PLLs, according to the user's needs, using the I^2C bus.

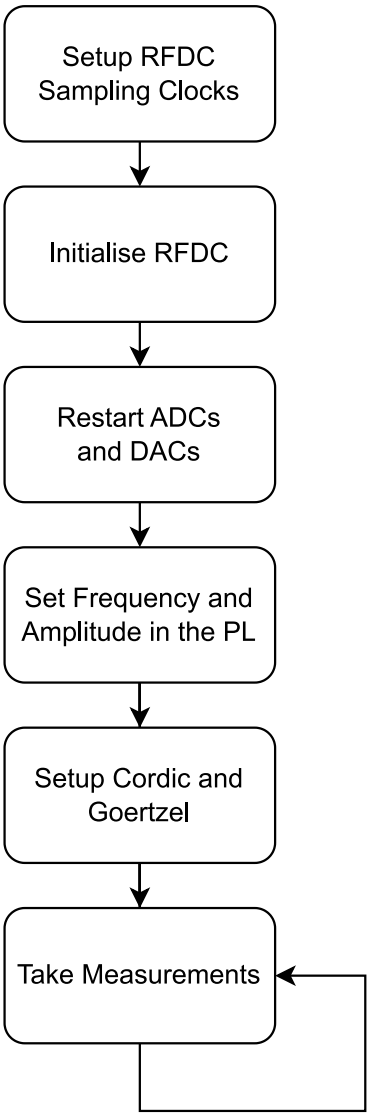


Figure 6.10: Flow chart of the BareMetal application

After setting up the necessary clocks, the RFDCs, the driver and controller of the data converters can be initialized and the used tiles will be restarted. The user can now input the frequency in Hz and output amplitude in dBFs that will be produced by the CORDIC module in the FPGA. The PL blocks will be set up accordingly and a measurement can take place.
The software makes it possible to take a single measurement or N measurements by computing the average power and phase of the received signal as well as the standard deviation, minimum and maximum values of these N measurements.
The calculated real and imaginary parts are then processed by software to calculate the power and phase using the periodogram approach as described in chapter 4.3.

6.4 Results

In this section, the results of the measurements using the BareMetal application are shown and discussed. Figure 6.11 shows the test setup with the output of the DAC in tile two connected directly to an analog low-pass filter (MiniCircuits SLP-750+) with a 3 dB cutoff frequency of 770 MHz. The filter is connected via an SMA cable to an ADC input.

Figure 6.11: Measurement setup of the ZCU111

Different measurements were taken in order to verify the functionality of the setup and calculation process. The first measurement is related to the calculated power of the signal. Measurements at different frequencies were taken and compared with the signal power measured by the spectrum analyzer. All measurements were averaged over 100 values in order to reduce the measurement uncertainty. Table 6.1 shows the obtained results. The DAC provides an output power of ≈ 4 dBm which will, after impedance matching, be reduced by 3 dB. Hence, without any losses, the maximum power that can be measured at the load is 1 dB. With the additional losses of the baluns, connectors and low-pass filter, the measured

power is further reduced.

Table 6.1: Comparison of the measured and calculated signal power

Frequency [MHz]	Power Spectrum Analyzer [dBm]	Calculated Power [dBm]
100	−1.67	−1.552
200	−2.19	−2.283
300	−2.83	−3.214
400	−3.34	−4.346
500	−3.45	−4.06
600	−3.45	−3.787

The measured and calculated power decrease as the set frequency increases. For the calculation on the RFSoC, the power for input frequencies of 300 MHz − 600 MHz are striking. In contrast to the spectrum analyzers measurement, where the power keeps decreasing continuously for higher frequencies, the power calculated by the board is significantly smaller for 400 MHz and starts to increase again for higher frequencies.
The second test was related to the phase of the received signal. Therefore, cables of different lengths were taken to compare how the signal phase changes. Since the phase is related to the group delay, the length difference of the cables can be used to estimate the difference in phase that should occur in the measurement. The measurements were taken for different frequencies, as seen in table 6.2, and the cables were chosen to have a delay difference of 1 ns.

Table 6.2: Calculated Phase Differences

Frequency [MHz]	Expected Phase Difference [°]	Measured Phase Difference [°]
100	36	35.6
200	72	73.2
300	108	115
400	144	143
500	180	180.6
600	216	226

The maximum deviation of the expected phase is about 10° and occurred at $f_{out} = 600\,\text{MHz}$. Since the exact length of the cables was not measured and precise phase detection is not required from the developed system, this measurement result is satisfying.
To validate the accuracy of DFT calculation, a frequency generator was used to provide a set frequency at a known power. The received power was calculated using the Goertzel algorithm combined with the periodogram up until an offset of $5 \cdot \Delta f = 5 \cdot 7812.5\,\text{Hz} = 39.0625\,\text{kHz}$. The frequency of the signal generator was set to 100 MHz. Since the power calculation does not use any specific windowing function, a rectangular window was applied. The different measured powers are illustrated in figure 6.12. As expected, with deviation from the main frequency bin, the calculated power decreases rapidly to around −60 dBm at a frequency difference of ≈ 40 kHz. At deviations of $128 \cdot \Delta f = 1\,\text{MHz}$ and $1280 \cdot \Delta f = 10\,\text{MHz}$ the measured power is around −90 dBm and −100 dBm respectively.

Since up to eight ADCs are used simultaneously in the test bench, their measurements have to be compared as well. To achieve this, the DAC output was connected to the first ADC and the measured values were averaged over 10000 measurements. After the measurements, the cable was disconnected from the first ADC input and reconnected to the second one etc. Tabel 6.3 shows power and phase measurements for the first four ADCs at a frequency of $f = 200\,\text{MHz}$.

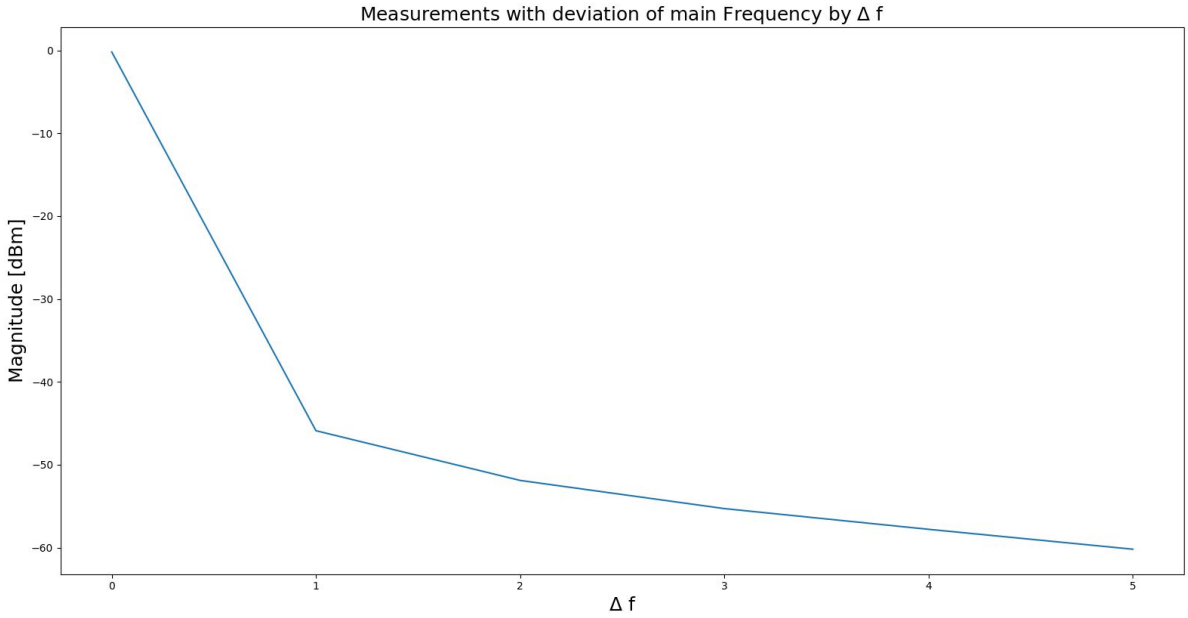


Figure 6.12: Measurements closely around known frequency

Table 6.3: Measurements of 4 ADCs

Parameter	**ADC0**	**ADC1**	**ADC2**	**ADC3**
Average Magnitude [dBm]	−2.5674	−2.433	−2.3209	−2.64
Min Magnitude [dBm]	−2.5792	−2.441	−2.3286	−2.646
Max Magnitude [dBm]	−2.5599	−2.413	−2.315	−2.632
Standard Deviation [dBm]	0.00215	0.0019	0.0017	0.00195
Average Phase [°]	165.258	164.3	168.029	163.988
Min Phase [°]	165.234	164.225	167.96	163.902
Max Phase [°]	165.307	164.582	168.356	164.334
Standard Deviation [°]	0.011	0.074	0.08	0.096

The measurements prove that the measurement spread on a single ADC is small. A standard deviation of 0.002 dB is equivalent to 0.0046%.
The measurements also show that the power measured by the different ADCs differs. This leads to the suspicion that the baluns and maybe even the ADCs have small differences in their attenuation. Since for a BPM measurement these differences between the ADCs will already lead to errors in the determination of the electrical center, a calibration routine must be implemented to eliminate the attenuation of baluns, cables and connectors.

To determine the noise floor of the setup, the connection between ADC and DAC was removed. The calculated noise power at the ADC was determined to ≈ -112 dBm which matches the theoretical prediction from chapter 3.5 at a bandwidth of 7.8125 kHz.

Chapter 7

Embedded Linux

In order for the ZCU111 Xilinx Ultrascale+ RFSoC to establish a communication with the LabView program that controls the mechanical structures of the stretched wire test bench, it was decided to use an embedded Linux distribution specifically developed for Xilinx Ultrascale+ boards. This brings all the advantages of a proper operating system, such as specific hardware drivers, software packages and its board specific Linux kernel for the given processor architecture. All of this is combined and adjustable in the Linux distribution „PetaLinux“ provided by Xilinx and based on the Yocto-Project.

7.1 PetaLinux

PetaLinux is an embedded Linux Software Development Kit targeting FPGA-based SoC designs. In order to boot a Linux Distribution, the following components are essential:

- A Bootloader
- A Device-Tree
- A Linux Kernel
- A root file system

The bootloader is responsible for initializing the underlying hardware, memory, registers and clocks before loading the Linux kernel into the processor's memory. The device-tree holds all the information about the hardware description and provides this information to the kernel. The kernel activates various services and processes for which it is configured and mounts the root file system. The root file system contains all crucial files necessary for the operating system functioning and installed user applications.

7.1.1 Bootloader

The bootloader is a piece of software, stored in non-volatile memory, that is executed when the system is powered on. The bootloader in the Xilinx RFSoC is divided into two stages, the First Stage Bootloader (FSBL) and the U-Boot stage. The FSBL initialises important blocks in the processing subsystem. This includes clearing the reset of the processors, initialising clocks, memory and different interfaces. Additionally, it can load the bitstream into the PL if needed [14] [15].
U-Boot works as a secondary bootloader after the FSBL hand-off. It loads Linux on the APU and configures the rest of the peripherals in the processing subsystem based on the board configuration [14] [15].

Since the device will be used as an independent measurement instrument the whole Linux operating system image will be stored on an SD-Card. Therefore, the necessary boot image for the PetaLinux distribution will be stored on a FAT32 BOOT partition on the SD-Card.

7.1.2 Device-Tree

The Linux kernel needs to know on which processor it is being executed, the peripherals that are associated with the processor, their interfacing with the processor and their physical addresses. In order to initialize the drivers and the services associated with these peripherals, the kernel also needs to check if the functionalities that have been activated in its configuration are actually supported by the hardware that it is controlling. This could concern the clocks and registers of the hardware or any peripheral associated with the hardware such as the external memory, Ethernet or I2C.
One way of achieving this would be by hard-coding the hardware description directly into the bootloader. The disadvantage of such a method is the missing flexibility, requiring every hardware design to have its own bootloader with its specific hardware description.
The Zynq UltraScale+ RFSoC uses the device-tree to work around this problem. It passes the device and peripheral information such as physical addresses of devices, I/O register addresses, memory address space and interrupt information to the kernel during boot time [15] [16]. Listing 1 shows a small excerpt of the device-tree file for the ZCU111 for the USB node.
USB port zero has the physical address $ff9d0000_h$ assigned to it as well as different configurations for its clocks, register addresses, resets and more. The USBs „child“ node is associated with the „dwc3“, a SuperSpeed USB 3.0 Dual-Role-Device controller. The controller address is also set, as well as its different parameters like the compatible drivers, register addresses and interrupts.
The device-tree is represented in textual format in a file with the „.dts“ extension. This is a source text file that describes hardware devices and interconnecting buses interfaced with the computing hardware. It is organized in the form of „nodes“ that have a root location represented by „/“ just like in the Linux root file system. Every node has a name that represents a device or a bus interfaced with the processor and the node consists of different „properties“. Each parent node for a certain peripheral or bus may contain „child“ nodes for devices that are interfaced with that peripheral or bus. Values of the properties can be strings, lists of strings or they could be empty if the absence or presence of the value conveys a Boolean logic to the kernel. The device-tree source file is compiled into a binary device-tree blob (.dtb).

```
usb0: usb0@ff9d0000 {
        #address-cells = <2>;
        #size-cells = <2>;
        status = "disabled";
        compatible = "xlnx,zynqmp-dwc3";
        reg = <0x0 0xff9d0000 0x0 0x100>;
        clock-names = "bus_clk", "ref_clk";
        power-domains = <&zynqmp_firmware PD_USB_0>;
        resets = <&zynqmp_reset ZYNQMP_RESET_USB0_CORERESET>,
                 <&zynqmp_reset ZYNQMP_RESET_USB0_HIBERRESET>,
                 <&zynqmp_reset ZYNQMP_RESET_USB0_APB>;
        reset-names = "usb_crst", "usb_hibrst", "usb_apbrst";
        reset-gpio = <&modepin_gpio 1 0>;
        ranges;

        dwc3_0: dwc3@fe200000 {
                compatible = "snps,dwc3";
                status = "disabled";
                reg = <0x0 0xfe200000 0x0 0x40000>;
                interrupt-parent = <&gic>;
                interrupt-names = "dwc_usb3", "otg", "hiber";
                interrupts = <0 65 4>, <0 69 4>, <0 75 4>;
                #stream-id-cells = <1>;
                iommus = <&smmu 0x860>;
                snps,quirk-frame-length-adjustment = <0x20>;
                snps,refclk_fladj;
                snps,enable_guctl1_resume_quirk;
                snps,enable_guctl1_ipd_quirk;
                snps,xhci-stream-quirk;
                /* snps,enable-hibernation; */
        };
};
```

Listing 1: Excerpt of the ZCU111 Device-Tree

7.1.3 Linux Kernel

The kernel is the essential foundation of an operating system and generally has complete control over everything that occurs in the system. It is the first element that is loaded into memory during boot-up and stays there for the entire session. It is therefore crucial that it is as small as possible while still providing all the functionalities needed by the operating system and applications. Because of its critical nature, the kernel code is usually written into a protected memory area that prevents it from being overwritten during runtime. The kernel performs its tasks in a dedicated kernel space in order to not interfere with user applications executed in the user space. Such separation is necessary to prevent the system from crashing or becoming unstable. The kernel provides basic services for all other parts of the operating system. This includes memory management, process management, file management and I/O management to access peripheral devices. These so called „system calls“ are requested by other parts of the operating system or applications. Process management is the part of the kernel that ensures, that every process or task gets its turn to run on the processor. It also manages, if multiple processes are running simultaneously, that they do not interfere with each other by writing to already used memory space. The contents of a kernel can vary, but usually they include a scheduler, which determines which

process gets dedicated processing time on the kernel, a supervisor, that grants the use of the computer to each process when it is scheduled, an interrupt handler, which handles all interrupts from various hardware peripherals, and a memory manager, which allocates the systems address spaces among all users of the kernel's services [17] [18].

7.1.4 Root File System

The root file system is at the top of the hierarchical file tree. It contains the files and directories critical for system operation, including the device directory and programs for booting the system. The root file system also contains mount points where file systems can be mounted to connect to the root file system hierarchy.
The main directories in the root file system are :

- /bin: Contains user executable files
- /boot: Contains the bootloader, kernel executable and configuration files required to boot a Linux computer
- /dev: Includes all device files for every hardware attached to the system; devices in Linux are presented as files and can be accessed in the same way
- /etc: Contains system configuration files
- /home: Points to the home directory for users; Each user has its own sub-directory in /home
- /lib: Includes shared library files that are necessary to boot the system
- /media: Provides a mounting point for the external removable media devices such as USB sticks
- /mnt: Provides a temporary mount point for the regular filesystem (can be used during repairs)
- /opt: Contains optional files such as vendor supplied application programs
- /root: Points to the root user home directory
- /sbin: Contains system binary files for system administration
- /tmp: Stores temporary files by the operating system or applications
- /usr: Contains shareable, read-only files including executable binaries and libraries
- /var: Contains variable data files such as log files, databases, etc.

7.2 Building a custom PetaLinux Distribution

To create a custom PetaLinux Distribution for the ZCU111 Evaluation Board some prerequisite software and files are needed: the PetaLinux installer, the specific board support package and the hardware design file created by Vivado. This project makes use of the PetaLinux release 2021.2 which can be downloaded from `https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2021-2.html`. On the host PC ran the CentOS 7 operating system.

After the installation of PetaLinux, one can commence with the creation of a custom distribution. In order to create a project, a shell script, shown in listing 2, was written. In order to avoid warnings, gcc-6 tool set was installed before and is enabled in line 13 (only needed to avoid gcc warnings and if gcc6 or higher is not already installed). The script creates a PetaLinux project called „zcu111-petalinux“ in the current working directory.

```
1  #!/bin/bash
2  ##########################
3  #Build the whole project
4  ##########################
5  export BSP_PATH=            #Path to the Board-Support-Package
6  export PETALINUX_PATH=      #Path to Petalinux Installation
7  export TMP_BUILD_DIR=       #Path of temp folder
8  export HW_PATH=             #Path to .xsa Vivado file
9
10 sudo mkdir -p $TMP_BUILD_DIR
11 sudo chmod 777 $TMP_BUILD_DIR
12
13 scl enable devtoolset-6 bash
14 source $PETALINUX_PATH/settings.sh
15
16 petalinux-create --type project --source $BSP_PATH --tmpdir $TMP_BUILD_DIR --name zcu111-petalinux
17 cd zcu111-petalinux
18
19 petalinux-config --get-hw-description $HW_PATH --silentconfig
```

Listing 2: Shell script to create PetaLinux project

Next, the paths to different locations must be set up, like the folder path to the Petalinux installation or the path to the Vivado hardware file. The script creates a temporary directory that is needed later for the build process of the project. Additionally, the project is created in the current working directory and is configured with the board support package as well as the hardware description for the FPGA.
To access the actual configuration of the linux kernel, the root filesystem, and menu configuration one can use the following commands in listing 3 in the project directory:

```
1  petalinux-config -c kernel  #Opens kernel configuration
2  petalinux-config -c rootfs  #Opens rootfs configuration
3  petalinux-config            #Opens general configuration
```

Listing 3: Commands to configure the PetaLinux Distribution

Figure 7.1 shows the result of executing the „petalinux-config -c kernel“command. It opens the default kernel configuration, which can now be changed and adapted according to the user's needs. For example, different device drivers can be enabled or disabled, and the boot process can be adjusted. Any changes made by a user are stored in a .cfg file which can be found under „project-spec/meta-user/recipes-kernel/files/“.

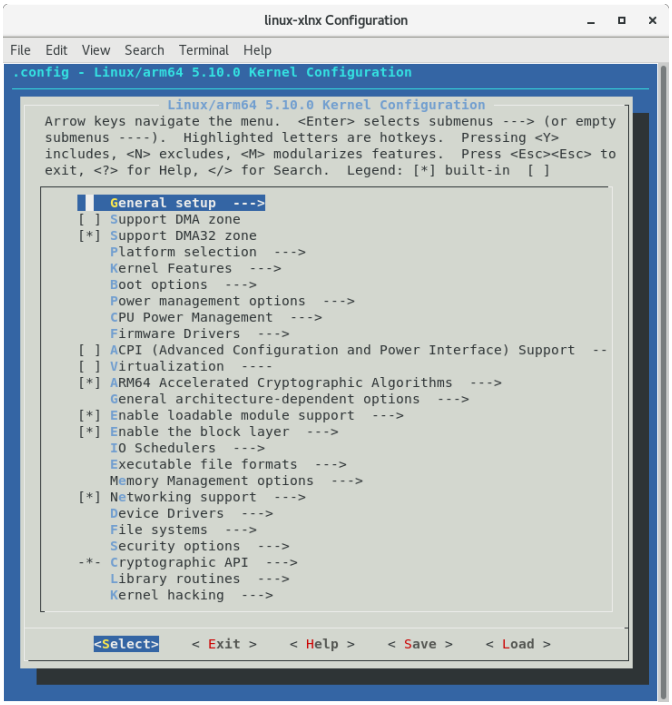


Figure 7.1: Menu for kernel configuration

The configuration menus for the root file system and the general settings look similar to the kernel menu. In the root file system configuration, one can, for example, add different libraries to the distribution e.g. libmetal, or openssh. The general settings menu is in this case used only to change the Media Access Control address of the device so the Dynamic Host Configuration Protocol server can assign an IP address to it. Additionally, new local users can be added to the operating system to avoid accessing the board as the root user.
For this project, the following options were enabled in the root file system configuration:

- **i2c-tools, i2c-dev**, for the I2C-bus
- **libmetal**, to access the hardware
- **tcf-agent, dropbear and openssh-sftp-server**, for debugging purposes

The kernel setting „Userspace IO platform driver with generic IRQ“ was changed from “modular” to “include” to be used by default. This uses the Userspace I/O driver and lets one map the hardware registers, created with the FPGA, to memory in the software. The registers can be easily accessed by the user and no additional driver needs to be developed. Writing to the mapped memory in software will write to registers on the FPGA.
Additionally, the kernel must be configured for using the USB-type A port on the ZCU111 as a serial port and the following device drivers must be enabled:

- USB Serial Console device Support
- USB Generic Serial Driver
- USB Serial Simple Driver

Both general serial drivers as well as some settings to use it as a USB gadget are needed since the board will not be the USB host of the connection. In the USB Support menu, using of the USB port as a serial gadget is enabled and USB Gadget Support option is set to “modular” so that the port can be configured on the run.
The developed FPGA firmware and the USB port must also be visible to the linux kernel. As shown by the device tree in section 7.1.2 (which showed the used USB port), the status of the USB port, as well as the DWC3, which is a SuperSpeed USB 3.0 Dual-Role-Device, is set to „disabled“. The user can add devices to the linux kernel or change some of their settings through a file named „system_user.dtsi“ in the created PetaLinux project folder. To enable the USB port and DWC3, and to configure it as a peripheral, the code shown in listing 4 was added to the file.

```
1 &usb0 {
2    status = "okay";
3 };
4 &dwc3_0 {
5    status = "okay";
6    dr_mode = "peripheral";
7 };
```

Listing 4: USB Device Tree

Adding FPGA registers to the linux kernel is more challenging. The names of the created registers (Verilog File name) must be known as well as their starting address and their names in the Vivado block diagram. Since they will be accessed over the mentioned Userspace I/O driver, their compatible field, which holds by default an artificial driver name specific to this component, must be changed to „generic-uio“. This tells the kernel that the registers are accessed using the Userspace driver. Additionally, the stdout of the Linux operating system is configured to use ttyPS0 with a baud rate of 115200 and will start printing early boot messages thanks to the earlyprintk option.
Similarly to the USB, these components are also added to the „system_user.dtsi“ file as shown in listing 5.

```
/ {
        AXI_Write_32b@a0050000 {
                compatible = "generic-uio";
                status = "ok";
        };
        AXI_Read_32b@a0040000 {
                compatible = "generic-uio";
                status = "ok";
        };
        chosen{
                bootargs = "console=ttyPS0,115200 earlyprintk uio_pdrv_genirq.of_id=generic-uio";
        };
};
&AXI_Write_32b_0 {
        compatible = "generic-uio";
};
&Goertzel_top_AXI_Read_32b_ADC_0 {
        compatible = "generic-uio";
};
```

Listing 5: FPGA registers Device Tree

These settings make it possible to build a PetaLinux Distribution that allows the FPGA registers to be accessed from software. To use the USB connection to exchange data extra configuration is needed while the board is running and booted up with PetaLinux.
To build the project and generate the boot components, which can be copied to an SD-card and inserted in the ZCU111, the following two commands in listing 6 must be executed:

```
petalinux-build
petalinux-package --boot --u-boot --format BIN --force
```

Listing 6: Commands to build PetaLinux project and generate boot components

The SD-card has to be divided into two partitions. Since a 32GB card was used, 1GB was formatted in FAT32 for the boot components while the rest was used for the root file system in an ext4 format.
Following the steps described in this chapter leads to a working PetaLinux distribution that can be booted on the ZCU111. The system-specific software applications which run on the board are described in the following chapter.

Chapter 8

PetaLinux applications

8.1 Measurement application

The application running on PetaLinux contains parts of the BareMetal application described in section 6.3 with some modifications and a USB parser which processes the SCPI commands sent by the LabView software. The application is written in C with every functionality coded in a separate file.
The parser reads a string received over the USB connection, processes it and executes one of the commands listed in table 4.2. Measurement data, status registers for the parser, and constants for the Cordic and Goertzel algorithms are all stored in structures.
The file names and their brief descriptions are gathered in table 8.1.

Table 8.1: PetaLinux application files

Filename	**Description**
main.c	Initialization of all structures and variables, setup of RFDCs, UIO devices and serial USB port
parse.c	Parsing a string input and finding the function to execute
parser_func.c	Collection of functions for each command
measurements.c	Functions to perform a measurement
Cordic-Goertzel.c	Setup functions and enable/disable for the cordic and goertzel algorithm
usb.c	Setup of the USB serial connection
led.c	Control of two LEDs, one indicating the calibrations status and one indicating if the device is ready to use
file_io.c	File operations to save/read calibration data (not used yet)
rfdc_clk.c	Provided by Xilinx; all functionalities to set up ADCs and DACs
rfdc_clk.h	Xilinx header file for rfdc_clk.c
header.h	Structure declarations, function prototypes, constants etc.

The header.h file was created to include all necessary libraries, declare structure definitions, functions prototypes and hold important constants.
The main.c file is the starting point of the application. First, variables are initialized, the FPGA registers are mapped to memory using the Userspace IO driver, and the RF data converters clocks are configured using the rfdc_clk.c and its header file. Additionally ,the serial USB port is initialized and configured using the c library „terminos“ with a baud rate of 115200. Next, a preset function is executed to set all variables to a known state and to turn on the “device ready” LED. Subsequently, the application waits to receive a transaction over the USB connection for 5 ms. If no string is received during this time, the program checks if it is configured in continuous measuring mode or single measuring mode. In the former case, a measurement is performed (which takes around $5ms$ as well), otherwise it waits again for data on the serial connection.
Once a string is received, it is passed over to the parser which extracts data from the string and performs the right operation. An overview of the applications flowchart can be seen in figure 8.1

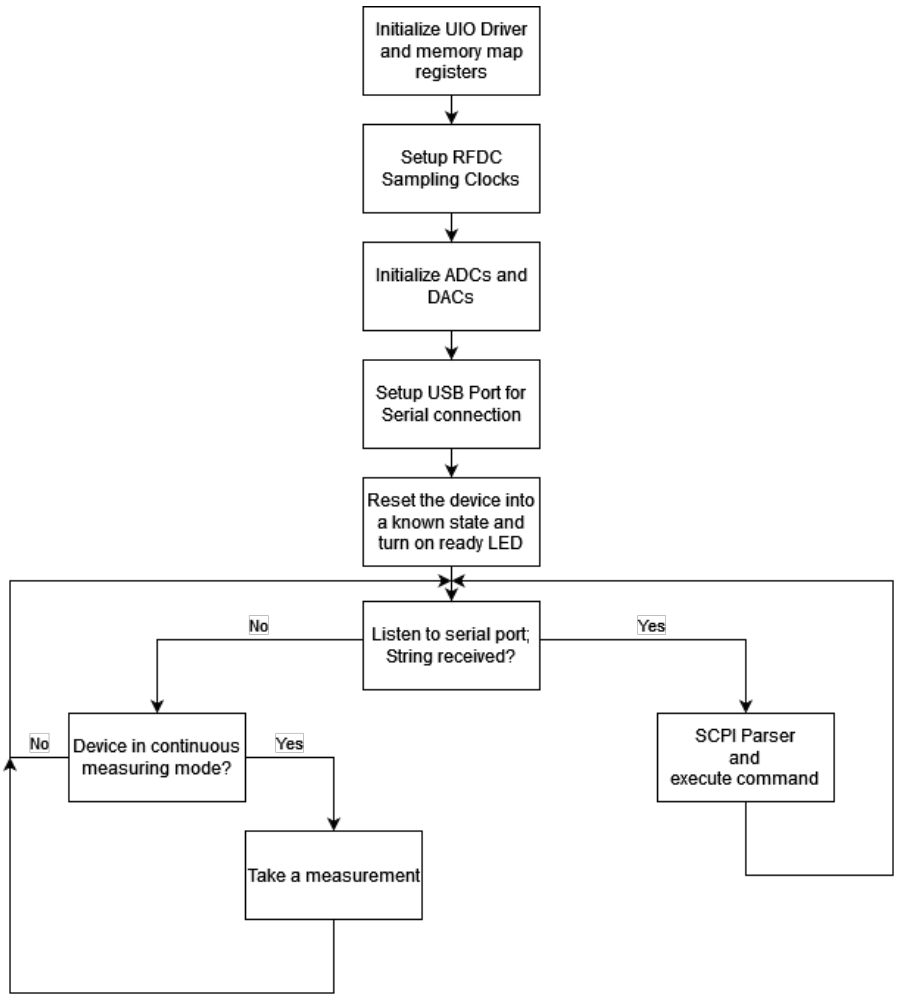


Figure 8.1: Flowchart of the PetaLinux application

8.1.1 Create a PetaLinux application

To create an application in the PetaLinux environment, the following command must be executed in the PetaLinux project folder:

```
1 petalinux-create -t apps --name vna-test --template c --enable
```

Listing 7: Application creation in PetaLinux

This creates an application „vna-test“. The template flag specifies it as a C application and it is directly enabled. By default PetaLinux creates a „Hello World“ project. The whole application is created inside a folder structure with a .bb file (a BitBake append file) that modifies the recipe file which can configure, compile, and deploy a given piece of software. In the same folder, there exists a „files“ folder that contains the .c files and the makefile. Any additional C files have to be moved into the „files“ folder. The makefile as well as the .bb file must be modified to include all C files. The makefile includes also additional libraries used in the application, for example, the math and libmetal libraries. Additional dependencies for internal Xilinx IPs such as the RFDC might also be required. These are added in the makefile and the BitBake file. Once everything is set up, the project can be built again following listing 6. The application can then be executed in PetaLinux by running „vna-test“in the console if not specified otherwise in the .bb file.

8.2 Boot script

The application should run automatically on system startup to make it an independent measurement instrument. The USB port also needs to be configured as a serial port while the board is running. Both of these goals can be achieved with a boot script that is automatically executed when the boot process has finished.
A shell script similar to an application in PetaLinux is created with the template changed from „C“to „install“

```
petalinux-create -t apps --name usbboot --template install --enable
```

Listing 8: Boot script creation in PetaLinux

The generated „usbboot.sh“ file can be modified as needed. The USB port requires different kernel module files to be inserted into the Linux kernel during run time to configure it as a serial port. These modules are part of the gadget driver and can be found in the root file system under „/lib/modules/5.10.0-xilinx-v2021.2/kernel/drivers/usb/gadget/“. To include these modules, the code in listing 9 has to be added to the shell script.

```
insmod /lib/modules/5.10.0-xilinx-v2021.2/kernel/drivers/usb/gadget/libcomposite.ko
insmod /lib/modules/5.10.0-xilinx-v2021.2/kernel/drivers/usb/gadget/function/u_serial.ko
insmod /lib/modules/5.10.0-xilinx-v2021.2/kernel/drivers/usb/gadget/function/usb_f_serial.ko
insmod /lib/modules/5.10.0-xilinx-v2021.2/kernel/drivers/usb/gadget/function/usb_f_acm.ko
insmod /lib/modules/5.10.0-xilinx-v2021.2/kernel/drivers/usb/gadget/legacy/g_serial.ko
```

Listing 9: Modules for the serial USB

The application is started by executing „/usr/bin/vna-test“in the script.
Finally, the BitBake file needs to be changed as well to execute the shell script after startup of the board. Therefore, the code in listing 10 has to be added to the file [19].

```
inherit update-rc.d
INITSCRIPT_NAME = "usbboot"
INITSCRIPT_PARAMS = "start 99 5 ."
```

Listing 10: Initialisation parameters of the shell script

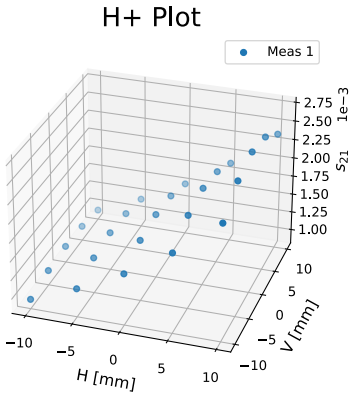
Chapter 9

Measurements

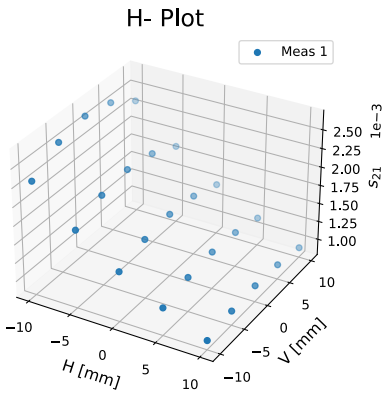
Various measurements were taken to validate the functionality of the device in combination with the physical structure of the stretched-wire test bench.
For the first test, the RFSoC was connected to the test bench to measure a PU and check if the obtained data fits the theoretically expected curve.
For the second test, the same PU was measured multiple times to validate the repeatability of the setup.
In the third test, the new setup was compared to the old one by measuring a BPM first with the RFSoC and then with the VNA connected to the test bench.

9.1 Single time measurement of a Pick-Up

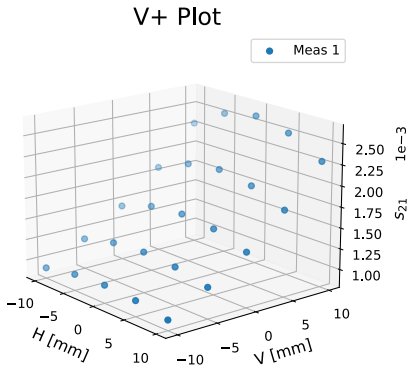
A four electrode button BPM was installed in the test bench. The wire was programmed to move to 25 positions on an equally spaced 20 mm by 20 mm grid. This results in a distance of 5 mm between two points on the grid in the horizontal and vertical direction of the BPM. The measurement results of the calculated s_{21} parameter are plotted in a linear scale for each electrode in figure 9.1. The measurements were taken without any additional signal amplification at a frequency of 30 MHz. Additionally, the RFSoC took eight measurements at a single position and returned the average of these eight values. The time it took to perform this measurement was around 10 min for measuring the 25 points and a few extra minutes at the beginning for the calibration routine.
The 3D plots of the four electrodes show that the closer the wire moves to one electrode, the greater the forward transmission s_{21} is. As expected, the opposite is happening when the wire moves away from the electrode. Additionally, a small curvature can be seen when the wire is moved in parallel to an electrode. This movement also increases the distance to the electrode and therefore results in a lower forward transmission.
The resulting maximum and minimum values are approximately 0.00265 (−51.54 dB) and 0.0009 (−61 dB), respectively.



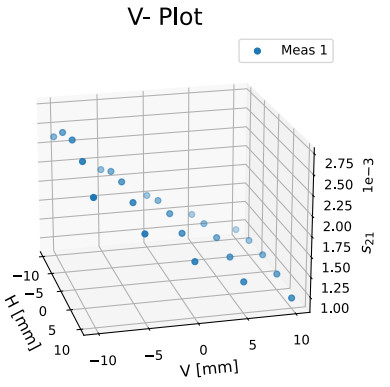
(a) H+ plot for a single measurement



(b) H- plot for a single measurement



(c) V+ plot for a single measurement



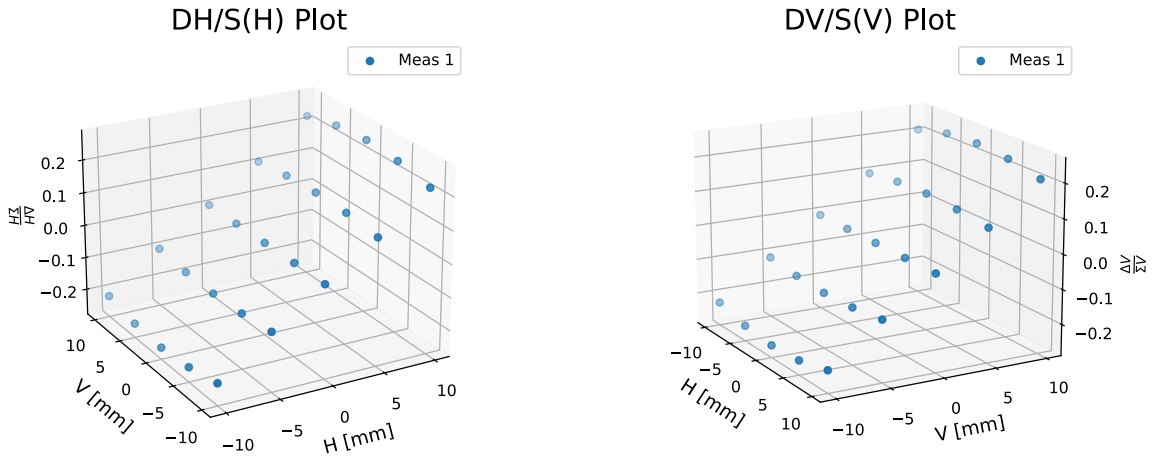
(d) V- plot for a single measurement

Figure 9.1: BPM measurement using the RFSoC

Using these measurements, the Delta-Sigma normalization can now be calculated for the horizontal and vertical plane.
The two calculated curves are plotted in figure 9.2. At first, both look like a flat plane, but after a closer inspection, one can see slight rounding of the plane, especially for large displacements from the center. Since the sensitivity and offset of the Delta-Sigma normalisation curve need to be known, a 2D line can be fitted to the curve. This is done for the on-axis where $V = 0$mm in $\frac{\Delta H}{\Sigma H}$ and $H = 0$mm in $\frac{\Delta V}{\Sigma V}$. The slope of the fitted line defines the sensitivity while the axis-crossing point is the measured offset δ between the electrical and the mechanical center.
The calculated sensitivities and offsets are listed in table 9.1. Ideally, the sensitivity should be equal for both planes, which is almost the case, and the offsets should both be zero.

Table 9.1: Measured BPM parameters

	Sensitivity [mm^{-1}]	Offset [mm]
H	0.0245	-0.2294
V	0.0244	0.382



(a) Delta-Sigma of the horizontal plane

(b) Delta-Sigma of the vertical plane

Figure 9.2: Delta-Sigma Normalization measurement using the RFSoC

9.2 Multiple measurements of the same Pick-Up

The measurements described in section 9.1 were repeated multiple times for the same pick-up to verify the repeatability and accuracy of the measurement. The measurements were taken on the same day, back to back, to avoid any external influences.
The single electrode measurements can be seen in figure 9.3. All measurement points for the electrodes lie very close to each other. This indicated that the system has a low noise floor with the ADCs and DAC of RFSoC.

Figure 9.4 illustrates the Delta-Sigma normalization plot for the two planes. The low noise level is also seen in the Delta-Sigma normalization. The calculated slopes and offsets for the BPM are summarised in table 9.2 for each of the five measurements.

Table 9.2: BPM Parameters of multiple measurements using the RFSoC

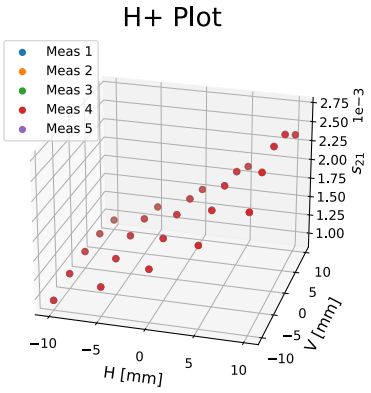
(a) Horizontal Parameters

	Sensitivity [mm^{-1}]	Offset [mm]
H_1	0.02457	-0.23
H_2	0.02459	-0.227
H_3	0.02456	-0.222
H_4	0.02456	-0.211
H_5	0.0246	-0.229

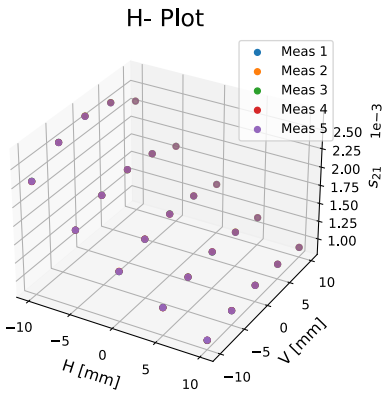
(b) Vertical Parameters

	Sensitivity [mm^{-1}]	Offset [mm]
V_1	0.02436	0.429
V_2	0.02432	0.418
V_3	0.02431	0.383
V_4	0.02434	0.416
V_5	0.0244	0.382

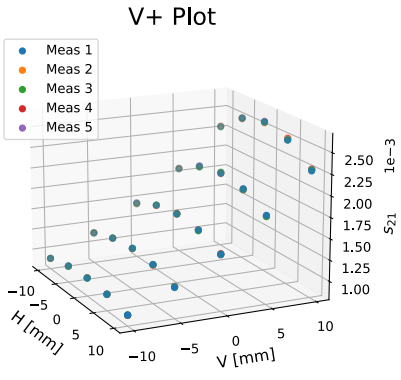
In the horizontal plane, the sensitivity of the PU remains virtually the same over the five measurements. The maximum deviation of the sensitivity is 4×10^{-5} mm^{-1} which proves high repeatability. The δ_H offset values are all within 0.019 mm which would determine the offset with an accuracy of roughly 20 µm.



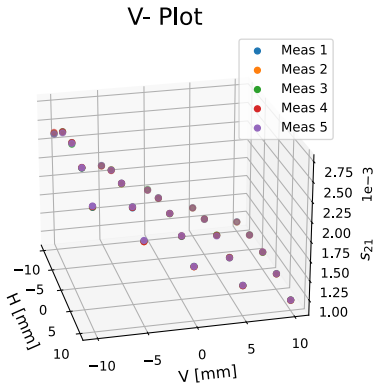
(a) H+ plot of RFSoC measurements



(b) H- plot of RFSoC measurements

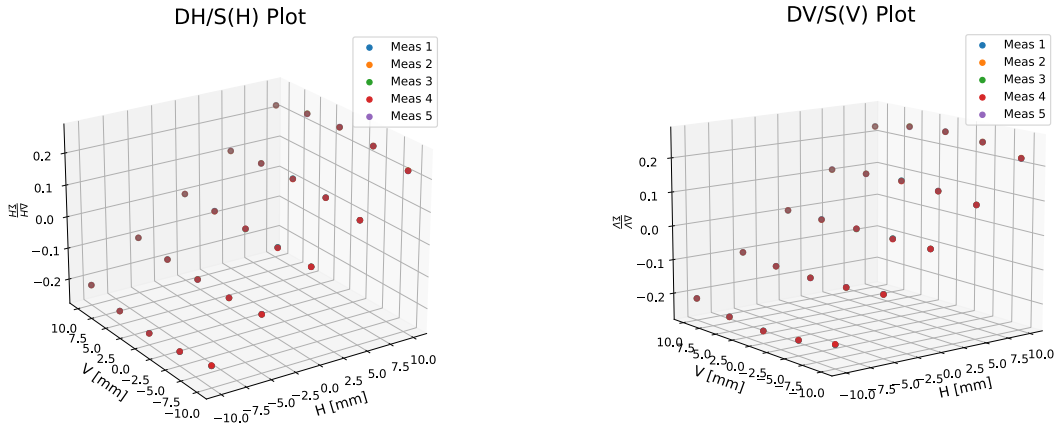


(c) V+ plot of RFSoC measurements



(d) V- plot of RFSoC measurements

Figure 9.3: Multiple measurements with the RFSoC



(a) Delta-Sigma of horizontal plane

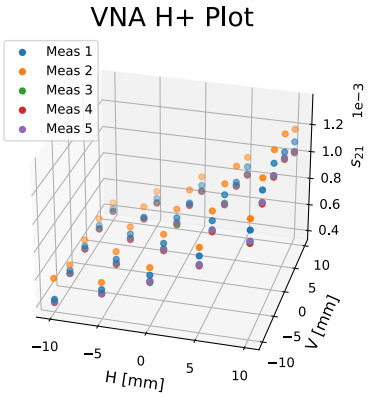
(b) Delta-Sigma of vertical plane

Figure 9.4: Delta-Sigma Normalization of multiple measurements using the RFSoC

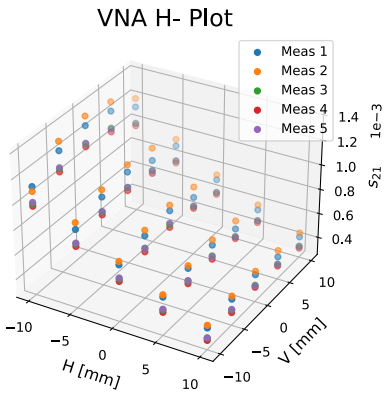
The sensitivity measured in the vertical plane is close to the one on the horizontal plane and all measured values are within 1.3×10^{-4} which is slightly higher than in the other plane. The same behaviour can be seen for the offset δ_V which has a spread of around 47 µm. Nevertheless, all the measurements are very close to each other. This leads to the conclusion that the setup can perform measurements repeatably with the same results.

9.3 Comparison of the old and new setup

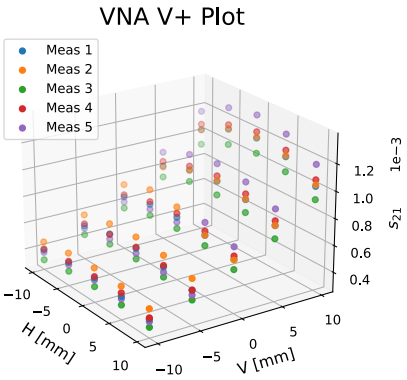
To compare the new setup with the old VNA, five measurements were taken with the old VNA and compared with the measurements taken in section 9.2. The five measurements are illustrated in figure 9.5. They were taken in the same conditions as when the RFSoC was used, i.e. at a frequency of 30 MHz, with 25 measuring points arranged in a grid of ±10 mm for the horizontal and vertical position. Performing a single measurement of the 25 point grid once, with the VNA took around 25 min with an additional few minutes for the calibration routine.
The spread in s_{21} measurements of each electrode is much higher than the in the measurements taken with the RFSoC and shown in section 9.2. This suggests that the VNA has a worse noise floor than the RFSoC.
The effect of the higher noise in the VNA measurement can also be seen in the Delta-Sigma normalization for each plane. The calculated points are relatively widely spread as it can be seen in figure 9.6 which shows the normalization for the horizontal and vertical plane of the VNA measurement.



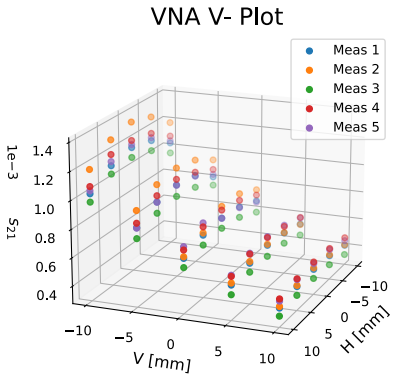
(a) H+ plot of VNA measurements



(b) H- plot of VNA measurements

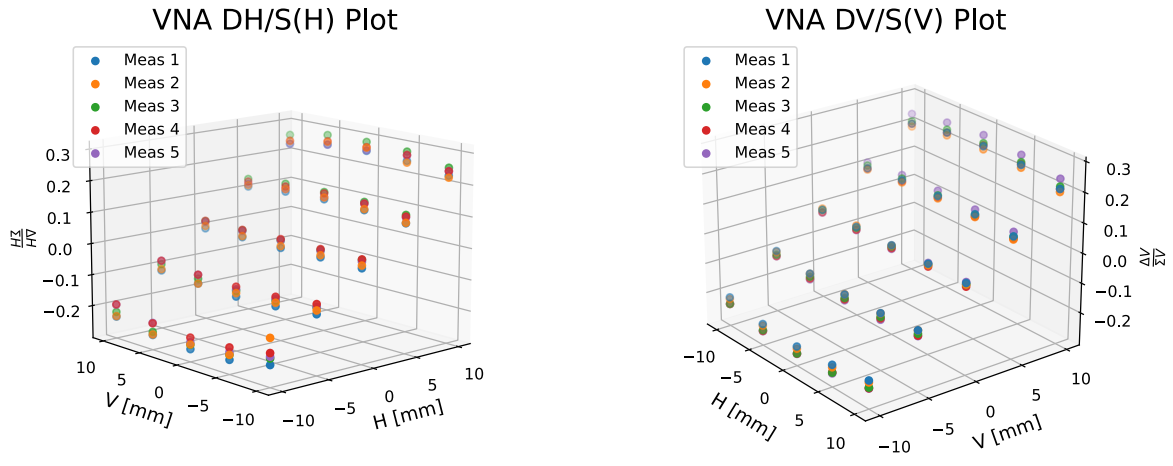


(c) V+ plot of VNA measurements



(d) V- plot of VNA measurements

Figure 9.5: Multiple measurements with the VNA used in the old setup



(a) Delta-Sigma of the horizontal plane

(b) Delta-Sigma of the vertical plane

Figure 9.6: Delta-Sigma Normalization of measurements using the VNA

As a result of that, it is expected, that the calculations for the slope and offset will also be worse than the ones taken with the RFSoC. Table 9.3 summarises the values calculated for the five measurements with the old setup.

Table 9.3: BPM parameters based on multiple measurements using the VNA

(a) Horizontal Parameters

	Sensitivity [mm^{-1}]	Offset [mm]
H_1	0.02586	0.176
H_2	0.02559	-0.197
H_3	0.02655	-0.66
H_4	0.02464	-0.849
H_5	0.02473	-0.45

(b) Vertical Parameters

	Sensitivity [mm^{-1}]	Offset [mm]
V_1	0.024	0.19
V_2	0.02363	0.421
V_3	0.02558	0.387
V_4	0.02526	0.589
V_5	0.02718	0.182

By comparing the values in tables 9.2 and 9.3 as well as looking at the figures in sections 9.2 and 9.3, one can clearly see a difference in measurement precision of the two setups. The standard deviation of the offset measured with the RFSoC is 51 times lower than the one measured with the VNA in the horizontal plane and around 8 times lower in the vertical plane. The accuracy improvement can also be seen in the sensitivity measurement. Whereas the standard deviation of the sensitivity for the VNA setup is around 7.2×10^{-4} in the horizontal plane and 1.3×10^{-3} in the vertical plane, the deviations with the newly developed setup are 1.6×10^{-5} and 3.2×10^{-5} respectively. Both standard deviations are therefore reduced by a factor of roughly 40.
The amount of time the operator of the test bench is occupied with the measurements is reduced by a factor of around 2.5 for these measurements. Additionally, the VNA could only measure two electrodes at once, while the RFSoC can measure up to eight in parallel. For BPMs with more electrodes, the new electronics will have an even greater time saving factor than 2.5.

To end the project and keep the RFSoC safe from external influences, it has been mounted inside a rack unit case. Besides the better performance advantages of the new device, it is also much smaller than the old setup. The two instruments can be seen In figure 9.7, with the VNA at the bottom and the RFSoC, packed in a case, on top of it.

Figure 9.7: Old VNA and RFSoC in a rack case

Chapter 10

Conclusion

This thesis describes the development of a new measurement setup for a stretched-wire test bench to calibrate different kinds of BPMs. A signal generation and acquisition platform has been created on the Xilinx Ultrascale+ RFSoC ZCU111 evaluation board. The FPGA of the device is responsible for the signal generation using a Coordinate Rotational Digital Computer algorithm. A sine signal can be created up to 1 GHz with variable output power. The acquisition of data is managed by 8 ADCs running at a sampling frequency of 2.048 GSPS and acquiring signals in windows of 128 µs. The data is processed in the FPGA with the Goertzel algorithm to determine a single frequency bin of a DFT, which can be used to calculate the received power at the specified frequency applying a periodogram. Measurements taken with the device outperformed the old measurements taken with the VNA setup regarding the noise floor. Fewer variations in the measured s_{21}-parameter and the resulting Delta-Sigma normalisation occur, as presented in sections 9.2 and 9.3. Additionally, the measurement time for a four port BPM has been reduced by a factor of 2.5.
All in all, the developed device fulfills the set requirements to upgrade the existing stretched-wire test bench and improves its measurement accuracy. Additional changes can be implemented to further improve the device. For example, the acquired values can be windowed before calculating the DFT to reduce spectral leakage. Since all the values are currently stored inside the FPGAs Block RAM, they could be transferred to the FPGAs external RAM.
The developed device has become a crucial part of the existing stretched-wire test bench used at CERN to calibrate BPMs before their installation in the accelerator tunnels. In the near future, it will be used to test a new set of BPMs which are currently under construction for an upcoming upgrade of the LHC.

Bibliography

[1] CERN, *Our History*. https://home.cern/about/who-we-are/our-history, 2022.

[2] ——, *Accelerators*. https://home.cern/science/accelerators, 2022.

[3] ——, *Pulling together: Superconducting electromagnets*. https://home.cern/science/engineering/pulling-together-superconducting-electromagnets, 2022.

[4] ——, *The accelerator complex*. https://home.cern/science/accelerators/accelerator-complex, 2022.

[5] P. Forck, P. Kowina, and D. Liakin, *Beam position monitors*. Gesellschaft für Schwerionenforschung GSI, Darmstadt, Germany, 2008.

[6] M. Wendt, *BPM Systems: A brief Introduction to Beam Position Monitoring*. Proceedings of the 2018 CERN-Accelerator-School course on Beam Instrumentation, Tuusla, (Finland), 2020.

[7] L. H. Crockett, D. Northcote, C. Ramsay, F. D. Robinson, and R. W. Stewart, *Exploring Zynq MPSoC*. Strathclyde Academic Media, 2019.

[8] Xilinx Inc., *Understanding Key Parameters for RF-Sampling Data Converters (WP509) v1.0*. Xilinx Inc., 2019.

[9] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Elsevier Science (USA), 2003.

[10] D. L. Jones, *Digital Signal Processing: A User's Guide*. 2006.

[11] P. Sysel and P. Rajmic, *Goertzel algorithm generalized to non-integer multiples of fundamental frequency*. EURASIP Journal on Advances in Signal Processing, 2012.

[12] E. Jacobsen and R. Lyons, *The Sliding DFT*. IEEE Signal Processing Magazine, 2003.

[13] F. Quint, *Signal- und Parameterschätzung*. Hochschule Karlsruhe für Technik und Wirtschaft, 2021.

[14] Xilinx, *Zynq UltraScale+ MPSoC Embedded Design Tutorial*. https://xilinx.github.io/Embedded-Design-Tutorials/docs/2022.1/build/html/docs/Introduction/ZynqMPSoC-EDT/8-boot-and-configuration.html, 2022.

[15] ——, *Zynq UltraScale+ MPSoC Software Developer Guide (UG1137)*. Xilinx, 2021.

[16] ——, *UG1144 PetaLinux Tools Documentation*. Xilinx, 2021.

[17] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*. O'Reilly Media, 2005.

[18] The Linux Information Project (LINFO), *Kernel Definition*. The Linux Information Project http://www.linfo.org/kernel.html, 2005.

[19] X. Wiki, *PetaLinux Yocto Tips*. Xilinx Wiki https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips, 2022.

Affidavit

I hereby certify that I have written this Master's thesis independently without the help of third parties and without using any sources or aids other than those indicated.
I have indicated all passages in the thesis that are taken from printed works or sources from the Internet, either in wording or in meaning, by citing the sources. This also applies to all illustrations.
The submitted work has not been the subject of any other examination procedure, neither in its entirety nor in essential parts.
I am aware that plagiarism is serious academic misconduct that will be reported to the examination board and will result in sanctions.
Furthermore, I assure that the electronic version of the Master's thesis corresponds to the printed version.

Karlsruhe, 06.03.2023 _____

Henrik Dötsch