FPGA-based Tomography of Propagating Quantum Microwaves

Master’s thesis
Robert Neagu
Advisor: Prof. Dr. Rudolf Gross
Garching – October 23, 2019
## Contents

1 Introduction  

2 Theoretical foundations  
   2.1 Propagating quantum microwaves  
      2.1.1 Gaussian quantum states  
      2.1.2 Tomography methods  
      2.1.3 Intensity correlation function  
   2.2 Quantum communication with propagating squeezed microwaves  
   2.3 Digital signal processing  
      2.3.1 Finite impulse response filter  
      2.3.2 Fixed-point number representation  
      2.3.3 Field-programmable gate array (FPGA)  

3 Experimental setup  
   3.1 Analog-to-digital-converter setup  
      3.1.1 Field programmable gate array (FPGA)  
      3.1.2 Trigger scheme  
      3.1.3 Acqiris card  

4 Architecture of the image  
   4.1 Building blocks  
      4.1.1 Channel balancing  
      4.1.2 IQ demodulation  
      4.1.3 Finite impulse response filter  
      4.1.4 Moment calculation  
      4.1.5 Averaging technique  
   4.2 Communication to PC  
   4.3 FPGA limitations and solutions
5 FPGA benchmarking 55
  5.1 Self-reference benchmarking .......................... 55
    5.1.1 Pulsed measurements ............................... 55
    5.1.2 Quadrature statistics .............................. 56
    5.1.3 Phase stability ................................... 59
    5.1.4 Wigner function reconstruction .................... 60
    5.1.5 Second-order correlation function ................. 61
  5.2 FPGA versus Acqiris benchmarking ..................... 63
    5.2.1 Comparison with Acqiris card ........................ 63

6 Summary and outlook 67
Chapter 1

Introduction

Quantum state reconstruction is one of the central issues in optical cavity electrodynamics (cQED) and circuit quantum electrodynamics (circuit QED)\[1]. Both fields investigate the physics of light-matter interaction between natural atoms and optical photons, or artificial atoms and microwave photons, respectively. Through this interaction, generation of highly nonclassical states of light is possible. In particular, superconducting circuits allow for generation of such quantum states at the frequencies of several GHz. The state tomography of such propagating quantum states requires advanced reconstruction methods, since compared to the optical regime, the energy of a microwave photon is lower by a factor of $10^{5}$. This energy difference underlies the absence of single photon detectors in the microwave regime. Therefore, alternative quantum state detectors and related state reconstruction techniques must be used in the microwave regime. One of them is the so-called reference state reconstruction method which uses a known state, such as the vacuum state, for calibration against an unknown noisy quantum signal. This method allows to efficiently subtract the noise during the data processing and obtain a genuine state tomogram of this unknown quantum signal.

In order to implement the aforementioned reference state reconstruction, one needs to perform several technical operations. The latter include microwave signal downconversion, filtering, and calculation of statistical moments of field quadratures. For these technical operations we are going to develop a specific detector, based on a field programmable gate array (FPGA), which allows one to process experimental data in real-time. In the end, the statistical moments can be exploited in order to construct the quasiprobability distribution of the quantum state - the Wigner function. Since we are going to work only with Gaussian states, moments up to the second order are needed to describe their Wigner function, and up to the fourth order to check their Gaussianity.

Furthermore, with the development of fast FPGAs, we are building a foundation
for the next line of advanced quantum communication [2] experiments with propagating quantum microwaves - such as quantum teleportation. For quantum teleportation one needs to generate a feedforward signal in order to teleport an unknown quantum state. This feedforward signal must be generated within a short time-window, typically, less than $1 \mu s$ due to the fragile and short-lived nature of quantum correlations. Thus, we’re going to need device capable of fast state tomography and fast feedforward signal generation. Modern FPGAs allow us, in principle, to fulfil both criteria.

In the present thesis, we implement an FPGA-based reference state tomography of propagating microwaves. In the past, an analog-to-digital converter (ADC) card was used for the same purpose. A crucial difference with the ADC card is that it is incapable of internal data processing (such as moment calculation), therefore, the latter had to be implemented after the data acquisition on a standard PC. The process of data transfer between the ADC card and the PC takes a lot of time and the PC-based data processing itself is relatively slow due to a low number of parallel processing units (typically, 7-8). In this regard, a detector based on the FPGA can significantly improve the measurement time by analyzing the microwave signal directly on the card, without sending it to the computer which is one of the limiting factors when using the ADC card. Another advantage of the FPGA is due to its large number of parallel processing units as compared to a standard PC (up to $10^5$!) which additionally allows to speed up the state tomography process.

Main goals of this thesis are (i) development of an FPGA image for microwave state tomography, (ii) experimental benchmarking of the FPGA image, (iii) experimental comparison of the FPGA-based tomography against the ADC card-based tomography, (iv) providing a plan on further improvements of the FPGA code.

This work is structured as follows: Chapter 2 provides basic theory which is necessary for understanding of measurement technique, as well as basic concepts in digital signal processing, such as digital filtering and fixed-point arithmetic. Chapter 3 presents a short description of the experimental setup we use. Chapter 4 gives an insight into the FPGA code developed for the microwave state tomography. Additionally, limitations and possible solutions for the FPGA card are discussed. Chapter 5 presents experimental measurements and benchmarking with the FPGA card and its comparison to the ADC card. Last chapter 6 gives a summary of the whole thesis and an outlook.
Chapter 2

Theoretical foundations

In this chapter we present a theory basis for this work. First, we introduce the concept of propagating quantum microwaves and how they can be described in terms of a quasi-probability distribution, namely the Wigner function. Next, we discuss certain important states, i.e. thermal, coherent, and squeezed states. This is followed by a description of the state tomography based on the reference state reconstruction method.

Second, we consider several important quantities for quantum communication protocols, such as entanglement and the second-order correlation function $g^{(2)}$. In addition we discuss the quantum teleportation protocol based on propagating microwaves. Finally, a theory basis of digital signal processing is presented, such as the concepts of fixed-point number representation, digital filtering, and field programmable gate arrays (FPGA).

2.1 Propagating quantum microwaves

Microwave radiation corresponds to the electromagnetic frequency spectrum between 300 MHz and 300 GHz, which yields wavelengths between 1 m and 1 mm. A single-mode classical microwave signal $A(t)$ can be described as

$$A(t) = A \sin (\omega t + \phi), \quad (2.1)$$

where $A$ is the amplitude of the signal, $\omega = 2\pi f$ is the angular frequency, $f$ is the frequency of the signal, and $\phi$ is the phase. An analogous description can be given by decomposing $A(t)$ into the in-phase and out-of-phase quadratures, $I(t)$
2.1 Propagating quantum microwaves

and $Q(t)$ respectively

$$
A(t) = A \sin (\omega t + \phi) \\
= A \cos (\phi) \cos (\omega t) + A \sin (\phi) \sin (\omega t) \\
= I(t) \cos (\omega t) + Q(t) \sin (\omega t),
$$

(2.2)

with $I(t) = A \cos (\phi)$, $Q(t) = A \sin (\phi)$, $A = \sqrt{I(t)^2 + Q(t)^2}$, and $\phi = \arctan (Q(t)/I(t))$.

For a quantum-mechanical description of the propagating microwave signal we introduce the bosonic creation and annihilation operators $\hat{a}^\dagger$ and $\hat{a}$, respectively, with the commutator $[\hat{a}, \hat{a}^\dagger] = 1$. Additionally, we also introduce the quadrature operators $\hat{q}$ and $\hat{p}$ and their relation to the annihilation and creation operators

$$
\hat{p} = \frac{1}{2i} (\hat{a} - \hat{a}^\dagger), \quad \hat{q} = \frac{1}{2} (\hat{a} + \hat{a}^\dagger), \quad [\hat{q}, \hat{p}] = \frac{i}{2},
$$

(2.3)

with $i$ being the imaginary unit. In contrast to classical mechanics, in quantum mechanics we have to take into account additional properties such as the Heisenberg uncertainty relation for two complementary variables

$$
\Delta \hat{q} \cdot \Delta \hat{p} \geq \frac{1}{4},
$$

(2.4)

where the standard deviation $\Delta \hat{O}$ of an observable $\hat{O}$ is given by

$$
(\Delta \hat{O}) = \sqrt{\langle \hat{O}^2 \rangle - \langle \hat{O} \rangle^2}.
$$

(2.5)

There are different methods to describe a quantum state, with one of the most used ones being via the quasi-probability distribution function, or the Wigner function [4]. The Wigner function represents a quantum state in the phase space and is defined as

$$
W(q,p) = \frac{1}{\pi \hbar} \int \langle q-y | \hat{\rho} | q+y \rangle e^{2i\pi y/\hbar} dy,
$$

(2.6)

with $q$ and $p$ being the amplitudes of the quadratures and $\hat{\rho}$ is the density operator. The Wigner function maps an arbitrary quantum state onto the phase space. We can also define the Wigner function by introducing the symmetrically ordered characteristic function [1]

$$
\xi = \text{Tr} \left\{ \hat{\rho} \hat{D}(\eta) \right\} = \text{Tr} \left\{ \hat{\rho} e^{\eta \hat{a}^\dagger - \eta^* \hat{a}} \right\}.
$$

(2.7)
Here we introduced the displacement operator $\hat{D}(\eta) = \exp \left( \eta \hat{a}^\dagger - \eta^* \hat{a} \right)$ with $\eta$ being a complex displacement amplitude. The characteristic function returns an expectation value of the displacement operator. By Fourier transforming the characteristic function we can obtain the Wigner function

$$W(\alpha) = \frac{1}{\pi^2} \int \exp (\eta^* \alpha - \eta \alpha^*) \xi (\eta) d^2\eta , \quad (2.8)$$

where $\alpha$ is also a complex amplitude. By defining $\alpha = q + ip$, the Eq. (2.8) is also a map of our quantum state in the phase space. The knowledge of the signal moments $\langle (\hat{a}^\dagger)^m \hat{a}^n \rangle$ with $m,n \in \mathbb{N}_0$ allows us to calculate the Wigner function since we can rewrite the characteristic function as follows

$$\xi (\eta) = e^{-|\eta|^2/2} \sum_{m,n} \frac{\langle (\hat{a}^\dagger)^m \hat{a}^n \rangle}{m! n!} \eta^m (-\eta^*)^n . \quad (2.9)$$

We can insert the characteristic function $\xi (\eta)$ into Eq.(2.8) and obtain

$$W (\alpha) = \sum_{m,n} \frac{\langle (\hat{a}^\dagger)^m \hat{a}^n \rangle}{\pi^2 m! n!} \int \eta^m (-\eta^*)^n \exp \left( -\frac{|\eta|^2}{2} + \eta^* \alpha - \eta \alpha^* \right) d^2\eta . \quad (2.10)$$

Therefore, by knowing the complete moment matrix, we can obtain the full quantum description of any arbitrary state. For many applications, such as the reconstruction of Gaussian quantum states, we only need the information of moments up to the second order, $0 < m + n \leq 2$. The Wigner function of a Gaussian state can be written as

$$W(q,p) = \frac{1}{\pi \sqrt{(\nu + 1/2)^2 - |\mu|^2}} \times \exp \left( -\frac{(\nu + 1/2) |\xi - \langle \hat{a}^2 \rangle| - (\mu^*/2) (\xi - \langle \hat{a} \rangle)^2 - (\mu/2) (\xi^* - \langle \hat{a}^\dagger \rangle)^2}{(\nu + 1/2)^2 - |\mu|^2} \right) , \quad (2.11)$$

where $\xi = q + ip$, $\mu = \langle \hat{a}^2 \rangle - \langle \hat{a} \rangle^2$, and $\nu = \langle \hat{a}^\dagger \hat{a} \rangle - |\langle \hat{a} \rangle|^2$. The amplitude $|\xi|^2 = q^2 + p^2$ gives mean the number of photons in the Gaussian quantum state.

### 2.1.1 Gaussian quantum states

In this section, we introduce a variety of important quantum microwave states, as well as their basic properties and their phase-space representation in terms of
2.1 Propagating quantum microwaves

Wigner functions.

**Thermal states**

A thermal state describes incoherent radiation from a black body at a finite temperature $T$. The thermal density matrix can be expanded in terms of Fock states as

$$ \hat{\rho}_{\text{th}} = \sum_{n=0}^{\infty} p_n |n\rangle \langle n|, \quad (2.12) $$

with $|n\rangle$ being a Fock state for a certain mode of the field [5]. The thermal state density matrix also obeys the Boltzmann distribution

$$ \hat{\rho}_{\text{th}} = \frac{e^{-\hat{n} \omega/(k_B T)}}{\text{Tr} \left[ e^{-\hat{n} \omega/(k_B T)} \right]} , \quad (2.13) $$

where $\omega$ is the frequency of the field mode, $\hat{n}$ is the photon number, $k_B$ is the Boltzmann constant, and $\hbar$ is the reduced Planck constant. Using the geometric sum

$$ \sum_{k=0}^{\infty} q^k = \frac{1}{1-q} , \quad (2.14) $$

with $q = e^{-\hbar \omega/(k_B T)}$, we can evaluate the average number of photons in a thermal state in the mode $\omega$

$$ \langle \hat{n} \rangle = \text{Tr} \left[ \hat{\rho}_{\text{th}} \hat{n} \right] = \frac{1}{e^{\hbar \omega/(k_B T)} - 1} , \quad (2.15) $$

which gives us the Planck distribution. By using Eq.(2.5) and Eq.(2.15) we can retrieve the moments of a thermal state

$$ \langle (\hat{a}^\dagger)^m \hat{a}^n \rangle_{\text{th}} = m! \langle \hat{n} \rangle^m \delta_{m,n} , \quad (2.16) $$

where $m,n \in \mathbb{N}_0$ and $\delta_{m,n}$ is the Kronecker delta function. Thermal states are important to consider since they describe the thermal noise in the measurement setup. Low thermal population at microwave frequencies is achieved by cooling down the environment to millikelvin temperatures. For example, the $\langle \hat{n} \rangle = 2$ thermal state at the frequency $\omega = 2\pi \cdot 5 \text{ GHz}$ requires the environmental temperature of $T = 592 \text{ mK}$.

We can retrieve the Wigner function for the thermal state (depicted in Fig.
2.1b) by inserting the moments from Eq.(2.16) into the expression Eq.(2.11) which gives us

\[ W_{th}(q,p) = \frac{1}{\pi \langle \hat{n} \rangle + 1/2} \exp \left( -\frac{q^2 + p^2}{\langle \hat{n} \rangle + 1/2} \right). \]  

(2.17)

Figure 2.1: Examples of Wigner functions of important Gaussian states. (a) the vacuum state \( |0 \rangle \). (b) thermal state with photon number \( \langle \hat{n}_{th} \rangle = 5 \). (c) coherent state with the amplitude \( |\alpha|^2 = 5 \) and angle \( \theta = 45^\circ \). (d) squeezed state with \( r = 1 \) and \( \phi = 180^\circ \).
Coherent states

The coherent state $|\alpha\rangle$ is created by applying the displacement operator $\hat{D}(\alpha)$ to the vacuum state $|0\rangle$ (the Wigner function of the vacuum state is depicted in Fig. 2.1a)

$$|\alpha\rangle = \hat{D}(\alpha)|0\rangle ,$$

(2.18)

here $\alpha = |\alpha| \exp (i\Theta)$ and we define the phase $\theta = \pi/2 - \Theta$ as the angle between the displacement direction and the $p$-axis. Furthermore, a coherent state is a state with minimal quadrature fluctuations $(\Delta q)^2 = (\Delta p)^2 = 1/4$.

Moreover, the coherent state is an eigenfunction of the annihilation operator, $\hat{a}|\alpha\rangle = \alpha|\alpha\rangle$, from which we can retrieve the normally ordered moments $\langle \alpha| (\hat{a}^\dagger)^m \hat{a}^n |\alpha\rangle = (\alpha^*)^m \alpha^n$.

The Wigner function for a coherent state $|\alpha\rangle = |Q + iP\rangle$ (depicted in Fig. 2.1c) is

$$W(q,p) = \frac{2}{\pi} \exp \left(-2 \left((q - Q)^2 + (p - P)^2\right)\right) .$$

(2.19)

Squeezed states

The quadrature variances for the coherent and vacuum states are equal and minimal $(\Delta q)^2 = (\Delta p)^2 = 1/4$. Another minimum uncertainty state is a single-mode squeezed state for which the product of its quadratures fluctuations is limited by the Heisenberg uncertainty. We introduce the squeeze operator

$$\hat{S}(\xi) = \exp \left(\frac{1}{2} \xi \hat{a}^\dagger \hat{a} - \frac{1}{2} \xi^* (\hat{a}^\dagger)^2\right) ,$$

(2.20)

with the complex amplitude $\xi = re^{i\phi}$. By applying $\hat{S}(\xi)$ to the vacuum $|0\rangle$ we can generate the squeezed state $|\xi\rangle = \hat{S}(\xi)|0\rangle$. The phase $\phi$ defines the direction of squeezing in the phase space and the squeeze amplitude $r$ defines the amount of squeezing.

The Wigner function of a squeezed vacuum state (depicted in Fig. 2.1d) is

$$W(q,p) = \frac{2}{\pi} \exp \left(-\left((q + iP)^2 - \frac{1}{2} (e^{2r} - e^{-2r}) e^{-i\phi} (q + iP)^2\right)\right) \times \exp \left(-\frac{1}{2} (e^{2r} - e^{-2r}) e^{i\phi} (q - iP)^2\right) .$$

(2.21)
2.1.2 Tomography methods

Reference-state method for signal reconstruction

Since the energies of photons in microwave regime are much lower ($\sim 10^5$ times) than in optical regime, detecting the microwave photons is still experimentally challenging. Therefore, we resort to other means of microwave quantum state tomography such as the use of linear amplifiers before an actual signal detection might happen. However, amplifying a signal adds some noise which makes it difficult to reconstruct the initial quantum state. The reference-state method makes use of a known state $\hat{v}$ (usually the vacuum state $|0\rangle$) to characterize the noise added by an amplifier $\hat{V}$ with a gain $G$ [6, 7]. Consequently, the weak quantum signal $\hat{a}$ can be reconstructed by calibrating out the amplifier noise as it is schematically shown in Fig. 2.2.

![Figure 2.2: Schematic setup for the reference state method. A reference state $\hat{v}$ (usually the vacuum state $|0\rangle$) is used in a pulsed measurement to characterize the amplification noise $\hat{V}$ and later calibrate it out in order to obtain the initial signal $\hat{a}$.](image)

2.1.3 Intensity correlation function

In order to describe photon statistics of quantum signals one can use the probability distribution $P(n)$, which shows the probability to find $n$ photons in a state. The expected value for the mean number of photons is $\langle n \rangle = \langle \hat{a}^\dagger \hat{a} \rangle$. For
coherent states we find that the probability distribution is the Poisson distribution [1] and for thermal states the Planck distribution [8] (depicted in Fig. 2.3 (a))

\[
P(n) = \begin{cases} 
\frac{\langle n \rangle^n}{n!} e^{-\langle n \rangle}, & \text{for coherent state,} \\
\frac{1}{(\langle n \rangle + 1)^n}, & \text{for thermal state.}
\end{cases} \tag{2.22}
\]

Since it can be hard to obtain \( P(n) \) experimentally, we can measure the second-order correlation function \( g^{(2)}(\tau) \) which can be expressed as [1]

\[
g^{(2)}(\tau) = \frac{\langle \hat{a}^\dagger(t) \hat{a}^\dagger(t+\tau) \hat{a}(t+\tau) \hat{a}(t) \rangle}{\langle \hat{a}^\dagger(t+\tau) \hat{a}(t+\tau) \rangle \langle \hat{a}^\dagger(t) \hat{a}(t) \rangle}, \tag{2.23}
\]

where the variance of the photon number \( \text{Var}(n) \) can be measured similar to the experiment of Hanbury Brown and Twiss [9, 10, 11, 12]. For a stationary field and \( \tau = 0 \) the correlation function becomes

\[
g^{(2)}(0) = \frac{\text{Var}(\hat{a}^\dagger \hat{a}) - \langle \hat{a}^\dagger \hat{a} \rangle}{\langle \hat{a}^\dagger \hat{a} \rangle^2} + 1. \tag{2.24}
\]

For the coherent field, the variance of the photon number is given by the squared field amplitude \( \text{Var}(\hat{a}^\dagger \hat{a}) = |\alpha|^2 \), which is also the expectation value of the photon number \( \langle \hat{a}^\dagger \hat{a} \rangle \). Therefore, Eq. 2.24 becomes 1 for a coherent field. For the thermal field, we find \( \text{Var}(\hat{n}_{\text{th}}) = n_{\text{th}}^2 + n_{\text{th}} \) and \( \langle \hat{a}^\dagger \hat{a} \rangle = n_{\text{th}} \) from the Eq. 2.15. In this case, Eq. \( g^{(2)}(0) = 2 \) independently of the mean photon number.

For displaced thermal fields the second order correlation function \( g^{(2)}(0) \) can vary between 2 and 1 depending on the parameters of the state. The variance and the expectation value of the photon number are given by the combined thermal and coherent field, which we assume to be uncorrelated

\[
\text{Var}(\hat{a}^\dagger \hat{a}) = (n_{\text{th}}^2 + n_{\text{th}}) + |\alpha|^2, \tag{2.25}
\]

\[
\langle \hat{a}^\dagger \hat{a} \rangle = n_{\text{th}} + |\alpha|^2. \tag{2.26}
\]

Eq. 2.24 becomes

\[
g^{(2)}(0) = \frac{n_{\text{th}}^2}{(n_{\text{th}} + |\alpha|^2)^2} + 1. \tag{2.27}
\]

Fig. 2.3 (b) shows the smooth transition from the thermal state with \( g^{(2)}(0) = 2 \) to \( g^{(2)}(0) = 1 \) for different thermal photon number contributions.
Figure 2.3: (a) Probability to find \( n \) photons for thermal and coherent states with \( \langle n \rangle = 7 \) photons. (b) Second-order correlation function for thermal and coherent states as a function of \( n = |\alpha|^2 \). Dashed and dotted lines illustrate the case for displaced thermal states with \( n_{th} = 1 \) and \( n_{th} = 3 \), respectively.

2.2 Quantum communication with propagating squeezed microwaves

In this section we briefly introduce a somewhat related quantum communication protocol, the quantum teleportation, which can be implemented in the future by using these squeezed microwaves.

2.2.1 Quantum teleportation

A quantum teleportation protocol (QTP) for discrete variables by the means of quantum entanglement and classical communication was proposed in 1993 [13]. First experimental realizations of quantum teleportation with discrete variables were shown in 1997 [14, 15], by teleporting a polarization state of an optical photon. Theoretical work on QTP with continuous variables has been done by L. Vaidman [16] and S. L. Braunstein et al. [17]. Experimental implementation of QTP with continuous variables has been demonstrated in 2011 [18]. However, because of technical difficulties, such as the lack of single microwave photon detector, quantum teleportation with continuous variables in the microwave domain has not yet been realized. Here, we present a quantum teleportation protocol using continuous variables.

In this protocol party \( A \) (Alice) wants to teleport an unknown target quantum state \( \Phi_T \) to party \( B \) (Bob). \( A \) and \( B \) are spatially separated and share an Einstein-
Podolski-Rosen (EPR) [19] pair $\Phi_{AB}$ which is defined as follows

$$ (\hat{q}_A + \hat{q}_B) |\Phi_{AB}\rangle = \delta (q_A + q_B) , \ (\hat{p}_A - \hat{p}_B) |\Phi_{AB}\rangle = \delta (p_A - p_B) , \quad (2.28) $$

where $\hat{q}$ and $\hat{p}$ are the conventional quadrature operators. The unknown quantum state is characterized by $\hat{q}_T$ and $\hat{p}_T$.

Alice superimposes her signal with the target state $T$ via a beam splitter and performs a Bell-type measurement on the mixed system. Doing so the classical results $a$ and $b$ can be read

$$ q_T + q_A = a , \ p_t - p_A = b . \quad (2.29) $$

The measurement is a local measurement and its outcome does not contain any information on the state $T$, because the systems $A$ and $T$ are still entangled. After this measurement, on Bob’s side, the EPR pair collapses into

$$ q_B = q_T - a , \ p_B = p_T - b . \quad (2.30) $$

Afterwards, Alice reveals her measurement results $a$ and $b$ to Bob via a classical channel (feed-forward signal) and Bob displaces his state by a complex amplitude $\alpha = a + ib$. After this displacement, the state $B$ becomes identical to the state $T$.

Since the entangled systems are constrained in time by the decoherence time, the classical signal has to be generated quickly enough ($\approx \mu s$) to retain the quantum information. Since FPGAs can process data in the ns regime, a fast feed-forward signal can be achieved digitally by using an FPGA.

### 2.3 Digital signal processing

In digital signal processing (DSP) a discrete-time signal $x[n]$ is obtained by sampling a continuous-time signal $x(t)$, with an interval $T$ between two successive samples, known as the sampling period. Continuous variables such as a continuous-time signal are denoted in parentheses ($\cdot$), and discrete variables in square brackets $[\cdot]$. We define $f_s = 1/T$ as the sampling rate which provides the number of samples per second. A practical A/D converter (ADC) has as input an analog signal $A(t)$ and analog reference $R(t)$ and after a certain amount of time (conversion time) provides as output a digital signal $D[n]$ such that

$$ A(t) \approx R(t) D[n] = R \left( b_1 2^{-1} + b_2 2^{-2} + ... + b_B 2^{-B} \right) , \quad (2.31) $$
with $b_i$ being the i-bit number and $B$ defining the resolution $\Delta = 2^{-B}$ of an ADC [20]. The value $D$ is a digital approximation of the ratio $A/R$ within the resolution. This process is repeated at every sampling period. This process of conversion is depicted in Fig. 2.4.

**Figure 2.4:** Block diagram representation of an analog-to-digital (ADC) converter. A signal is first sampled with the sampling frequency $f_s = 1/T$ creating a time-discrete signal $x[n] = x(nT)$. Then, the quantizer with a specified bit resolution $B$ gives us the quantized digital signal $x_d[n]$. Last, a coder rewrites the magnitude of each digitized value into a binary number for further computational processing.

The sampling of a sinusoidal signal is shown in Fig. 2.5. The sampler converts the continuous-time signal to a discrete-time signal by measuring the signal values at regular intervals of time. The quantizer converts the continuous amplitude $x$ into a discrete amplitude $x_d$. The final digital signal has a different shape than the discrete-time signal due to a quantization error $e[n]$. The quantization error decreases with increasing number of bits $B$, since the difference between discrete-time and digital signal vanishes (an ideal analog-to-digital converter $B \to \infty$).

The quantization error is given by

$$e[n] = x_d[n] - x[n], \quad (2.32)$$

where $x_d[n]$ is the digitized signal. The digitizing process is described by its nonlinear in-/output characteristic $x_d[n] = Q(x[n])$, with $Q(\cdot)$ being the quantizer function. For linear uniform quantization it is common to differentiate between two quantizer functions:

$$Q(x[n]) = \Delta \left\lfloor \frac{x[n]}{\Delta} + \frac{1}{2} \right\rfloor, \quad (2.33)$$

for the rounding process, and

$$Q(x[n]) = \Delta \left( \left\lfloor \frac{x[n]}{\Delta} \right\rfloor + 1 \right), \quad (2.34)$$
2.3 Digital signal processing

Figure 2.5: Example of a sinusoidal signal in continuous-, discrete-time, and digital form. Amplitudes of $x(t)$ and $x[n]$ are known with infinite precision, whereas $x_d[n]$ is only known with a finite precision $\Delta$. $e[n]$ denotes the noise produced by the quantization process.

for the truncating process. Here, the $\lfloor \cdot \rfloor$ denotes the floor function, which maps a real number to the largest integer not greater than its argument. The smallest difference between values is given by the least significant bit. Therefore the quantization step is given by

$$\Delta_{\text{step}} = X_m 2^{-B}, \quad (2.35)$$

with $X_m$ being the largest achievable integer number. The quantization of a number to $(B + 1)$ bits can be done by using either rounding or truncation. Here the ranges of the quantization error is $-\Delta_{\text{step}}/2 < e < \Delta_{\text{step}}/2$ for rounding and $-\Delta_{\text{step}} \leq e \leq 0$ for truncation, as long as $|x| < X_m$. If a value $x$ exceeds $X_m$, the overflow condition arises. While the error of a rounded number tends to be minimized, the magnitude of it could be greater than the original value which sometimes is not desirable. Truncating an integer always assures that the number is less or equal to the original value.

Analog signal processing

Analog signal processing (ASP) deals with conversion of analog signals into electrical signals by using transducers or sensors and their processing by analog
electrical circuits. Usually some form of conditioning such as amplification is needed before the output of the sensor can be processed by the analog signal processor. When comparing ASP with DSP, there are many advantages of the digital systems over the analog systems such as [20]:

1. Cost-effective way to solve sophisticated signal processing problems using digital techniques.

2. There exist important signal processing techniques which are impossible to implement using analog electronics.

3. Digital systems are more reliable, compact, and less sensitive to environmental noise and component ageing than analog systems.

**Signal sampling**

The sequence of samples $x[n]$ is obtained from a continuous-time signal $x_c(t)$ commonly by taking values at equally spaced points in time. The periodic sampling is defined by

$$x[n] \equiv x_c(t)_{t=nT} = x_c(nT),$$

(2.36)

with $T$ being the fixed time interval between the samples. By analysing the discrete-time Fourier transform (DTFT) $X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\omega n}$ of $x[n]$ and continuous-time Fourier transform (CTFT) $X_c(\Omega) = \int_{-\infty}^{\infty} x_c(t)e^{i\Omega t}dt$ of $x_c(t)$, we can find an equivalent relationship for the periodic sampling in Eq. (2.36) in the frequency domain as

$$X(\omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c \left( \frac{\omega}{T} - i\frac{2\pi}{T}k \right).$$

(2.37)

The spectrum of $x[n]$ can be recreated if $x_c(t)$ is limited in its bandwidth, that is when $X_c(\Omega) = 0$ for $|\Omega| > \Omega_H$, with $\Omega_H$ being the highest frequency in the spectrum. Fig. 2.6 shows a bandlimited $X_c(\Omega)$ and the corresponding $X(\omega)$ for two cases, $\Omega_s > 2\Omega_H$ and $\Omega_s < 2\Omega_H$. We observe in Fig. 2.6(b) that the sequence $x_c(t)$ can be reconstructed from $x[n]$, since the input spectrum $X_c$ remains the same when $\Omega_s > 2\Omega_H$ (Nyquist-Shannon sampling theorem [21]). In contrast as depicted in Fig.2.6(c), if $\Omega_s < 2\Omega_H$, copies of $X_c$ overlap and the spectrum cannot be recovered any more. This effect of overlapping is called *aliasing*. Two conditions are necessary to prevent overlapping of the spectral bands: the continuous-time
signal must be limited in its bandwidth and the sampling frequency must be sufficiently large [20]:

\[ X_c(\Omega) = 0, \text{ for } |\Omega| > \Omega_H , \]  

(2.38)

and

\[ \Omega_s \geq 2\Omega_H . \]  

(2.39)

An example of aliasing effect in time-domain is given in Fig. 2.7, where two sinusoidal signals with frequencies 2 Hz and 18 Hz are sampled at two different frequencies 20 Hz and 1000 Hz.
2.3.1 Finite impulse response filter

A finite impulse response (FIR) filter is a linear shift invariant filter with an impulse response sequence \( h[n] \). The output function \( y[n] \) to an arbitrary input function \( x[n] \) is given by the convolution between the impulse response and the input function \[ y[n] = h[n] * x[n] = \sum_{k=0}^{N} h[k] x[n-k] \tag{2.40} \]

In Fig.2.8 a block diagram of a FIR filter which results from the Eq. (2.40) is shown.

The discrete-time FIR filter of the order \( N \) has the impulse response sequence

\[ h[n] = \sum_{k=0}^{N} h[k] \cdot \delta[n-k] \tag{2.41} \]

By z-transforming the impulse response sequence we obtain the transfer function
of a $N^{th}$ order FIR filter:

$$H(z) = \sum_{k=0}^{N} h[k] z^{-k} = \frac{h[0] z^N + h[1] z^{N-1} + \ldots + h[N]}{z^N}.$$  \hspace{1cm} (2.42)

The frequency response function of the filter can be derived from the transfer function for $z = e^{i\omega}$

$$H(e^{i\omega}) = \sum_{k=0}^{N} h[k] e^{-ik\omega}.$$  \hspace{1cm} (2.43)

Equation (2.43) is the discrete time Fourier transformation (DFTF) of $h[k]$. We see that the frequency response function of the FIR filter is essentially the Fourier transform of the impulse response function. Since the ideal filter response function, which is theoretically of infinite length, has an output before applying a signal, it is not realisable in digital signal processing applications. Therefore, in order to realize a FIR filter, we have to cut the impulse response function at $-N \leq n \leq N$ and shift it $N$ times to the right. The operation of cutting the function we can replace by multiplying the impulse response of the ideal filter with a rectangular weighting window sequence $w[n]$ which is defined as

$$w[n] = \begin{cases} 1, & 0 \leq |n| \leq N, \\ 0, & \text{otherwise}. \end{cases}$$  \hspace{1cm} (2.44)

Various authors proposed window shapes other than the rectangular [22]. They
are characterized by different values of a main lobe width and side lobes amplitude. The most important and frequently used are:

- The **Bartlett window**:

\[
w[n] = \begin{cases} 
\frac{2n}{N}, & 0 \leq n \leq \frac{N}{2}, \\
2 - \frac{2n}{N}, & \frac{N}{2} \leq n \leq N.
\end{cases}
\]  
(2.45)

- The **Hanning window**:

\[
w[n] = \begin{cases} 
0.5 - 0.5 \cos \left(\frac{2\pi n}{N}\right), & 0 \leq n \leq N, \\
0, & \text{otherwise}.
\end{cases}
\]  
(2.46)

- The **Hamming window**:

\[
w[n] = \begin{cases} 
0.54 - 0.46 \cos \left(\frac{2\pi n}{N}\right), & 0 \leq n \leq N, \\
0, & \text{otherwise}.
\end{cases}
\]  
(2.47)

- The **Blackman window**:

\[
w[n] = \begin{cases} 
0.42 - 0.5 \cos \left(\frac{2\pi n}{N}\right) + 0.08 \cos \left(\frac{4\pi n}{N}\right), & 0 \leq n \leq N, \\
0, & \text{otherwise}.
\end{cases}
\]  
(2.48)

The stopband attenuation is independent of the window length but can be changed only by changing the shape of the window. The Hamming window provides the best compromise between a narrow transition width and high stopband attenuation.

### 2.3.2 Fixed-point number representation

Real numbers \( n \in \mathbb{R} \) in the computational world are described by integer numbers with some fractional part. Our modern computational field bear fundamental hardware support (floating-point unit or FPU) for floating-point numbers to reproduce good approximation of real numbers. The most common representation of float-point arithmetic is defined by the Institute of Electrical and Electronics Engineers standards (IEEE 754 [23]). In Tab. 2.1 we extract some of the most important representations of float-point numbers.
Figure 2.9: Comparison between the rectangular, Bartlett, Hanning, Hamming, and Blackman windows with the length \( n = 20 \), in both the time and frequency domains.

Table 2.1: Most important representations of float-point numbers.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Largest number</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>32bit</td>
<td>( \approx 3.4 \cdot 10^{38} )</td>
</tr>
<tr>
<td>double</td>
<td>64bit</td>
<td>( \approx 1.7 \cdot 10^{308} )</td>
</tr>
<tr>
<td>single extended, minimum</td>
<td>43bit</td>
<td>( \approx 2 \cdot 10^{308} )</td>
</tr>
<tr>
<td>double extended, minimum</td>
<td>79bit</td>
<td>( \approx 1 \cdot 10^{4932} )</td>
</tr>
</tbody>
</table>

However floating-point is not the only way to express fractional numbers. Fixed point data type is widely used in digital signal processing applications where performance (speed) is more important than precision. As we will see later, using integer arithmetic circuits we can recreate fractional numbers with some loss of precision when comparing to the floating-point representation.

**Binary Point**

Just like in a decimal system where the decimal point denotes the position of the fractional part of a numeral, the binary point acts the same, only with a different base of \( 2^n \) instead of \( 10^n \). All digits to the left of the binary point carry the weight of \( n = 0, 1, 2, \ldots \), and on the right of it, the weight of \( n = -1, -2, -3, \ldots \). Let us have a look at an example. The number 12.5 can be represented as following in the
Chapter 2 Theoretical foundations

binary form:

\[ 1100.1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} = 12.5. \]

We can now realize that the bit pattern of 25 and 12.5 are the same, with the only difference being the position of the binary point: 1100.1 = 12.5, but 11001. = 25.0. We could also shift the binary point of the same bit pattern to the left obtaining the number 6.25 = 110.01. Therefore, by shifting the binary point by a given number \( n \) to the left (right), we achieve a division (multiplication) of the initial value by \( 2^n \). This shifting process is the key to understand fixed point number representation. We note that there is a trade-off between range and resolution, which is controlled by the location of the binary point.

Here we can define a fixed point numeral by simply specifying the bit width of the number and the binary point position within this number. Within the rest of the thesis we will use the following notation for the fixed-point representation:

\[ \text{fixed-point } \langle \pm, w, b \rangle , \]

where the first input \( \pm \) (+) denotes if the number is signed (unsigned) which is necessary if we need to represent also negative numbers. The \( w \) serves as the bit width of the number and the \( b \) gives the position of the binary point. For example (fixed-point\( \langle +, 12,6 \rangle \)) represents a 12-bit unsigned (reaching from 0 to \( 2^{12-6} - 1 \)) fixed point number, of which, the 6 bits after the binary point are fractional. These bits define the smallest distance between two numbers, in this case \( 2^{-6} \). Therefore, the bit pattern 000101001101 represents the following number: 000101.001101 = 1.2^2 + 1.2^0 + 1.2^{-3} + 1.2^{-4} + 1.2^{-6} = 5.203125. If we treat this bit pattern as integer, it represents the number 000101001101 = 333.

To represent negative numbers we need to denote the type as signed number, e.g. fixed-point\( \langle \pm, w, b \rangle \). The weight of each bit is a power of two, except for the most significant bit, whose weight is the negative of the corresponding power of two. Here, the value \( x \) of an \( N \)-bit number \( a_{N-1}a_{N-2}...a_0 \) is given by the formula

\[ x = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i2^i. \tag{2.49} \]

For example, the bit pattern 1001\(_2\) with the type fixed-point\( \langle \pm, 4,1 \rangle \) represents the number -3.5, whereas 0001\(_2\) of the same type represents 0.5. This method of representing signed numbers, and more generally, fixed point binary values,
is called two’s complement [24]. Using float point arithmetic in digital signal processing leads to higher power usage when compared to lower precision fixed-point arithmetic [25]. This is also true for field programmable gate arrays (FPGAs), where floating-point implementations require larger amounts of FPGA resources than an equivalent fixed-point solution. Therefore, modern FPGAs designs make use of the fixed-point data type optimizing power and resource usage.

2.3 Field-programmable gate array (FPGA)

Field programmable gate arrays (FPGAs), are integrated circuit devices which can be programmed using a hardware description language (HDL). Although the most common HDL languages used for FPGA programming are Verilog and VHDL, for this thesis the whole project is implemented in LabVIEW FPGA for a number of reasons. Foremost, since most of our the measuring instruments are already designed in LabVIEW, it is easier to make the FPGA code more compatible by also using LabVIEW for it. Other than that, for this work, we use the PXIe-7972R FPGA model with a Xilinx Kintex-7 K325T device chip on it which is provided by National Instruments (NI). Also because LabVIEW and LabVIEW FPGA are supported by NI, it is also reasonable to program the FPGA image on the same language.

FPGA design

Generally, FPGAs are built up from different types of modules. These are Input/Output blocks (I/O blocks), interconnection wires, and configurable logic blocks (CLB). The basic FPGA architecture consists of a two-dimensional array of logic blocks with programmable interconnection between them (Fig. 2.10). The configurable logic blocks are made of multiplexers (MUX), D flip-flops (DFF), and look-up tables (LUT) (Fig. 2.11). The look-up table is used to implement combinational logical functions. The number of inputs to the LUT can vary between 3, 4, 6, and 8. Fig.2.11 shows a 6 input look-up table, which is also used for the Kintex-7 FPGA family, and one output which is a combination of the 6 inputs. D flip-flops, also known as delay flip flops, capture the value from the D-input at a definite portion of clock cycle, such as the rising edge of the clock [26]. This value becomes the Q output, which is given out at the next clock cycle. Therefore, the DFF can be viewed as a shift register. A truth table for the D flip-flop is shown in Tab.2.2. A multiplexer is a device which selects between several input signals and forwards it to a single output line. A 2-to-1 (2 inputs,
Figure 2.10: Scheme of a modern FPGA hardware architecture. I/O blocks allow interfacing between internal configurable logic blocks and pins a processor. The logic blocks provide combinatorial and synchronous logic, sometimes also used as distributed memory. The interconnection between the logic blocks are usually also programmable by using switch blocks.

one output) MUX implements the boolean equation

$$Z = (A \land \neg S_0) \lor (B \land S_0), \quad (2.50)$$

where $A$ and $B$ are the two inputs, $S_0$ is the selector input, and $Z$ is the output. We can also express this equation as a truth table (Tab. 2.3).

Table 2.3: Truth table for the 2-to-1 multiplexer (MUX). When the selector $S_0 = 0$, the output $Z$ becomes $A$, else $B$ [27].

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$A$</td>
</tr>
<tr>
<td>1</td>
<td>$B$</td>
</tr>
</tbody>
</table>

In larger multiplexers, the number of selector pins is equal to $\log_2(n)$, with $n$ number of inputs. A realization of the 2-to-1 multiplexer would need 2 AND gates, an OR gate and a NOT gate [27]. The computing functionality of an
2.3 Digital signal processing

**Figure 2.11:** Schematic picture of a configurable logic block (CLB). To implement particular functions using CLB, the output columns of a truth table are loaded into memory as a LUT. The input columns of a truth table are used as select inputs to multiplexers. Further, flip-flops are used for delaying signal outcomes if needed.

**Table 2.2:** Truth table for the D flip-flop. Input D is delayed by one cycle after the rising edge of the clock. The X denotes an irrelevant condition, since Q will only change if we have an active transition of the clock.

<table>
<thead>
<tr>
<th>Clock</th>
<th>D</th>
<th>Q_{next}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rising edge</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rising edge</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Non-Rising</td>
<td>X</td>
<td>Q</td>
</tr>
</tbody>
</table>

FPGA is provided by the configurable logic blocks which are connected to each other through a programmable routing network, the interconnection. The latter consists of wires and programmable switches that form the required connection. The configurable logic blocks can be programmed in such a way that logical operations, such as a NAND gate, are a combination of inputs of the look-up tables. Therefore, by changing the combinations of the look-up tables inputs we also define our signal processing chain.
Chapter 3

Experimental setup

In this chapter, we discuss the experimental setup which is developed to implement the tomography of propagating microwaves. First, we present an actual experimental setup with two different acquisition devices. Next, we discuss generation of microwave coherent signals and noise for testing of the tomography routines. Finally, we establish a modulation trigger scheme required for efficient and stable measurements.

3.1 Analog-to-digital-converter setup

Here, we would like to develop a faster microwave state reconstruction setup, as compared to older implementation based on conventional ADC cards and PC-based data processing. In this regard, using an FPGA-based data acquisition and processing allows for optimal parallelization of many tasks yielding a significant speed-up. In convetional set-ups, measurement data stored in the memory of an ADC card needs to be sent to the PC for further processing which significantly slows down the whole measurement routine. Therefore, processing the data on an FPGA directly has a big advantage that only a processed and averaged data needs to be sent to the PC, thus, saving a lot of time. From a crude estimation, the expected measurement speed-up factor varies between 5 and 50 depending on a particular measurement scenario.

3.1.1 Field programmable gate array (FPGA)

Within the scope of this thesis, we use two NI FlexRIO cards with modular I/O. The FlexRIO architecture consists of a FlexRIO transceiver adapter module, for the analog I/O and an FPGA module which can be programmed with LabVIEW or Verilog/VHDL. A PXIe-7972R model is used for the most of the thesis. Later, the code is tested with a more powerful model PXIe-7975R. The PXIe-7972R
3.1 Analog-to-digital-converter setup

features a Xilinx Kintex-7 K325T FPGA and 2 GB of onboard dynamical random access memory (DRAM). The most important specifications of the two FlexRIO modules are given in Tab. 3.1.

Table 3.1: Feature summary of the Kintex-7 FPGAs. Each of the configurable logic blocks (CLBs) contain four look-up tables (with 6 inputs and 1 output each) and eight D flip-flops, some of which can also be used as distributed random access memory (RAM). Each of the digital signal processing (DSP) slices contain a pre-adder, a 25 × 18 multiplier, an adder, and an accumulator. Furthermore, both FPGA modules include 2 GB of onboard DRAM.

<table>
<thead>
<tr>
<th>Model</th>
<th>FPGA</th>
<th>CLB slices</th>
<th>DSP slices</th>
<th>Block RAM (kbits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXIe-7972R</td>
<td>Kintex-7 K325T</td>
<td>50,950</td>
<td>840</td>
<td>16,020</td>
</tr>
<tr>
<td>PXIe-7975R</td>
<td>Kintex-7 K410T</td>
<td>63,550</td>
<td>1,540</td>
<td>28,620</td>
</tr>
</tbody>
</table>

The FPGA module is placed in a NI PXIe-1073 chassis which potentially allows for housing of multiple microwave PXIe-compatible devices or FPGAs (Fig. 3.1). The communication to the host computer is established via a PCI-express 3.0 cable, which has a maximum transfer rate of 8.0 GT/s.

Transceiver adapter module

To digitize the signal, we use the NI 5782 transceiver adapter module, which has the maximum sampling frequency of 250 MHz which corresponds to a 4 ns time resolution and a 14-bit analog input resolution. The normal operating input range of the ADC for an AC signal is fixed at ±2.05 V. The transceiver is connected on top of the FlexRIO card as shown in Fig. 3.2.

Figure 3.2: FlexRIO module with a Xilinx Kintex-7 FPGA and a transceiver adapter module with I/O SMA connectors on the front panel.
Chapter 3 Experimental setup

Figure 3.1: Photograph of the FPGA-based microwave state tomography setup.

On the front panel, there are two single-ended (SE) analog input channels (AI 0 and AI 1), two SE analog output channels (AO 0 and AO 1), a trigger input channel (TRIG), a SE external reference input (CLK IN) and an auxiliary input/output connector (AUX I/O).

The sample rate and other timing functions on the transceiver adapter module are controlled by the clocks inside of the device. In order to achieve correct timing, usually the NI 5782 internal clock needs to be referenced to an external clock source.

Microwave FPGA setup

Fig. 3.3 shows the microwave experimental setup used for the FPGA-based state tomography. Two arbitrary waveform generators (AWG’s)\(^1\) are used in our setup to generate coherent signals, simulate thermal noise, and send trigger

\(^1\)Keysight 81160A AWG’s
Figure 3.3: Block diagram of the FPGA-based microwave setup. The top AWG send trigger pulses to start the FPGA data acquisition and pulse-modulate the SGS output signals. The SGS sends a microwave signal whenever the trigger is on. The bottom AWG generates a noisy signal to simulate thermal states. All devices are referenced to the 10 MHz Rubidium reference source.

pulses. The latter are used to trigger the FPGA data acquisition and modulate the amplitude of the SGS\textsuperscript{2} coherent signal output. Each device is synchronized to the 10 MHz frequency reference\textsuperscript{3} to stabilize the phase which allows us to avoid unwanted phase-drifts and jitter between different pulses. Noise signals are generated by combining artificial Gaussian noise from an AWG and a signal either from another AWG or, for pulsed measurements, via combining a coherent tone from the SGS with a noise from the AWG using a power combiner\textsuperscript{4}. This power combiner has a transmission loss of $\sim 3 \text{ dB}$ in the frequency range we use for the microwaves.

\textsuperscript{2}Rohde&Schwarz SGS100A
\textsuperscript{3}Stanford Research FS725 Rb frequency standard
\textsuperscript{4}MCLI PS2-7
Chapter 3 Experimental setup

3.1.2 Trigger scheme

Figure 3.4: Schematic overview of the trigger scheme for the FPGA data acquisition. Plot (a) shows the data acquisition time window (greyed area) started by a short trigger pulse (blue line). Red line shows the SGS modulation pulse which is at 0 V for half of the window duration and 0.5 V for another half. Plot b shows the raw data registered by the FPGA.

In order to recover low-power signals from the noisy background, we need to average over a certain amount of measurements. As a result, we have to be able to store the outcomes of multiple similar measurements. For this purpose, the time modulation of signals and measurements need to be considered. The amplitude and phase of our generated signal has to be identical for each trace. Due to the memory limitation on the FPGA card (BRAM), we are limited by the trace length of up to 1000 demodulated points which corresponds to a time trace length of $88 \mu s^5$. This sets a limit on the maximal frequency for the trigger repetition rate $f_{\text{trigger, max}} = 1/(88 \cdot 10^{-6}) \text{Hz} = 11.36 \text{kHz}$. Since the data acquisition and transfer happen in parallel we do not need to take in consideration the times needed for the latter. In our experiments, the highest frequency used for triggering the FPGA and the Acqiris card was $f_{\text{trigger}} = 10 \text{kHz}$. The averaging duty cycle in our experiments is given by the formula

$$D = \frac{f_{\text{trigger}}}{f_{\text{trigger, max}}},$$

(3.1)

which in our case, for the highest frequency used, is $\sim 88\%$. The amplitude of the trigger pulse needs to be equal or above 1.6 V (maximum of 3.6 V) and

$^5 \sim 90\%$ of the BRAM is used by storing traces of 1000 points length, see chapter 4 for details.
its pulse width needs to be at least $1/f_s$ or longer, which for our transceiver adapter modules is 8 ns. The modulation pulse for the SGS switches the output preamplifier to ON/OFF, thus generating a pulsed signal. Next, for the reference state reconstruction method, we need at least two pulses. The first pulse serves to define a reference signal, which usually is the vacuum state in cryogenic experiments. In our case, we use a known noise signal generated by the AWG as the reference signal emulating the amplified vacuum signal. The second pulse is the microwave signal we want to reconstruct. Such a pulsed scheme is shown in Fig. 3.4(b). To achieve such a scheme, we need a modulation signal for the SGS which in our case is also generated by one of the AWGs. For this purpose, we generate a square signal as shown in Fig. 3.4(a).

### 3.1.3 Acqiris card

![Block diagram of the Acqiris card setup](image)

**Figure 3.5:** Block diagram of the Acqiris card setup. As for the FPGA setup, an AWG generates a trigger pulse to start the data acquisition. The SGS and the bottom AWG generate, combined, a noisy coherent signal output.

In previous experiments, we used the Acqiris DC440 digitizer board, with a sampling rate of 400 MS/s and a 12-bit resolution. The input voltage ranges are from ±125 mV to ±5 V. The data accumulation process in the Acqiris card is limited by a small internal memory ($\sim 10^7$ samples). Whenever the memory is full, the data is transferred to a computer for further processing (averaging, filtering, etc.). Unlike with the FPGA card, the demodulation, filtering and calculation of higher quadrature order is performed on the computer. The raw data is transferred to the PC via a PCI cable module installed on the Acqiris card. On the processing
computer a PXI-to-PCI card is installed which receives the data from the Acqiris card. The samples are stored in $N$ segments, each consisting of $M$ samples, and therefore, $N \times M \leq 10^7$. The moment calculation is repeated for $L$ cycles, and therefore, each trace is averaged $N \times L$ times. The setup for the Acqiris card is shown in Fig. 3.5. Two analog inputs, C1 and C2, are provided by the digitizer card, but only one is needed for our measurements which are described in chapter 5. Similar to the measurements with the FPGA, the AWG is used to create a pulse [28, 29].
Chapter 4

Architecture of the image

In this chapter, we focus on technical aspects of the FPGA image. First, we discuss processes on the card which are required in order to achieve our goal. Further, we present a brief overview on the communication between the FPGA card and the PC which is used for final data processing and data storage. Finally, we describe a set of technical problems related to LabVIEW FPGA programming together with possible solutions.

4.1 Building blocks

First of all, we establish necessary requirements on the FPGA card for the intended signal processing. As discussed (see chapter 2), we describe a single-mode propagating electromagnetic wave $A(t)$ by its amplitude $A$, frequency $\omega$, and phase $\phi$. Since the frequency of the microwave is known (the unknown state is down-converted with a local-oscillator to a intermediate frequency of 11 MHz), we need to obtain the amplitude and phase information in order to reconstruct the original signal. An equivalent description is possible to using the in-phase ($I$) and out-of-phase ($Q$) quadratures, for which we can use corresponding quantum-mechanical operators. Moreover, by calculating statistical moments of these field quadratures, we can reconstruct the Wigner function of the initial signal. For Gaussian states, the knowledge of moments up to the second order is needed, and moments up to the fourth order are needed to check the Gaussianity of the state. To calculate the $IQ$-quadratures, we have to first demodulate our signal (DM) by mixing the incoming microwave signal with a digitally generated signal of the same frequency $\omega$ and integrating over one period. Essentially, this process is equivalent to the Fourier transform of one respective frequency $\omega$. Doing so, the demodulation process yields two signals at $\omega_{dm} = 0$ and $\omega_{dm} = 2\omega$. To get rid of the higher frequency part, we will need to implement a finite impulse response.
(FIR) filter. Furthermore, we use the filter to define a precise bandwidth of our measurements.

Since our final goal is to detect microwave states with only very few photons, the expected signal-to-noise ratio (SNR) is going to be very small because of the thermal noise dominance over the actual signal. To eliminate most of this thermal noise heavy averaging with a stable phase. For this, we use the trigger scheme presented in chapter 3. Also, for later experiments such as quantum teleportation, we will need to perform state reconstruction of two two-mode signals in order to prove entanglement between them. Therefore, we need two distinct input channels. These input channels need to have the same offset from the origin (OS) and peak-to-peak amplitude (AS). A sketch depicted in Fig. 4.1 shows the architecture of the image developed for these tasks with the NI FPGA.

![Diagram of signal processing chain in the FPGA](image)

**Figure 4.1:** Sketch of the signal processing chain in the FPGA. The incoming microwave signals at the analog inputs 0 and 1 (AI0 and AI1) are digitized into two chains $C_1$ and $C_2$. The channel balancing consists of two processing blocks, an offset subtraction (OS) and amplitude scaling (AS). After balancing the digitized microwave signals are demodulated (DM) into $I$ and $Q$ quadratures. The quadratures are then fed into the FIR filter to obtain a well-defined bandwidth. The last steps consist of quadrature moment calculation and averaging.

### 4.1.1 Channel balancing

Since we have two analog inputs (AI0 and AI1), we have to balance these two input channels, meaning that the offset and the amplitude have to be the same for both. This is done by subtracting an offset $V_{\text{offset}}$ from the digitized values and further multiplying them with a factor $a_{1,2}$ to scale the amplitude (Fig. 4.2)

$$V[n]_{\text{balanced}1,2} = (V[n]_{1,2} - V_{\text{offset}1,2}) \cdot a_{1,2},$$  

where $[n]$ represents the discretized data points, $\{1,2\}$ is the channel number, and $V[n]$ the sampled signal.
4.1 Building blocks

Figure 4.2: Channel balancing of sinusoidal signals by subtracting the offset from each of the digitized points and subsequent scaling of the amplitudes by multiplying them with the respective factors $\alpha_1, \alpha_2$. Digitized points $V_{1,2}[n]$ are obtained from the A/D converter of the transducer adapter module.

Data precision

The raw digitized data $V_{1,2}[n]$ is given by LabVIEW as 16 bits signed integer values (INT16), meaning that it ranges from $-32768$ to $32767$ (including 0). To use it in the FPGA, we have to convert the integers into fixed-point data type (see section 2.3). The precision given by the ADC is fixed-point $\langle \pm 16, 16 \rangle$. Since subtracting not lead to surpassing the aforementioned range for the INT16 values, we choose the same type after calibrating for the offset. However a multiplication operation scales the data, and can, by using a large multiplication factor $\alpha$, generate an overflow. Therefore, we need change the precision to a higher value. The precision is set to fixed-point $\langle \pm 48, 20 \rangle$ which allows us to process a signal with a peak-to-peak voltage of a maximum 200 mV without overflowing.

4.1.2 IQ demodulation

As discussed, we use the $I$ and $Q$ quadratures to describe our microwave signal. In order to calculate $I$ ($Q$) we need to mix the digitized input signal with the digital local oscillator $\sin(\omega_dt)$ ($\cos(\omega_dt)$). Doing so, the microwave signal $A(t) = A \sin(\omega t + \phi)$ is converted into demodulated components with the
frequencies $\omega \pm \omega_d$. By using a FIR filter with a cut-off frequency lower than the $\omega + \omega_d$, this frequency term is excluded. We obtain the DC quadratures by summing over a period and normalizing the $I, Q$ values

\[
I = \frac{\omega}{2\pi} \sum_{i=0}^{N} \cos(\omega_d t_i) A(t_i) \cdot \Delta t = \frac{A}{2} \cos(\phi), \tag{4.2}
\]

\[
Q = \frac{\omega}{2\pi} \sum_{i=0}^{N} \sin(\omega_d t_i) A(t_i) \cdot \Delta t = \frac{A}{2} \sin(\phi), \tag{4.3}
\]

where $\Delta t = 2\pi/\omega_s$ being the discrete time step which is defined by the sampling frequency $\omega_s$ and $N$ being the number of points for the integration. The latter also depends on the sampling rate $\omega_s$ of the adapter module and the frequency of our signal $\omega$

\[
N = \left\lfloor \frac{\omega_s}{\omega} \right\rfloor. \tag{4.4}
\]

**Digital signal generation**

As it was shown above, for the digital demodulation we require digital cosine and sine signals to be generated within the FPGA. There are two approaches how to create these signals within the card (i) by predefining values for the sin and cos in a memory, or (ii) by calculating them at each FPGA cycle. One FPGA cycle corresponds to a time $t = 1/f_s$ which is 8 ns for a sampling frequency $f_s = 125 \text{ MHz}$. The first approach is illustrated with Fig. 4.3. However, there are certain problems with this approach of generating digital signals. Firstly, the on-board memory (BRAM) is crucial for further signal processing with the FPGA card. So, this approach consumes valuable memory resources which are needed for storing the averaged traces. Second, and more important problem with this method, lies within the non-divisibility of the signal and sampling frequencies. Since we want to down-convert the incoming microwave signal at the frequency $\omega$ to a DC frequency, it is required that $\omega_d = \omega = 11 \text{ MHz} = 1/\Delta_2$. Here, the predefined values of the generated digital signal are called each FPGA cycle with a frequency of $125 \text{ MHz} = 1/\Delta_1$. The phase of the generated signal then shifts after one signal period, since $\Delta_2/\Delta_1$ is a real number with infinite decimals. The maximum phase shift which can occur is $\Delta_1/2$.

The second approach, i.e. to calculate the values for a digital signal each cycle, is depicted in Fig. 4.4. In this approach we have to determine the constant $2 \cdot \omega_d/\omega_s$ from the desired frequency $\omega$ of the generated signal. Since $2/\omega_s = 0.016$
is constant, we need to multiply this value with the signal frequency $\omega/2\pi$ - in our case with $11\text{ MHz}$, thus obtaining $0.176$ for the constant. This value is multiplied with the running number $i$, which starts at 0 and has a +1 increment after each FPGA cycle until the external trigger signal resets it back to 0. The calculated number is then fed into the digital local oscillator generator which is a predefined function in LabVIEW. This digital local oscillator calculates new values for the $\sin$ and $\cos$ in each FPGA cycle with a latency of 34 sampling periods which is specified by the precision used for it. The latency occurs due to the parallel programming nature of FPGAs.

In this case, the previous phase-matching problem will not occur any more, since the values are calculated continuously up to the point, where a trigger signal resets the running number $i$ back to 0. There is one disadvantage with this scheme, which results from the implemented hardware on the FPGA, since this needs 47 latency FPGA cycles (which correspond to a time latency of $376\text{ ns}$) to calculate the first values for the digital signal, thus restricting a fast reconstruction, e.g. for the digital feed-forward signal in the quantum teleportation protocol.
Chapter 4 Architecture of the image

Figure 4.4: Calculation-based digital signal generation. The constant value $2 \cdot \frac{\omega_d}{\omega_s}$ is multiplied with a running number $i$ which increases by 1 each FPGA cycle. The calculated value is fed into the \( \sin(\cos) \) generator which calculates each cycle a new value for the digital signal.

### IQ calculation

In order to get the actual DC values for $I$ and $Q$ we have to integrate over, at least, one whole period. In our case the integration turns into a summation (Eq. 4.3), since we work with discrete values of the digitized signal. We calculate summation period by using

$$\frac{f_{\text{sampling}}}{f_{\text{signal}}} = \frac{125 \text{ MHz}}{11 \text{ MHz}} = 11.36.$$  

Therefore, the summation period in our case is approximately 11 digitized points. This is realized by storing the number in a register and adding up for eleven cycles. Followed by that, the register data is sent to the filter as a valid number and set back to 0. The $IQ$ demodulation on the FPGA card is summarized in Fig. 4.5.

### 4.1.3 Finite impulse response filter

The FIR filter is designed by using the MATLAB filter designer (Fig. 4.6). We use a windowed low-pass FIR filter with an order parameter 90. The window used in our case is the Hamming one which gives the best ratio of the main lobe width to the side lobe amplitude (see chapter 2). The sampling frequency is $\omega_s/2\pi = 11 \text{ MHz}$ since for the demodulation we used 11 points of integration and for the cutoff frequency we use $\omega_c/2\pi = 0.2 \text{ MHz}$. The coefficients from the
4.1 Building blocks

Figure 4.5: Scheme of the IQ demodulation on the FPGA card. A digitized microwave signal of the form $A \cdot \sin(\omega t_i + \varphi)$ is first multiplied with values of digitally generated $\sin(\omega_d t_i)$ ($\cos(\omega_d t_i)$). Then, to obtain the down-converted $I$ and $Q$ signals we sum over one period $T = 2\pi/\omega$.

A filter designer can be stored in a "coe"-file and imported in the LabVIEW FPGA FIR Compiler. To successfully import the coefficients, the data has to be saved in fixed-point arithmetic. Also, we have to customize the precision for the coefficients and adjust it to be the same in LabVIEW FPGA for the filter block, otherwise the FIR Compiler will fail computing the correct filter. The precision for the coefficients we use is fixed-point $\langle \pm, 20, 20 \rangle$.

Furthermore, two more FIR filters with the same order parameter, window, and sampling frequency $\omega_s/2\pi$ at different cutoff frequencies, 0.1 MHz and 0.4 MHz, are implemented. To realize this, we implemented a LabView case from which we can choose the filter with the desired cutoff frequency.

Fig. 4.7 shows the experimentally measured power spectrum for the quadratures after filtering with the implemented FIR filter on the FPGA card. The setup used for this measurement is the one described in chapter 3. The power spectrum is calculated on the host PC with LabVIEW by using a predefined function from a 1000 points length trace of the demodulated $I_1$ quadrature. We observe a sharp peak at a frequency $\omega/2\pi = 0$ Hz which corresponds to the down-converted and demodulated quadrature of the coherent input signal with a frequency of 11 MHz. We additionally notice the 6 dB loss at the cutoff frequency $f_c = 200$ kHz, which agrees with the design values. In the FPGA image, we use a so-called digital
Figure 4.6: Filter designer in MATLAB with settings used for our purpose. The frequency response is shown in the graph within the design interface.

divider to avoid overflow while multiplying and summing within the FIR filter. This digital divider is actually a multiplication operation with the inverse of the denominator. The precision after the FIR block is fixed-point $\langle \pm 45, 25 \rangle$ which is also the precision of the first moments going to the averaging block. A comparison of $I_1(t)$ time traces, with and without the FIR, is shown in Fig. 4.8. The same result we observe for all other quadratures ($I_2, Q_1, Q_2$).

4.1.4 Moment calculation

In this section, we discuss how we obtain higher-order quadrature moments $I_j^1 I_k^2 Q_l^1 Q_m^2$. This is realized by multiplying each raw quadrature coming from the FIR filter with each other up to the fourth order $0 < j + k + l + m \leq 4$. The number of IQ-combinations $L$ for moments up to order $n$ with $N$ variables results from the urn model with replacement [30], where the number of possibilities is given by the binomial coefficient

$$L = \binom{N + n - 1}{n}. \quad(4.5)$$
In our case \( N = 4 \) is the number of quadratures, therefore calculating moments up to the 4-th order gives us \( L = 69 \) possible combinations. In order to achieve true parallel moment calculation we use delay registers (feedback nodes) to compensate for the latency caused by the multiplication blocks which in our case is 14 FPGA cycles (determined by trial and error to assure the compilation of the code onto the specific FPGA card). From the first order moments we can calculate higher order moments inductively - first the second order quadrature moments by multiplying first order moments with each other, then the third and fourth order moments by multiplying the second with the first and second order moments with each other, respectively. Therefore, we need overall 28 feedback nodes for the first order moments, 14 for the second order, and none for the third and fourth order (Fig. 4.9). A feedback node is a latency block which stores one data point for one FPGA cycle. Therefore 28 feedback nodes correspond to a time delay of 224 ns. Another important aspect to discuss is the precision used for the moment multiplication. In Tab. 4.1 the fixed-point representation used for every moment order is shown.

**Figure 4.7:** Measured power spectrum of FIR filtered and down-converted \( I,Q \) signals. We observe the same shape as the filter designed in MATLAB, with the DC peak corresponding to our demodulated signal. We observe the 6 dB loss at the cutoff frequency \( f_c = 200 \text{kHz} \).
Figure 4.8: Time trace of the $I_1$ quadrature with and without the FIR filter. We observe the smoothing of the DC quadrature resulting from filtering out of the broadband noise.

with the given range and precision. We observe that for the second and third order moments a lower amount of bits is used as compared to the first order. This happens because resources (DSP slices) on the FPGA card are limited, and therefore, we have to sacrifice the precision in order to compile the code successfully. First order moments are already calculated from the FIR block and it is not necessary to change the bits number, although the values which are calculated never reach full range.

Table 4.1: Fixed-point representation used for each moment order with given range and precision.

<table>
<thead>
<tr>
<th>Moment order</th>
<th>Fixed-point $(\pm, w, b)$</th>
<th>Range $[\text{Minimum} : \text{Maximum}] : \Delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>$(\pm, 45, 25)$</td>
<td>$[-1.68E7 : 1.68E7] : 9.54E-7$</td>
</tr>
<tr>
<td>2nd</td>
<td>$(\pm, 35, 20)$</td>
<td>$[-5.24E5 : 5.24E5] : 3.05E-5$</td>
</tr>
<tr>
<td>3rd</td>
<td>$(\pm, 42, 28)$</td>
<td>$[-1.34E8 : 1.34E8] : 6.10E-5$</td>
</tr>
</tbody>
</table>
4.1 Building blocks

Figure 4.9: Schematic picture of the moment calculation block. Raw quadratures coming from the FIR filter are used to calculate higher-order moments inductively. By introducing a latency time of 224 ns (28 feedback nodes) for the first order quadratures and 112 ns (14 feedback nodes) for the second order, we achieve true parallel stream for the real-time moments calculation which are later fed into the averaging system.

4.1.5 Averaging technique

After the calculation of the 69 quadrature moments, which is done in parallel, we need a mechanism to add traces of length \( l \) and divide by the number of the traces \( N \) in order to average out noise as depicted in Fig. 4.10. Each averaged point of the trace can be expressed as

\[
\langle I_1^I I_2^Q Q_1^I Q_2^Q \rangle = \frac{1}{N} \sum_{i=1}^{N} I_1^I_i I_2^Q_i Q_1^I_i Q_2^Q_i ,
\]

(4.6)

On the FPGA card the averaging is done by defining memories of certain address lengths \( l \) for each moment. The maximum length \( l \) depends on the used fixed-point precision which also varies for different moment orders, thus depends on the FPGA resources. In our case, we set trace lengths to 1000 points for each moment which correspond to a time-traces of 88 \( \mu s \). Using this length for 69 moments we use \( \sim 90\% \) of the FPGAs BRAM. Further, we also use this trace length for the rest of the thesis. Just like the calculation of the moments is done in parallel, so is the averaging. The averaging system based on the block RAM is explained in a flowchart in Fig. 4.11. Each memory on the FPGA card is initialized by clearing the data on it and setting the address number \( n \) and averaging number \( m \) to 0 ("reset memory") and then the data from address \( n \) is called until a valid point
Chapter 4 Architecture of the image

Figure 4.10: Schematic picture of the averaging process. Here, traces of matching phases are added up minimizing the input noise.

from the next trace comes in. These points are added together and saved in the same address of the memory. This procedure is repeated until \( n \) reaches the last address number of the memory \( N \). If reached, the averaging system will wait until an external trigger signal arrives. Next, if the trigger signal is true, the address number is set back to 0 and the averaging number \( m \) increases by 1. This whole scheme is repeated until \( m \) reaches the value \( M \) which is entered by the user. When achieved, the averaging mechanism stops and the data is transferred to the PC for post processing. The precision of each moment order used for the memories is listed in Tab.4.2. Fig. 4.12 shows the first averaging measurements for

<table>
<thead>
<tr>
<th>Moment order</th>
<th>Fixed-point ( \langle \pm, w, b \rangle )</th>
<th>Range ( [\text{Minimum} : \text{Maximum}] : \Delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>( \langle \pm, 40,25 \rangle )</td>
<td>([-1.68E7 : 1.68E7] : 3.05E - 5 )</td>
</tr>
<tr>
<td>2nd</td>
<td>( \langle \pm, 55,35 \rangle )</td>
<td>([-1.72E10 : 1.72E10] : 9.54E - 7 )</td>
</tr>
<tr>
<td>3rd</td>
<td>( \langle \pm, 60,40 \rangle )</td>
<td>([-5.50E11 : 5.50E11] : 9.54E - 7 )</td>
</tr>
<tr>
<td>4th</td>
<td>( \langle \pm, 64,50 \rangle )</td>
<td>([-5.63E14 : 5.63E14] : 6.10E - 5 )</td>
</tr>
</tbody>
</table>
4.2 Communication to PC

In order to analyse the averaged data from the FPGA, we need a transmission interface from the FPGA card to the host computer. Beside that, we should be able to efficiently exchange status and set configuration parameters of the FPGA.
card from the PC. This section provides relevant characteristics of the different information transfer methods.

**Controls and indicators**

For each control and indicator on the FPGA image, the compiler creates a register map with a set of hardware registers. The FPGA host interface provides access to these registers as read/write control functions which supports scalar data, such as numeric and boolean controls, and structured data, such as arrays and clusters. These controls and indicators are the lowest latency choice for exchanging information with the FPGA, usually in microsecond range.
We know that the FPGA can update the registers much faster compared to the processing speed of the host, thus we require extra synchronization mechanism such as handshaking protocols to achieve lossless data transfer.

**Handshaking protocols**

Handshaking protocols are signals which inform an upstream or downstream blocks in processing chains about the validity of their outputs or inputs. This is necessary in a single cycle timed loop, since some nodes need more than one cycle to compute valid data, but they are required to return data each clock cycle. This protocol involves four boolean terminals (four-wire protocol): Input Valid, Output Valid, Ready for Output and Ready for Input.

The input valid function needs to know about the validity of the data coming in and indicates to the function if the data passed is valid before it is executed. The output valid specifies if the output signal carries valid data. If two nodes include the four-wire protocol, one can connect the output valid of the upstream to the input value of the downstream.

The ready for output signal reports if a downstream can accept new output. This can be interpreted as being the readiness to accept data for the downstream. The ready for input tells upstream nodes if the block can read data on the next cycle.

**FPGA host interface FIFO**

With the aim of transferring large amounts of data, such as averaged traces, from the FPGA to the host PC, one has to access the host memory which stores the data before doing a read operation. This is done by making use of direct memory access (DMA) engine which consists of a DMA channel, provided by the local bus, and two first-in-first-out (FIFO) buffers: one on the FPGA and one on the host PC (Fig. 4.13). The FPGA is much faster with providing data to the DMA engine than the host system (because of the higher number of parallel processing units), therefore the CPU usage goes up when reading data from the FIFOs. Thus, using better PC processors might improve the overall efficiency of the system.

The FPGA buffers are constrained by limited resources, therefore we have to take care how much of the precision we use when setting up such memories, since using higher precision of the FIFOs leads to more resource usage. In our case, the FIFO buffers are set up for each of the 69 moments with different fixed-point precision for each moment order. Moments of the same order are then stacked in
arrays, such that combinations of 1, 2, 4, 8 and 16 width length arrays are used to fill up the FIFOs. This is due to the fact that the FIFO memories can only send a finite numbers of elements per write, which is the length of the input array. The precision we use is the same as the one used for our averaging structure, since using more would not be of any advantage but just cost us more resources.

The FIFO memories have to be controlled and monitored in order to avoid an overflow. This is done in our case by using the streaming VI from LabVIEW FPGA (Fig. 4.14). By counting the valid elements stored in the FPGA buffer, this VI can assure a stable data transfer from the FPGA to the host. The stream control on the FPGA side counts the elements of the averaged array if valid and sends a true value to the „Input Valid“ of our FIFO memory. This then stores the data in the FPGA buffer which will eventually be sent to the host via local bus and stored there in the buffer until the number of samples matches the value

Figure 4.13: Schematic picture of the FIFO interface system. Valid data is acquired by the FIFO memory on the FPGA card and sent through the local bus (DMA channel) to the host computer, where the data is first stored in a buffer then received in package form by the host FIFO. Picture recreated from www.ni.com.

Figure 4.14: Streaming VI predefined in LabVIEW FPGA. The input „element count“ counts the valid data points coming from the averaging system and sends them to the FIFO memory for the transfer to the host computer.
we control by using the „stream.num.samples (n)“ node from the computer to retrieve the data from the buffer. Here, (n) is the number of stream, since we need multiple FIFOs for different moment order with different fixed-point precision.

4.3 FPGA limitations and solutions

Timing errors

One of the most critical constraints in LabVIEW FPGA programming with high speed ADC is the critical path in a so-called single cycle timed loop (SCTL). The critical path is the data path with the longest propagation delay within such a loop. If this propagation delay is longer than the clock period of our SCTL, the compilation fails.

The total delay of our propagating signal has two components: routing delay and logic delay. Routing delay is attributed to the amount of time it takes signals to cross between circuit components. The signals travel approximately with speed of light but the clock rates are also high, therefore we must take these delays into account. We investigate this effect by calculating the travelling distance $l$ of a signal during one clock cycle via $l \approx c \cdot \frac{2\pi}{\omega_s} \approx 2.4 \text{ m}$. This result shows that routing delays must be taken into account, since the routing network of an FPGA occupies $80 \text{ – } 90\%$ of the total area.

Logic delays specify the time it takes for signals to propagate through logic components, such as the time it takes signals to go from the inputs of a logic circuit to its output.

Whenever a timing error occurs (total delay $>$ clock period), LabVIEW FPGA provides timing violation analysis. In the case shown in Fig. 4.15, the total delay of the multiplication block for second quadrature order was almost the same as a clock period ($8 \text{ ns}$) and therefore a timing error occurred. To solve this problem, we use higher internal latency ($12$ cycles) for the multipliers, which increases the total latency of our design, but shortens the critical path. The optimizing process of the trade-off between using higher latency for blocks and at the same time minimizing latency for faster reconstruction is a try and error process. We compiled the FPGA image for different latencies, until no timing error occurred any more.

Sampling frequency

The NI 5782 transceiver adapter module includes component-level intellectual property (CLIP) to enable hardware description language (HDL) IP programming
Figure 4.15: Timing violation error due to a high total delay of the propagating digital signal. The window on the left provides the timing violation analysis. Here, the logic of the multiplication for the second order moment needed longer time than the inverse clock rate, thus, leading to the compilation failure.

in LabVIEW FPGA. This facilitates the I/O modules to the LabVIEW project, permitting the user to make use of connector signals (shown in chapter 3).

There are two CLIPS provided by the NI 5782: the multiple and single sample CLIP. For the first one, the analog input channels generate two samples per clock cycle at a clock rate which is half the sample rate (125 MS/s). The latter CLIP generates one sample per clock rate at the default cycle frequency of 250 MHz. Both CLIPS have different impact on the implemented code. For example by using the single sample CLIP, the single cycle timed loop (SCTL) works at a frequency of 250 MHz. This implies that each block up to a feedback node inside the SCTL needs to process the data at this rate. Higher clock rates provide greater time resolution but can make it harder to compile the design on the card due to timing errors (4 ns restriction for the single sample CLIP).

For our design, we use the multiple sample CLIP which by default acquires two data points \( N \) and \( N - 1 \) for both analog input channels. This makes the code more complex, since we need to process four data points at the same time instead of two. To handle this problem, one could just neglect one data point per channel and work with half the time resolution, thus losing some of the experimental data. However, we could obtain a better solution to this problem by rewriting Eq. 4.2 in two parts \( A(t) = A'(t) + A''(t) \):

\[
I = \sum_{i=0}^{T} A(t_i) \cdot \cos(\omega t_i) = \sum_{i=0}^{T} A'(t_i) \cdot \cos(\omega t_i) + \sum_{i=0}^{T} A''(t_i) \cdot \cos(\omega t_i) \quad (4.7)
\]

Therefore, by adding two digitized values \( N \) and \( N - 1 \) just before integrating over
one period solves this problem, while still retaining the original time resolution. For this, we have to generate another \sin and \cos to create digital signals for the demodulation. However, the running number \( i \) needs to start at \( 1/2 \) instead of 0, since the second points are acquired with a delay of 4\,\text{ns}. A performance

![Figure 4.16: Standard deviation of \( I_1 \) for increasing number of averages. The standard deviation was calculated from traces of 1000 points.](image)

analysis is done by looking at the standard deviation \( \sigma \) of a raw quadrature, in this case the in-phase quadrature of the first channel, \( I_1 \). Since \( \sigma \propto 1/\sqrt{n} \), where \( n \) is the number of averages, we expect an improvement in performance of \( 1/\sqrt{2} \), which can be extracted from the Fig. 4.16. The standard deviation is calculated from 1000 point traces of a demodulated coherent signal with input power around \(-43\,\text{dBm} \) (including the losses from the power combiner) and artificial Gaussian noise generated by the AWG. We fit the data with the following equation

\[
\sigma(n) = \frac{A}{\sqrt{n}} + c ,
\]

where \( A \) and \( c \) are fitting parameters. The values for \( A \) are \( A = 4.92 \) for the one-point acquisition and \( A = 3.70 \) for the two-point acquisition. Calculating the ratio of these values we obtain 1.33 which is close to \( \sqrt{2} \). The latter means that our modified demodulation and averaging procedure works successfully.
Dynamic random access memory (DRAM)

Since BRAM is limited (~16 kbit) on our FPGA card, we could also make use of the dynamic random access memory (DRAM) provided by the FPGA card. The 7972R card includes 2GB of onboard DRAM (Tab. 3.1). Even though this memory quantity would be more than enough to store traces of large lengths (> $10^5$ points), making use of DRAM leads to certain difficulties.

One of the main problems with DRAM is the clock management of the memory. The DRAM interface is driven by a 166.67 MHz clock which differs from the 250 MHz acquisition clock. This means that the storing process needs to be placed in another timing loop with a frequency given by the DRAM clock. The data needs to be transferred from the acquisition loop to the DRAM loop which requires internal memory for intra FPGA FIFO memories. Aside from that, time resolution might be lost when processing the data at lower frequencies. Moreover, we had technical difficulties with the DRAM implementation in the LabVIEW code, such as calling and retrieving the data from the DRAM properly, as well as writing on it at the right time, since we cannot call a value from an address and write on it at the same time. In the end, the DRAM usage remains to be an unsolved challenge for now due to the number of technical problems. However, in the future it would be interesting to develop a lossless transfer interface between BRAM and DRAM in order to increase the amount of data we can process (i.e. average) directly on the FPGA card.

Resources on the FPGA card

As discussed in chapter 2, there are different types of modules which are available on the FPGA, such as LUT blocks and DSP slices. After compiling the LabVIEW code onto the FPGA card, the compiler gives a report on how many resources of the FPGA device have been utilized for the given design (Fig. 4.17). Since the FPGA has a finite number of resources for each type, adding too much logic to the design might lead to the compilation failure. For example, this can be caused by using too many DSP-slice, which are needed in our case for the multiplication of the quadratures. A typical solution to this problem is to reduce the usage of the slices directly, or reduce the precision used for the multiplication, such that all moments can be calculated with the given resources of the FPGA card. Examples of FPGA hardwares with different resource usages are shown in Fig. 4.18. A small design allows for successful compilation of the code, but is inefficient in resource usage, whereas a larger design can lead to compilation failure.
4.3 FPGA limitations and solutions

Figure 4.17: FPGA resource usage after last compilation. Almost all of the BRAM was used (~ 90%) due to the distribution of memories for the 69 traces of the moments. One can improve this number to almost ~ 100% for example by increasing the precision used for the fixed-point numbers in some of the memories, thus, utilizing the FPGA card more efficiently. The same could be optimized for the other modules on the card.

Compiling

Programming in LabVIEW FPGA requires the application to be synthesized on the FPGA card. The process of compilation can be time consuming which might be limiting factor during image programming and debugging. Here, we present the most important aspects of the compilation process.

While the code is still in development, first runs can be simulated on the computer by using the simulation environment in LabVIEW FPGA and resolve any programming errors before going to hardware. However, this environment only generates random signals, such as random integers from the I/O modules which is not useful for practical results. Therefore, we need to compile the code in order to test the application with the hardware which interfaces with real signals.

The full compilation process is depicted in Fig. 4.19. The start of a compilation initiates the generation of intermediate files, which converts the LabVIEW block diagram code into a form that the Xilinx compiler can make use of. During this process, errors such as using unauthorized code can occur, thus, causing a failure in the compilation process. The Xilinx compiler actually transforms the intermediate files into digital logic elements (analysis and synthesis) and divides the logic between the physical building blocks on the FPGA card (map). Afterwards, the compiler attributes the logic to to physical building blocks and routes the connections between them, while still maintaining the time restrictions of the compilation (place and route). Last, a binary data is created from the
compile worker which LabVIEW saves inside a bitfile. This file is used to run the designed application on the FPGA VI.

In order to reduce compilation times, which can go up to several hours (∼ 2 h for our case), there are some hardware factors which should be taken into account. The processor clock rate plays the most important role, since the compile job can only be executed on a single core (the process of compilation cannot proceed in parallel, since it is an optimization problem which is generally serial process), therefore, the clock rate is more important than the number of cores. Furthermore, the amount of RAM and the operating system will have an impact on the compilation as well. A 64-bit operating system can address more than 4 GB of RAM, thus reducing compile times even more. Finally, a compile worker based on a Linux operating system will further improve the performance of the compilation, since the Xilinx compile tools were originally build and optimized for the Linux operating system.

![Figure 4.18:](image)

**Figure 4.18:** (Left) Hardware layout of a small FPGA design. (Right) Hardware layout of a heavily packed FPGA. Red area marks the used FPGA physical blocks, the black area the unused.

![Figure 4.19:](image)

**Figure 4.19:** Schematic overview of the compilation process. Picture recreated from [www.ni.com](http://www.ni.com).
Chapter 5

FPGA benchmarking

In this chapter, we test our FPGA image by performing multiple measurements of a well-known microwave signal. We analyse the influence of signal averaging and check whether the filtering and moment calculation work as intended. We realize this by using the moments retrieved from the FPGA image in the reference state reconstruction algorithm for a variety of known microwave states. As the result, we obtain the Wigner function as well as the $g^{(2)}(0)$ correlation function and use them as benchmarks. Furthermore, we compare the performance of the FPGA-based measurement setup and the Acqiris-based setup in respect of the measurement time and show that we obtain a significant measurement speed-up with the FPGA card. In addition, we use the $g^{(2)}(0)$ correlation function to evaluate the influence of parameters which disturb our result, such as thermal noise or finite resolution of the digitized data.

5.1 Self-reference benchmarking

In this section, the performance of the FPGA is characterized by using the setup depicted in Fig. 3.1.

5.1.1 Pulsed measurements

For the reference state reconstruction method described in chapter 2, pulsed measurements are needed to reconstruct the original signal. For this, we employ a trigger scheme described in chapter 3, where a short trigger pulse generated by the AWG is used to initialize the FPGA data acquisition and another square-shaped pulse is used to modulate the SGS coherent output. After the implementation of the second order moments in the FPGA code, we attempt to check the pulsed scheme for a coherent signal only. Fig. 5.1(a) shows the first order moments and 5.1(b) the second order moments of a signal with
an amplitude of 7.25 mV. We can see that \( I, Q \neq 0 \) and \( I, Q = \text{const} \) in the first pulse of the signal, therefore we know the phase and the amplitude are constant, thus, obtaining a coherent signal. In the second pulse \( I, Q = 0 \). This results from \( \langle I \rangle = \langle Q \rangle = 0 \) for a thermal state which in this case is the environmental temperature.

![Figure 5.1: Pulsed measurement of a coherent signal with an amplitude of 7.25 mV. Top graph shows the first order moments and bottom graph the second order moments. The first pulse corresponds to the coherent signal, with the moments \( I, Q \neq 0 \), and the second pulse corresponds to the thermal noise.](image)

### 5.1.2 Quadrature statistics

Demodulating a noisy signal leads to noisy \( IQ \)-quadratures, hence the noise impacts the calculated moments. Here, we look at the standard deviation of
5.1 Self-reference benchmarking

the fluctuating $I$ and $Q$ traces and their dependence on the number of averages $N$. We measure traces with varying signal-to-noise ratios (SNR) to check the scaling of the standard deviation $\sigma \propto 1/\sqrt{N}$. Fig. 5.2 shows the time traces of the $I, Q$ quadratures of a coherent signal with a power of $-63$ dBm with added noise from the AWG of a peak-to-peak amplitude of 500 mV for two different averaging numbers ($10^4$ and $10^6$). In Fig. 5.2 we can observe how the time traces of the demodulated signal are flattened with increasing number of averages $N$. The traces correspond to two-pulse measurements which consist of two parts: the first part contains only a Gaussian noise from the AWG, the second one contains a combined noisy coherent signal. The knowledge of the first part is important for the reference state reconstruction, since it characterizes the amplification noise of an experimental setup. Furthermore, we observe a finite ring-up at the beginning of the second pulse which is a result of the digital FIR filter. Since we use a 90-tap FIR filter, the time needed to obtain the correct value for the quadrature we calculate by using $t_{\text{ring-up}} = 90 \cdot t_{\text{resol}}$, where $t_{\text{resol}}$ is the time resolution. For a demodulated signal with 22 point summation at a sampling frequency $f_s = 250$ MHz we obtain a resolution time of $t_{\text{resol}} = 88$ ns. Therefore, the ring-up time for the FIR filter is estimated to $t_{\text{ring-up}} = 7.92 \mu$s. Furthermore, Fig. 5.3 shows how the standard deviation of such demodulated quadratures

Figure 5.2: Time trace of a demodulated noisy coherent signal for different averaging numbers. Power used for the coherent signal was $-63$ dBm and 500 mV for the noise generated by the AWG. The ring-up time of the FIR filter can be observed in the middle of the trace.
behaves for 3 different SNRs. The SNR is calculated as follows

\[ SNR = \frac{V_{\text{signal}}^2}{V_{\text{noise}}^2}, \]  

(5.1)

where \( V_{\text{signal}} \) is the peak-to-peak amplitude of the coherent part and \( V_{\text{noise}} \) is the peak-to-peak amplitude of the Gaussian noise. As expected, we can fit the curves

\[ \sigma = \frac{A}{\sqrt{N}} + C, \]  

(5.2)

where \( A, C \) are fitting parameters and confirm the \( 1/\sqrt{N} \) behaviour. The fitting parameters are listed in Tab. 5.1.

Figure 5.3: Standard deviation of averaged quadratures calculated with dependence of number of averages for different signal to noise ratios (SNR) in a double logarithmic plot. For higher SNR we observe that the standard deviations are generally lower, since the thermal contribution is reduced. Furthermore, with increasing number of averages, the standard deviation also decreases with \( 1/\sqrt{N} \), where \( N \) is the number of averages.
Table 5.1: Fitting parameters for standard deviation measurement for different SNR.

<table>
<thead>
<tr>
<th>SNR</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000064</td>
<td>5.40 \cdot 10^{-4}</td>
<td>-2.11 \cdot 10^{-7}</td>
</tr>
<tr>
<td>0.000212</td>
<td>2.88 \cdot 10^{-4}</td>
<td>2.96 \cdot 10^{-8}</td>
</tr>
<tr>
<td>0.001024</td>
<td>1.22 \cdot 10^{-4}</td>
<td>8.84 \cdot 10^{-8}</td>
</tr>
</tbody>
</table>

Moreover, we observe that for the higher signal-to-noise ratios, the data points are shifted downwards since there are less fluctuations of the quadratures due to a relative reduction of the power of the noise. These measurements serve as straightforward benchmark of the FPGA-based moment calculations for various levels of input noise.

5.1.3 Phase stability

Figure 5.4: Phase of a coherent signal with a power of \(-19\) dBm for the input channel AI 0 of the FPGA was measured for increasing time. We observe a phase drift of \([0.10 \pm 0.01]^\circ/h\). Measurement points were taken manually by the author of the thesis, therefore there are no points between 15 and 22 hours.

Due to the fact that the signal-to-noise ratio is very small, averaging over many periods is required which can lead to long measurement times. Therefore, here
we investigate how phase of the reconstructed signal changes with time. For this test, a signal with a power of $-19\text{dBm}$ is generated by the SGS with noise added through the power divider (MCLI PS2-7) from the AWG and fed into the FPGA card. The FPGA card demodulates the signal into $I$ and $Q$ quadratures, filters it with the respective FIR filter, and averages over $2 \cdot 10^4$ traces. We calculate the phase with

$$\phi = \arctan \left( \frac{\langle Q \rangle}{\langle I \rangle} \right).$$

(5.3)

This procedure is repeated in intervals of one hour for approximately one day. The results are plotted in Fig. 5.4. A linear fit $\phi(t) = kt + c$ was used for the phase stability measurement, with $k = [0.10 \pm 0.01]^\circ / h$ characterizing the phase drift and $c$ an offset used for correct fitting. This drift usually occurs due to uncompensated temperature changes inside of the FPGA, the AWG, and the SGS. Essentially, slow temperature deviations affect phase stability and responses of output preamplifiers (AWG, SGS) and may also affect the internal clocks in the FPGA or in the Rb source itself.

### 5.1.4 Wigner function reconstruction

Knowledge of the first and second order quadrature moments is sufficient for the reconstruction of the Wigner function of a Gaussian state. Here, the Wigner function is measured for a coherent signal with a peak-to-peak amplitude of 500 mV and added noise of 200 mV for different phase-offsets $\phi$, which is directly controlled the SGS (Fig. 5.6). Furthermore, the Wigner function of a thermal

![Wigner function of a broadband Gaussian noise generated by the arbitrary wave generator with a peak-to-peak amplitude of 500 mV.](image)

**Figure 5.5:** Wigner function of a broadband Gaussian noise generated by the arbitrary wave generator with a peak-to-peak amplitude of 500 mV.
5.1 Self-reference benchmarking

state is reconstructed. A thermal noise signal generated from the AWG with an amplitude of 500 mV without a coherent signal was fed into the FPGA for the reconstruction. Fig. 5.5 shows the broadened Wigner function in the phase space. These Wigner reconstructions show that the FPGA-based tomography of propagating microwaves works as intended. Future reconstructions of squeezed microwaves still have to be done.

![Wigner functions](image)

Figure 5.6: Wigner functions of noisy coherent signal for different phase offsets $\phi$.

5.1.5 Second-order correlation function

With the implementation of the third and fourth order moments on the FPGA card, we can calculate the $g^{(2)}(0)$ function, which gives an insight in the photon statistics of an arbitrary signal (see chapter 2). We calculate the second-order
correlation function versus an averaging time (Fig. 5.7) by rewriting Eq. 2.23 in terms of $I$ and $Q$ statistical moments

$$g^{(2)}(0) = \frac{(I^4) + 2 \cdot (I^2 Q^2) + (Q^4)}{(I^2)^2 + 2 \cdot (I^2)(Q^2) + (Q^2)^2}.$$  \hspace{1cm} (5.4)

An important aspect of the $g^{(2)}(0)$ function is its relation to the noise. Noise contributions can originate from different sources, such as thermal noise from environment or AWG and quantization noise (created by the analog-to-digital converter due to amplitude digital quantization process - see chapter 2). Depending on the resolution of the ADC, digital quantization effects can have a higher impact on the signal-to-noise ratio for lower resolution. Since the converter resolution also depends on the sampling frequency, it is also important to consider it in order to make predictions on the second-order correlation function and hence the minimal number of bits which we can use for the ADC. Therefore, we develop a model for the $g^{(2)}(0)$ function depending on the noise generated by the A/D converter. The thermal background noise as well as the quantization noise are not correlated with the signal. Hence, the variances of all contributions add
up to the total variance of the input. As in Eq. 2.24, where we considered the thermal field being uncorrelated to the coherent signal and looked at the combined fields, the variance of the photon number is given by the variances of every field \( \text{Var}(\hat{a}^\dagger \hat{a}) = \text{Var}(n_{\text{th}}) + \text{Var}(c) + \text{Var}(b) \), where \( c \) is the photon number for the coherent field and \( b \) is the photon number due to quantization effect. The expectation value and the variance of the quantization effect are given only by the bit number of the ADC

\[
E[b] = \frac{1}{B},
\]

\[
\text{Var}(b) = \frac{B^2}{12},
\]

where \( B \) is the bit resolution of the analog to digital converter, and \( E[\cdot] \) is the expectation value. With this we obtain for the second-order correlation function

\[
g^{(2)}(0) = \frac{n_{\text{th}}^2 + B^2/12 - 1/B}{(n_{\text{th}} + |\alpha|^2 + 1/B)^2} + 1.
\]

Furthermore, we can describe the bit resolution with negative logarithmic frequency dependence, since for higher sampling frequencies the effective number of bits reduces [31]

\[
B = -k \cdot \log(f_s) + B_0.
\]

### 5.2 FPGA versus Acqiris benchmarking

#### 5.2.1 Comparison with Acqiris card

Comparative benchmark measurements of the FPGA card versus the Acqiris card are done for the calculation of the \( g^{(2)}(0) \) function. In these measurements we compare the time the Acqiris card and the FPGA needed for the calculation of the second order correlation function. For this comparative measurement we use the setups described in chapter 3. The second order correlation function is calculated for a coherent signal with a power of \(-53 \text{ dBm}\) generated by the SGS and a noisy environment created by the AWG with a peak-to-peak amplitude of \(500 \text{ mV}\). For the ADC card-based setup averaging up to \(10^6\) traces took about \(45 \text{ min}\), and for the FPGA-based setup \(12 \text{ min}\). For this measurement we use a trigger frequency of \(1.25 \text{ kHz}\). In Fig. 5.8 the \( g^{(2)}(0) \) function is plotted versus averaging time. The lowest measurement time (lowest average number) of the
Figure 5.8: Second-order correlation function calculated versus averaging time for the FPGA and Acqiris card. We observe the convergence of the second-order correlation function to the value $1$ but the FPGA reaches a typical number of averages $10^6$ roughly 3 times faster.

Acqiris card is limited to $\sim 12\,\text{s}$. This value is due to the memory of the ADC card which first needs to fill up to send the raw data for post processing. For the FPGA card, since the analysis is done on the card, the averaging time is calculated as follows:

$$t_{\text{FPGA}} = \frac{n_{\text{average}}}{f_{\text{trigger}}} + \frac{L_{\text{trace}} \cdot N_{\text{demod}}}{f_s} ,$$  \hspace{1cm} (5.9)

where $n_{\text{average}}$ is the averaging number, $f_{\text{trigger}}$ is the trigger frequency for the FPGA card, $L_{\text{trace}}$ is the time trace length, $N_{\text{demod}}$ is the number of the points used for the integration to calculate the quadratures, and $f_s$ is the sampling frequency. Since $f_s \gg f_{\text{trigger}}$ (250 MHz $\gg$ 10 kHz), we can neglect the second term of the equation 5.9 and approximate

$$t_{\text{FPGA}} \approx \frac{n_{\text{average}}}{f_{\text{trigger}}} .$$  \hspace{1cm} (5.10)
Naturally, the lowest achievable averaging time for the FPGA card is given for $n_{\text{average}} = 1$. For a trigger frequency of 10 kHz we obtain $t \approx 10^{-4}$ s for one average. Furthermore, we observe the convergence of the correlation function to the value 1 for a test coherent signal for both devices. We can not see a time improvement by calculating the $g^{(2)}(0)$ function from one measurement, since both devices are within 5% deviation from the actual value after $\sim 100$ s averaging time. Therefore, to see an actual improvement of the detection time, the standard deviation of an ensemble of measurements has to be analysed. Fig. 5.9 shows the time dependence of the calculated standard deviation of the correlation function of a coherent signal with added noise. We use a signal power of $-53$ dBm for the coherent signal. The standard deviation is calculated from an ensemble of 10 measurements for each point at a trigger frequency of 10 kHz for both devices. We can extract

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{fig5_9.png}
\caption{Standard deviation of the $g^{(2)}(0)$ function of a coherent signal with added Gaussian noise for increasing averaging time for two detection setups. We see a speed-up by approximately 7.47 times with the FPGA card when compared to the Acqiris card. Each point corresponds to the standard deviation of 10 different measurements.}
\end{figure}

several characteristics in Fig. 5.9. First, for low average time with the Acqiris card we can observe a non-linear increase in the double log plot for the standard deviation. This is due to the fact that acquiring data on the digitizer is limited
by the internal memory, which allows for a certain amount of storage on the card
followed by the signal processing on the computer. For low averaging number the
time for the data averaging will be the same, therefore the standard deviation
will be slightly higher. Only after the card is required to send the data to the
computer each time the memory is full, the process of moment calculation slows
down.

Further, we fit both plots with $a/\sqrt{\text{N}} + c$ curves which we expect to be the case
when looking at the standard deviation of averaged Gaussian noise. The values $a$
and $c$ are listed in Tab. 5.2 We compare the time needed to achieve a particular
standard deviation for both cards by using the parameters extracted from the fits
in Fig. 5.9. The results are shown in Tab. 5.3. We compare the two calculated

Table 5.2: Fitting parameters for the standard deviation of the second-order
correlation function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Acqiris card</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.3173</td>
<td>0.1148</td>
</tr>
<tr>
<td>$c$</td>
<td>$-3.3946 \cdot 10^{-4}$</td>
<td>$-2.6896 \cdot 10^{-4}$</td>
</tr>
</tbody>
</table>

Table 5.3: Averaging times calculated from the fit parameters for a particular
standard deviation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Acqiris card</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma(g^{(2)}(0)$</td>
<td>$6 \cdot 10^{-3}$</td>
<td>$6 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>Averaging time needed</td>
<td>2505.16 s</td>
<td>335.35 s</td>
</tr>
</tbody>
</table>

times by dividing each $t_{Acq}/t_{FPGA} = 7.47$. The speed-up factor is approximately
7.5 with the FPGA card is due to the fact that the signal processing occurs
directly on the card rather than first sending the raw data to the computer for
post-processing. Furthermore, the FPGA processes the data in a heavily parallel
fashion which also reduces the computation time, whilst on the computer we can
use at maximum 7 parallel loops for this task.

In this chapter we first checked our pulsing scheme with the FPGA card
described in chapter 3. Afterwards, we measured noisy signals and confirmed the
scaling of the standard deviation of $1/\sqrt{\text{N}}$, where $\text{N}$ is the averaging number.
Then, we reconstructed different Wigner functions for thermal, and coherent
thermal states. Last, we compared the time needed for to achieve a particular
value for the second order correlation function for both FPGA and ADC card.
Here, we observed a significant speed-up of $\sim 7.5$ which is important for future
measurements.
Chapter 6

Summary and outlook

During this master project we have developed a state reconstruction image (driver) for a modern field programmable gate array (FPGA, model NI 7972R). The programming of this image has been performed via the LabVIEW FPGA language. We have successfully tested the resulting image and respective measurement setup for experimental state reconstruction of typical microwave states such as coherent, thermal, and coherent thermal states. At the end, we have compared the new FPGA-based approach with the older ADC-based setup (with the Acqiris card). We have observed a speed-up with the FPGA card of approximately 7.5 times in a benchmarking measurement of the second order correlation function $g^{(2)}(0)$ as compared to the older ADC-based setup (with the Acqiris card). This is due to the fact that the Acqiris card has a limited internal memory and cannot perform any calculations directly on the board. Conversely, the FPGA is capable of data processing directly on the board in a highly parallelized fashion. By further optimizing the FPGA code and the experimental setup, such as the trigger timing, we expect a further time improvement for the microwave reconstruction up to $\sim 8$ times in respect to the Acqiris card reference.

Even further improvements of the FPGA-based microwave reconstruction can be achieved by using more powerful FPGAs. For the presented thesis the block random access memory (BRAM) of the FPGA card sets a tight limitation on trace lengths which also has an impact on the the detector performance. In the future, an FPGA with more internal BRAM can be used, such as the UltraSCALE+ Virtex FPGA designs offered by Xilinx. For these cards, BRAM memory can go up to 455 Mbits, which is more than sufficient to store traces up to 100,000 points at once and time improvements up to $\sim 20$ times could be expected. Another improvement could be achieved by making use of the dynamic random access memory (DRAM) which is placed physically outside the FPGA chip. This type of memory works at different clock rates than the FPGA internal clocks, but nevertheless a communication channel can be potentially established and data
can be logged onto the DRAM. Finally, by using a higher performance FPGA with more logic cells, the fixed-point precision of the calculated traces can be increased, leading to a more accurate reconstruction.

In terms of quantum communications with microwaves, the FPGAs are indispensable for many protocols. As a prime example, quantum teleportation relies on quantum entanglement and a feed-forward operation which consists of a local measurement, classical communication, and local displacement. However, all these tasks must be performed on a timescale less the relative dephasing time in entangled signals. This time is usually on the order of several hundred of nanoseconds \cite{32} and conventional ADC-based data processing setups are too slow for this. However, by using the demonstrated FPGA with the respective image it would be already possible to perform local measurements, feed-forward generation, and local displacement within the aforementioned dephasing time.


Acknowledgements

Here, I would like to appreciate to all the people who made this thesis possible and also a great experience of my life.

Prof. Dr. Rudolf Gross for giving me the chance to complete my master thesis in the qubit group at the Walther Meissner Institute. His remarkable knowledge and dedication in Physics motivates me to achieve more.

Dr. Kirill Fedorov for acting as my advisor throughout this work. I enjoyed many discussions not only about fascinating physics but also non-physics related topics. I have learned many things from him throughout this year. He was always available for questions and I want to thank him for his suggestions and solutions regarding my experimental and theoretical problems. It was an honor working with him.

Michael Renger and Stefan Pogorzalek for helping me with valuable ideas and suggestions concerning the many technical and theoretical problems I had.

To the qubit group for the interesting discussions and topics related to physics.

To my office colleagues Leander Peis, Raffael Ferdig, Manuel Mueller, Stephan Trattnig and all other master students. The atmosphere in the office was always very nice and we had many funny moments together.

My family for the continuous support and encouragement throughout my studies. My family was always there to offer me comfort whenever I was in stress situations. Without their support this thesis would not have been possible.

Finally, I thank my girlfriend Adela for being there for me and supporting me, even though I spent a lot of time at the university. She is an amazing girl and I’m grateful to have her in my life.