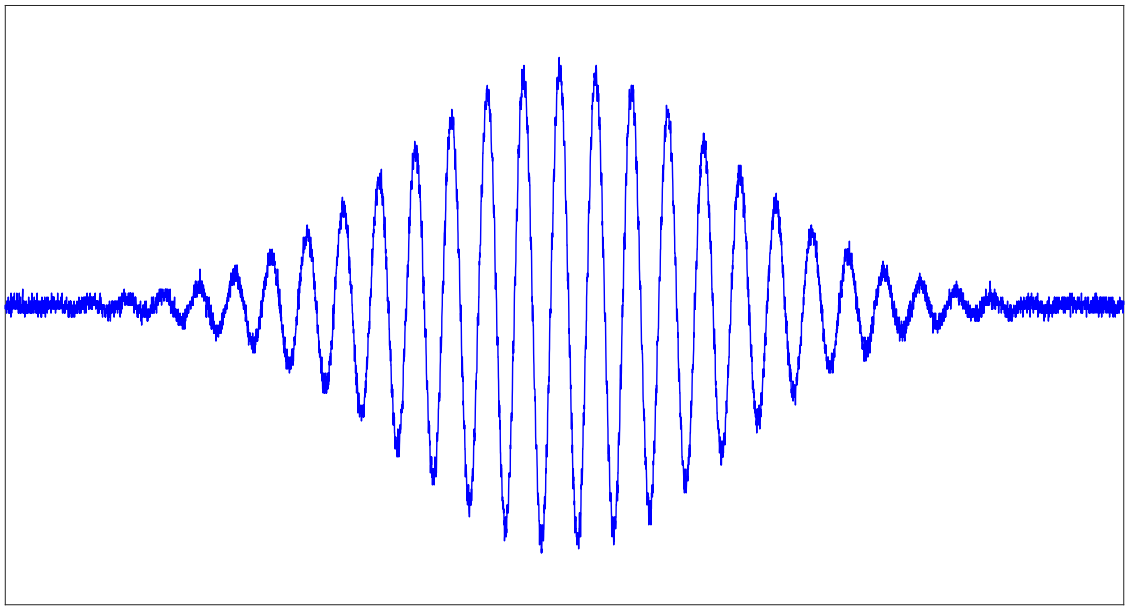
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



RF Pulse Generating AWG Design for Control of a Superconducting QPU

Project course in the Embedded Electronic System Design (MSc) program

HAMPUS LANG
GALO SANCHEZ
NICKLAS WRIGHT
SIMON JANSSON

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Embedded project 2024

RF Pulse Generating AWG Design for Control of a Superconducting QPU

HAMPUS LANG
GALO SANCHEZ
NICKLAS WRIGHT
SIMON JANSSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Microtechnology and Nanoscience
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

RF Pulse Generating AWG Design for Control of a Superconducting QPU
HAMPUS LANG
GALO SANCHEZ
NICKLAS WRIGHT
SIMON JANSSON

Supervisor: Lena Peterson, Department of Microtechnology and Nanoscience
Company advisors: Robert Rehammar & Christian Križan, Quantum Technology Lab, Chalmers University of Technology
Examiner: Lena Peterson, Department of Microtechnology and Nanoscience

Embedded Project 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Oscilloscope measurement of the implemented IF IQ modulator's output

Typeset in LaTeX
Gothenburg, Sweden 2024

RF Pulse Generating AWG Design for Control of a Superconducting QPU
HAMPUS LANG
GALO SANCHEZ
NICKLAS WRIGHT
SIMON JANSSON
Department of Microtechnology and Nanoscience
Chalmers University of Technology and University of Gothenburg

Abstract

The emerging field of quantum computing has shown great promise in the last few years. Quantum computers have even been proven to outperform classical computers in running some algorithms favoring highly parallel operations. In order to scale up the number of *qubits* in a quantum processing unit (QPU) the surrounding control circuitry must be scalable as well. As of today, arbitrary waveform-generators (AWGs) used to stimulate QPUs are far too expensive and power hungry for the number of qubits to potentially increase into the hundreds, thousands and beyond. Engineering trade-offs must therefore be made to design control circuitry that meets the requirements of the QPU without unnecessarily exceeding them. This report presents an all-digital implementation of an AWG for the purpose of generating microwave pulses to control a superconducting QPU. The design provides run-time control over frequency, phase, amplitude, pulse width, pulse spacing and resolution of the generated pulse. The target hardware for the implementation is the Zynq Ultrascale+ ZCU216 evaluation board. The design employs a two-stage digital upconverter (DUC) based on numerically controlled oscillators (NCOs) for sine- and cosine-wave generation. This setup enabled a tunable frequency-offset from a high-frequency (HF) carrier of about ± 40 MHz and a phase resolution of about 6.1 mrad. Amplitude control was achieved by a fixed-point multiplication algorithm allowing the amplitude to be scaled in 512 discrete steps with a maximum observed (relative) error of 0.06%. The resolution control was implemented by discarding the least significant bits (LSBs) of the intermediate-frequency (IF) signal, providing a configurable bit-resolution range of 1 to 15 bits. Due to some problems with sideband cancellation and limited testing of the UART receiver/decoder, the design was never run on an actual QPU. Thus, measurements of the effects on qubit *fidelity* when degrading the signal quality could not be obtained. This remains as future work.

Keywords: Quantum computers, Qubit, AWG, DDS, DUC, IQ modulator, RF, Pulse control, Mixing, NCO

Acknowledgements

The work of this project was long and arduous and would ultimately have borne little fruit had it not been for a few people that deserve our gratitude. First, and foremost, we would like to thank Christian Križan (PhD student) and Robert Rehammar (PhD) for their guidance and enthusiasm. Without tapping in to their great knowledge surrounding the subject of quantum computers and RF signal engineering this project would have been hard to – if not impossible – to finish on time.

Secondly, we would like to give our thanks to Lena Peterson (PhD) for aiding and guiding us in team and project management using the SCRUM method, as well as making sure the project was kept on the right track.

Finally, we would like to thank professor Thomas Eriksson at the division of Communications, Antennas, and Optical Networks at Chalmers for lending us the Zynq UltraScale+ RFSoC ZCU216 evaluation board. This piece of hardware was crucial to be able to fully test the AWG at higher carrier frequencies. Also we would like to extend our gratitude to Victor Åberg (PhD) at the Department of Microtechnology and Nanoscience at Chalmers for providing us with MATLAB scripts for storing data from the oscilloscope in the lab.

Hampus Lang, Galo Sanchez, Nicklas Wright, Simon Jansson, Gothenburg, May 2024

List of Abbreviations

Below is a list of acronyms frequently used throughout this report listed in alphabetical order.

AWG	Arbitrary Waveform Generator
CORDIC	Coordinate Rotation Digital Computer
DAC	Digital-to-Analog Converter
DDS	Direct Digital Synthesis
DUC	Digital Up-Converter
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GSPS	Giga Samples Per Second
HF	High Frequency
IF	Intermediate Frequency
IQ	In-phase and Quadrature
LO	Local Oscillator
LSB	Least Significant Bit
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
QPU	Quantum Processing Unit
RF	Radio Frequency
SNDR	Signal-to-Noise-and-Distortion Ratio
ROM	Read-Only Memory
SoC	System-on-Chip
SRAM	Static Random Access Memory
VHDL	(Very High Speed Integrated Circuit) Hardware Description Language

List of Symbols

Below is the nomenclature of the most commonly referenced parameters and variables in this report.

f_{c}	Carrier frequency [Hz]
f_{s}	Sampling frequency [Hz]
F	Qubit fidelity (%)
t_p	Pulse width [s]
e_r	Relative error (%)
T_1	Energy relaxation time of qubit [s]
T_2	Decoherence time of qubit [s]
Γ_ϕ	Pure dephasing rate [Hz]
σ	Standard deviation
$\|\psi\rangle$	Bloch vector

Contents

1

Introduction

As our knowledge of quantum physics has grown, it has given rise to a distinct type of computing called quantum computing. Quantum computing is an emerging field with great promise to solve certain problems at a pace far beyond what is achievable with classical computing [1]. Instead of the classical bits in classical computing, quantum computers use the analogous "qubits" with the difference being that they may be in a superposition of both '1' and '0' [2]. The entangled state of the qubits create an exponentially increasing vector space of different states which quantum algorithms exploit for massively parallelized operations [1]. In other words, N qubits can represent the equal number of states as 2^N classical bits. While this might not be exploitable by every algorithm to the point of them always being faster on a quantum computer rather than a classical computer, for some it enables solutions no classical computer could ever give [1].

1.1 Background

A lot has happened within the field of quantum computing since the renowned quantum physicist Richard Feynman first theorized them in 1981 [1]. It is not uncommon for the quantum computers of today to have hundreds of qubits each. Just last year, a quantum processor (QPU) containing a staggering 1000 qubits was unveiled by IBM [3].

While there are many different types of quantum computers exploiting different quantum phenomena, the focus of this report is on a superconducting quantum computer. Each qubit in such a QPU can be "programmed" by exposing it to a certain microwave pulse where the parameters of the pulse – for example phase, width, amplitude, or frequency – can determine the state of the qubit within a certain probability [2]. A precise and tunable waveform generator, known as an arbitrary waveform generator (AWG), is therefore a necessity. Such AWGs are implemented in digital hardware, with a high resolution digital-to-analog converter (DAC) producing the analog output used for the qubit control. Such high-performing AWGs unfortunately come with a high cost and typically with relatively high power consumption. With the number of qubits of the average QPU increasing, it has therefore become crucial to minimize the cost per qubit for quantum computers to find any realistic applications outside of research labs.

1.2 Problem Statement

To achieve realistic and scalable qubit control for the future, it is of the essence to have a good understanding of how different design characteristics affect system performance. Once it is known which performance metrics can be limited while still achieving good qubit control, the cost of the control system may be reduced. To investigate this further, researchers at Chalmers Quantum Laboratory have found it desirable to develop an AWG capable of degrading different parameters of the transmitted pulse in order to observe what minimum level of precision is needed to adequately control the qubit's state. Of particular interest is the ability to modify the bit resolution of the digital signal prior to the digital-to-analog conversion. This report details the design and implementation of an AWG capable of such resolution control.

1.3 Aim

The aim of this project is to create an AWG on the Zynq UltraScale+ ZCU216 evaluation board for controlling a single qubit. This entails mixing a carrier wave with an envelope to some desired frequency in the GHz range. The AWG should be as customizable as possible, since many different parameters can affect the state of the qubit. Most importantly, the AWG should have an adjustable bit resolution in order to see the effects of using a degraded signal to stimulate the qubit.

The requirements for the AWG can be divided into two main sections: the minimum requirements needed for conducting common experiments on the qubit, and the requirements needed for a perfect AWG. "Perfect" in this context means that the AWG can generate very high-quality waveforms, as well as its parameters being almost entirely tunable during run-time. The entire product specification can be found in Appendix A of this report.

1.4 Delimitations

To have achievable aims for the finished design, some delimitations as to what this project will focus on are required. Most notable is that it is assumed that the AWG will only be used to control one single qubit.

Furthermore, since the focus of this project is specifically on the AWG implementation, testing of the AWG on a real qubit will not be performed. This also means the qubit response will not be measured and that the power of the output signal is limited to only be clearly above any noise.

The generated output signal of the AWG will not be filtered by any external analog band-pass filter. This is because we want to see how the AWG performs on its own without any external influences, except noise, even if they constitute an improvement of the signal quality.

Finally, there are no set restrictions on the design in terms of hardware utilization, such as memory. Instead the main focus will be on creating a as flexible and compu-

tationally fast design as possible to achieve the requirements outlined by the product specification.

1.5 Report Outline

In Chapter 2, we present the theory needed to comprehend the design choices made in this project. In that chapter, Section 2.1 and 2.2 introduces some necessary quantum computing concepts while Section 2.3 and 2.4 presents the theory behind waveform generation in digital logic. Chapter 3 presents the relevant details on the hardware used in the project. In Chapter 4, the overall design and implementation of the AWG is detailed with explanations of the most important parts of the design. The results of a multitude of different performance measurements on the final AWG implementation are shown in Chapter 5. Lastly, conclusions drawn from analyzing the results of the tests are presented in Chapter 6, together with some suggestions for future improvements.

2

Theory

In this chapter the underlying theory will be presented to motivate design choices made when designing the AWG. It will first give the reader a very brief introduction to quantum computing in Section 2.1 and 2.2. Section 2.3 and 2.4 will then present some signal processing theory relating to digital modulation and digital waveform-synthesis.

2.1 Quantum Logic

In classical computing, the logic state of a single bit is represented using binary notation. That is, the value of a classical bit is either a logical '0' or '1'. These logical values correspond to a voltage in some implementation-specific range. For example, a system could allow voltages ranging from 0 - 0.5 V where 0 - 0.25 V is interpreted as a logic '0' and 0.25 - 0.5 V as a logic '1'. This way all voltages get discretized to represent either '0' or '1' and the possibility of storing any additional information as a voltage between 0 and 5 V is lost. This is in contrast to quantum logic where the state of a single quantum bit, known as a *qubit*, is a superposition of both a logic '0' and '1'. This is often described by a sphere formally referred to as the *Bloch sphere* [2]. The Bloch sphere, which has a unity radius, can be used to represent a two-level quantum mechanical system. The north pole of the sphere is commonly chosen to represent logic '0' and the south pole to represent logic '1', though other conventions exist. By denoting the logical-low state with $|0\rangle$ and logical-high state with $|1\rangle$, which is standard notation when describing quantum states, we can describe the state of a single qubit in a Cartesian coordinate system using the complex *Bloch vector*, $|\psi\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + j\beta\,|1\rangle\,,\ \alpha, \beta \in [-1, 1] \tag{2.1}$$

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

were α and β is the scalar factor of the Bloch vector in the $|0\rangle$ and $|1\rangle$ direction from origo respectively. From the condition of the unit sphere described by (2.2), for the case when α and β have equal magnitude, the equation describes the unit circle along the XY-plane. With it the end points of the x-axis can be seen to be

$$|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle \pm \frac{1}{\sqrt{2}}\,|1\rangle \tag{2.3}$$

And similarly, the end points of the y-axis are

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle \pm j\frac{1}{\sqrt{2}}|1\rangle \tag{2.4}$$

The Bloch vector is drawn from origo to a point on the surface of the Bloch sphere described in (2.2), which represents the current qubit state. This is illustrated in Fig. 2.1, where the Bloch vector points to an arbitrary location on the Bloch sphere-surface and the $|0\rangle$ and $|1\rangle$ states are marked (also commonly referred to as the *ground state* and the *excited state* respectively). The x and y axes represent superpositions between the states $|0\rangle$ and $|1\rangle$ on the z-axis.

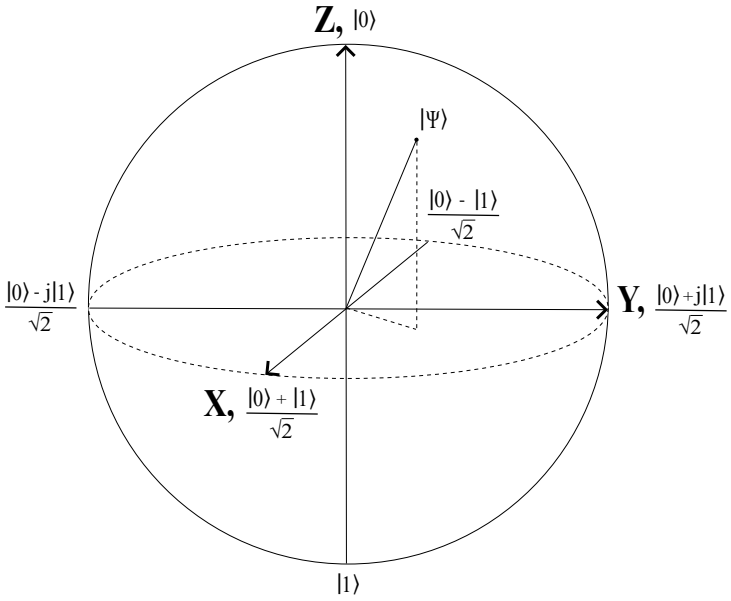


Figure 2.1: Bloch sphere with the Bloch vector pointing to an arbitrary qubit state.

It should be noted that in reality, the Bloch vector rotates around the z-axis at the frequency of the qubit [4]. This could give rise to some (additional) confusion. We can therefore choose to let x- and y-axis rotate about the z-axis at the same frequency, allowing us to visualize the Bloch vector as being stationary, as in Fig. 2.1.

Furthermore, it is important to emphasize that measuring the qubit state destroys the quantum property, and the qubit state will then during readout be either a logic '1' or '0'. This is known as *collapsing* the quantum system. In order to gain the benefits of computational parallelism that a quantum computer provides, it is therefore necessary to avoid collapsing the qubit state until the system has finished its desired operation.

2.2 Qubit Control

Now that we have established how the qubit state is represented, the question on how to manipulate this state arises. In classical computing, individual bits are manipulated by flipping their voltage potential between two discrete reference-levels. In quantum computing, however, a change of state in the qubit can be achieved by subjecting it to a microwave pulse of a certain frequency, amplitude and pulse-width [5]. More generally, this can be described as manipulating the energy level of the qubit. From Planck's relation we know that energy can be expressed as

$$E = hf \tag{2.5}$$

where h is Planck's constant and f denotes frequency. The energy required to force a state-change of the qubit puts us in the microwave domain in terms of frequency, hence the requirement of microwave pulses as stimuli. Subjecting the qubit to such pulses results in a rotation of the Bloch vector about some axis in the Bloch sphere, which can be considered as the quantum equivalent to flipping a bit.

The nature of the above-described control pulse lends itself to be implemented using standard electrical engineering-techniques. Microwave pulse-generation is a well established practice within radio frequency (RF) engineering, thus, many of the concepts can also be applied in qubit control-circuitry.

The reader should be aware of the fact that the control of the qubit is specific to the implementation of the quantum system in which it resides. Hence, there exists many different ways of controlling the qubit state that are not discussed in this report. Some of these other approaches are mentioned in [5].

A common figure of merit when controlling the state of the qubit is the qubit fidelity, F. The fidelity determines the probability that the measured state of a qubit after a certain quantum operation will be the desired one. For example, if a desired outcome of a quantum program is that the qubit should be in a state of $|1\rangle$ after collapsing the state, a fidelity of 1.0 would mean that there is a 100 % probability that the qubit state will be $|1\rangle$ and 0 % probability of it being $|0\rangle$. The fidelity of a single qubit, rotating along the x and z axes of the Bloch sphere, is closely related to the pulse width, t_p, of the RF pulse by the following expression [6]:

$$F = 1 - \frac{\Gamma_1 + \Gamma_\phi}{3} t_p + \frac{1}{8}\left(\frac{11}{12}\Gamma_1^2 + \frac{5}{3}\Gamma_1\Gamma_\phi + \Gamma_\phi^2\right) t_p^2, \tag{2.6}$$

where $\Gamma_1 = \frac{1}{T_1}$ and $\Gamma_\phi = \frac{1}{T_1} - \frac{1}{2T_2}$ are the energy relaxation rate and pure dephasing rate of the qubit respectively, with T_1 and T_2 being the energy relaxation time and decoherence time. The pure dephasing rate in the xy-plane is caused by noise along the z-axis that causes the qubit frequency to vary [4].

T_1 can be obtained through measurements on the qubit. Providing enough energy to the qubit to excite it from a state $|0\rangle$ to a state $|1\rangle$ can be done by sending a so-called π-pulse to the qubit [2]. From Fig. 2.1 we can see that the state transition requires a rotation of π radians along the origin, which explains its name. After the pulse is sent, the state of the qubit is read after some delay time τ (0 at first).

The test is then repeated many times with different values of τ in order to estimate the probability. The results from such measurements show that the probability of the state being high, $P(|1\rangle)$, decays exponentially as τ increases as $e^{-\tau/T_1}$ (Fig. 2.2). Thus, T_1 is defined as the characteristic time constant of a decaying exponential function, and can be measured from when $P(|1\rangle) = e^{-1} \approx 0.368$. In other words, when $\tau = T_1$.

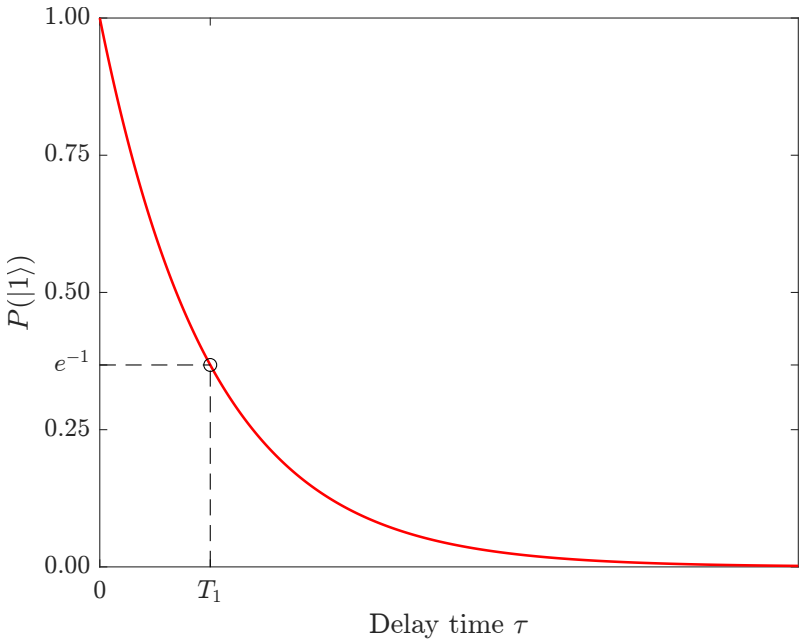


Figure 2.2: Typical plot of $P(|1\rangle)$ as the delay time between pulses, τ, increases. Note that $P(|1\rangle) = 0$ is equivalent to $P(|0\rangle) = 1$.

T_1 can therefore be seen as the time it takes for the qubit to reach its steady-state, which in the case of superconducting qubits is the ground state [4].

Similarly, the dephasing time T_2 is instead the time it takes for the Bloch vector to flip from a superposition state along the equator of the Bloch sphere to a readout state of $|0\rangle$ or $|1\rangle$ on the z-axis [4]. This is achieved by forcing the qubit to the xy-plane by applying an $\frac{\pi}{2}$-pulse to it, which moves the vector $\frac{\pi}{2}$ radians; putting it in a superposition state. After a time delay, t, another $\frac{\pi}{2}$-pulse is transmitted, which projects the superposition state to the z-axis in a state of either $|0\rangle$ or $|1\rangle$ where it then can be read by a subsequent readout pulse. This process is repeated in a series of measurements to get probability estimations, with t being increased after each measurement series. Since the xy-plane is rotating around the z-axis of the Bloch sphere at a constant rate [4], each $\frac{\pi}{2}$-pulse can be viewed as randomly placing the Bloch vector at a position on the edge of a rotating disc. Therefore, we can describe the probability function of the qubit as some sort of oscillating function.

It can be shown that increasing t between the first and second $\frac{\pi}{2}$-pulse before each readout, in what is known as a Ramsey interferometry experiment, will attenuate this probability function's oscillations in an exponential manner as seen previously in Fig. 2.2, however, with it instead approaching equilibrium at 50-50 probability

of $|0\rangle$ or $|1\rangle$ when $t \to \infty$ [4]. In other words, it can be viewed as the Bloch vector becoming shorter as the time between pulses increases. With the decay being described by approximately e^{-t/T_2}, the dephasing time can be estimated in a similar fashion as T_1. Fig. 2.3 shows an example plot of a fictitious measurement series, from which T_2 then is determined.

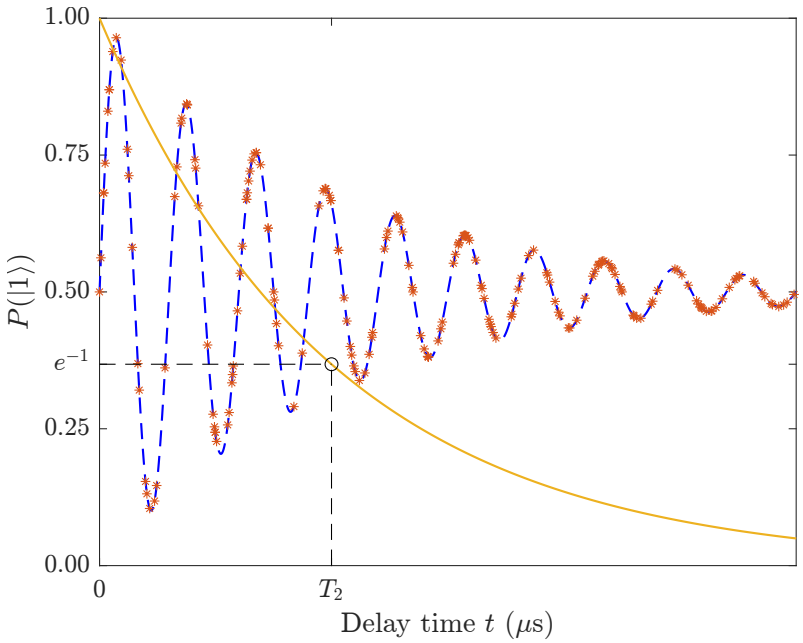


Figure 2.3: Typical plot of $P(|1\rangle)$ as the delay time between $\frac{\pi}{2}$-pulses, t, increases between successive readouts of qubit state. Note again that $P(|1\rangle) = 0$ is equivalent to $P(|0\rangle) = 1$.

Previous measurements done in Chalmers Quantum Laboratory have shown that achievable values for T_1 and T_2 are in the range of 79 µs ± 25 µs and 33 µs ± 10 µs respectively, and will as a result be used in this report as a reference.

2.3 IQ-signals and Quadrature Modulation

As stated in the previous section, common practices within RF engineering can be applied in the generation of pulses for qubit control. One such practice is the use of IQ signals for modulating an envelope onto a carrier wave. This is a heavily used technique for implementing different modulation schemes in modern wireless and digital communication-systems [7]. Its popularity can largely be attributed to the fact that high bandwidth-efficiency can be achieved since both amplitude and phase can be used to encode information.

Before the upcoming derivations in this section, a review of IQ signals is warranted. The I and Q stand for *in-phase* and *quadrature* or, equivalently, *real* and *imaginary*. These in-phase and quadrature components can be combined to form a two-dimensional complex signal which is commonly referred to as an IQ signal [8].

Euler's identity provides a very compact notation for such a signal, namely

$$e^{j\phi} = \cos\phi + j\sin\phi \tag{2.7}$$

or

$$e^{-j\phi} = \cos\phi - j\sin\phi \tag{2.8}$$

The introduction of the j-operator is a source of confusion for many. One can spend a lot of time trying to devote some physical meaning to it, however this serves no purpose. The advantages of complex representation of a signal lies in its mathematical properties [8]. For example, trigonometric equations are reduced to simple algebra of exponential functions and the summing of signals can be performed as vector addition. We need not to worry about how to represent j in hardware since the definition of the j-operator is only implied by its function. Thus, we implement it by the way we choose to interpret our signals. Using Euler's formula, as described in (2.7) and (2.8), we can express the real cosine and sine functions as

$$\cos(\omega t) = \frac{e^{j\omega t}}{2} + \frac{e^{-j\omega t}}{2} \tag{2.9}$$

$$\sin(\omega t) = j\frac{e^{-j\omega t}}{2} - j\frac{e^{j\omega t}}{2} \tag{2.10}$$

where ω is the angular frequency. We can conveniently use these definitions to derive the output of a *quadrature modulator*, as depicted in Fig. 2.4.

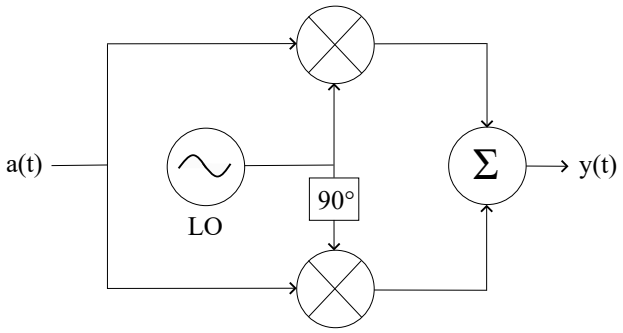


Figure 2.4: Block diagram of a quadrature modulator.

The 90° out-of-phase carriers can be realized using sine and cosine waves. Doing so yields an expression for the modulator output as described in (2.11).

$$y(t) = a(t)\left(\cos(\omega_{\mathrm{LO}}t) + \sin(\omega_{\mathrm{LO}}t)\right) \tag{2.11}$$

The two carriers, or local oscillators (LOs), operate at the same angular frequency of ω_{IF}. Summing sines and cosines of equal amplitude yields a cosine wave with and additional phase of 45°. It is therefore easy to see that the above expression is equal to a cosine wave with an *amplitude modulation* (AM) of $a(t)$.

If the above-described signal would be used to stimulate a qubit, the frequency ω_{LO} would correspond to the resonance frequency of the qubit. The exact value of this frequency is not known before calibration of a quantum processor. Calibration involves sweeping the frequency in the vicinity of where the qubit is believed to exist [2]. This can be achieved by generating a signal at tunable offset from a fixed carrier, such that the frequency of the modulated wave becomes $\omega_{\mathrm{HF}} \pm \omega_{\mathrm{IF}}$ where HF and IF denote high frequency and intermediate frequency. In hardware this can be realized by supplying each of the output channels (before summation) of a quadrature modulator operating at ω_{IF} to the input channels of a second modulator operating at ω_{HF}. This is described in Fig. 2.5.

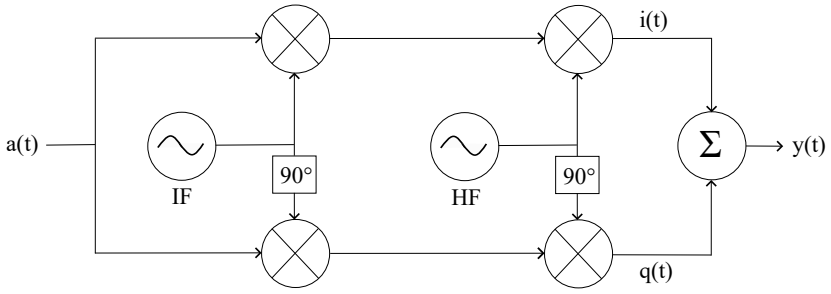


Figure 2.5: Two-stage quadrature modulator.

With the above configuration we can derive the outputs in each channel, $i(t)$ and $q(t)$, as

$$i(t) = \frac{a(t)}{2}\left(\frac{e^{j(\omega_{\mathrm{HF}}+\omega_{\mathrm{IF}})t}}{2} + \frac{e^{-j(\omega_{\mathrm{HF}}+\omega_{\mathrm{IF}})t}}{2} + \frac{e^{j(\omega_{\mathrm{HF}}-\omega_{\mathrm{IF}})t}}{2} + \frac{e^{-j(\omega_{\mathrm{HF}}-\omega_{\mathrm{IF}})t}}{2}\right) \tag{2.12}$$

$$q(t) = \frac{a(t)}{2}\left(\frac{e^{j(\omega_{\mathrm{HF}}-\omega_{\mathrm{IF}})t}}{2} + \frac{e^{-j(\omega_{\mathrm{HF}}-\omega_{\mathrm{IF}})t}}{2} - \frac{e^{j(\omega_{\mathrm{HF}}+\omega_{\mathrm{IF}})t}}{2} - \frac{e^{-j(\omega_{\mathrm{HF}}+\omega_{\mathrm{IF}})t}}{2}\right) \tag{2.13}$$

As can be seen, both expression contain the upper and lower *side-bands*. That is, the positive and negative frequency offsets. It is easy to see that by summing the outputs of the two channels we can cancel the upper side-band by means of destructive interference. Thus yielding

$$\begin{aligned} y(t) = i(t) + q(t) &= a(t)\left(\frac{e^{j(\omega_{\mathrm{HF}}-\omega_{\mathrm{IF}})t}}{2} + \frac{e^{-j(\omega_{\mathrm{HF}}-\omega_{\mathrm{IF}})t}}{2}\right) \\ \Leftrightarrow y(t) &= a(t)\cos\left((\omega_{\mathrm{HF}} - \omega_{\mathrm{IF}})t\right) \end{aligned} \tag{2.14}$$

Assuming that ω_{IF} is tunable, the above configuration allows offsetting the modulated signal from the carrier frequency by $-\omega_{\mathrm{IF}}$, where ω_{IF} can be an arbitrary frequency. The question now arises of how to generate a positive offset? This is achieved by a small change in the two-staged modulator, as presented in Fig. 2.6

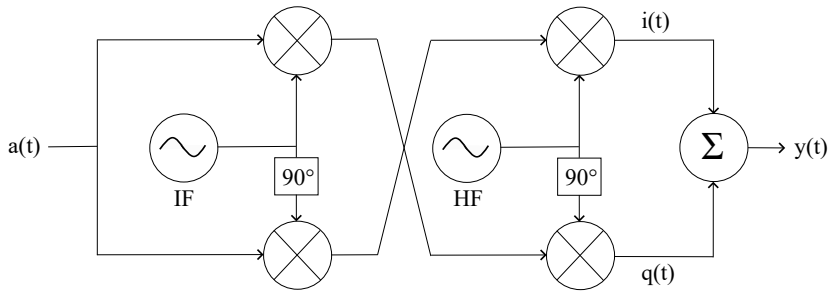


Figure 2.6: Two-stage quadrature modulator for generating the upper side-band.

The inputs to the second stage of the modulator are now flipped. Using the same approach as above, we can now derive the modulator output as

$$\begin{aligned} y(t) = i(t) + q(t) = a(t) \left(j\frac{e^{-j(\omega_{\mathrm{HF}}+\omega_{\mathrm{IF}})t}}{2} - j\frac{e^{j(\omega_{\mathrm{HF}}+\omega_{\mathrm{IF}})t}}{2} \right) \\ \Leftrightarrow y(t) = a(t)\sin\left((\omega_{\mathrm{HF}} + \omega_{\mathrm{IF}})t\right) \end{aligned} \tag{2.15}$$

With the configurations in Fig. 2.5 and Fig. 2.6 we can thus create an arbitrary frequency offset from the carrier ω_{HF} of $\pm\omega_{\mathrm{IF}}$. Furthermore, we can also generate a tunable phase by providing a phase offset to the IF carriers. This yields the following expressions for the lower and upper side-bands

$$y_{lower}(t) = a(t)\cos\left((\omega_{\mathrm{HF}} - \omega_{\mathrm{IF}})t - \theta_{\mathrm{lower}}\right) \tag{2.16}$$

$$y_{upper}(t) = a(t)\sin\left((\omega_{\mathrm{HF}} + \omega_{\mathrm{IF}})t + \theta_{\mathrm{upper}}\right) \tag{2.17}$$

In order to get same phase reference for both side-bands, the two phase offsets can be provided as $\theta_{\mathrm{lower}} = -\varphi$ and $\theta_{\mathrm{upper}} = \varphi + \frac{\pi}{2}$ where $\varphi \in [0, 2\pi]$. Doing so ensures that we always provide the phase as a positive offset in relation to a cosine wave.

By cancelling unwanted sidebands using destructive interference we omit the need for filters in the processing chain. This is advantageous since the project aims to explore the effects of using lower quality signals for qubit control.

2.4 Direct Digital Synthesis

When implementing IQ modulators using analog mixers, a number of impairments have to be considered. Amongst them are quadrature errors, carrier feed-through and IQ skew [9]. These impairments can be circumvented by implementing the signal generation digitally using direct digital synthesis (DDS). This can be achieved by combining an AWG with a digital IQ modulator, also known as a digital up-converter (DUC), and outputting the waveform using a high-speed RF DAC. Doing so allows the modulation scheme to be implemented using memory-based oscillators (see Section 2.4.1) and simple arithmetic circuitry.

Since this project aims to explore the effects on the qubit fidelity when degrading the signal quality of the control pulses, implementing the waveforms mathematically using DDS allows different types of distortion to be simulated by simply extending the mathematical model of the waveform.

Direct digital synthesis involves techniques and algorithms used to generate different types of waveforms in digital systems [10]. The algorithm explained below, is one of the various options available for waveform generation.

2.4.1 Numerically Controlled Oscillator

A numerically controlled oscillator (NCO) is a widely used topology in systems in which digital signal generation is required. It is normally used for the synchronous generation of sinusoidal signals. The simplest implementation is composed of two blocks: a phase accumulator (PA) and a phase-to-amplitude converter (PAC), which can be in the form of a look-up table (LUT) or read-only memory (ROM).

The primary feature of the NCO is its ability to dynamically control the frequency of the digital signal it generates, which is accomplished by adjusting the input frequency-setting word (FSW) of the device. This capability is demonstrated in (2.18), where all other parameters are fixed during the design phase.

$$f_{\text{out}} = \text{FSW} \cdot \frac{f_{\text{clock}}}{2^N}, \tag{2.18}$$

where N is the number of bits in the FSW.

3

Hardware

The AWG implementation was done with the help of two different field-programmable gate array (FPGA) boards. First a less capable Nexys A7-100T board was used for basic testing of some modules early on in the project. The final system implementation was done on the much more capable Zynq Ultrascale+ RFSoC ZCU216 evaluation board to be able to output the desired high-frequency analog signals.

3.1 FPGA

FPGAs are re-configurable semiconductor devices composed of a multitude of different controllable logic blocks. By configuring the logic blocks and their interconnects digital hardware logic can be mapped onto the constituent logic blocks. For improved performance FPGAs commonly also include non-configurable logic as the reconfigurability carries with it a significant overhead. An example of non-configurable logic used for this project is the internal DAC on the Zynq board.

The digital logic which is mapped onto the configurable logic blocks is described in code with hardware description languages (HDLs) such as the one used in this project; VHDL. The distilling of the code to a set of settings given to the FPGA are done in steps of which the ones relevant to mention in this report are in order: synthesis, implementation and bitstream generation. In synthesis a description in code is turned into a map of connections between registers and logic gates called a netlist. By generating an implementation, the netlist is turned into a set of transistor connections specific to the FPGA it is meant to run on. When generating the bitstream the transistor mappings are turned into a set of data which when downloaded onto the FPGA reconfigures it to implement the desired logic. Simulation can be done before synthesis, in which case it is called behavioral simulation, but also after both the synthesis and implementation steps. The validity of the bitstream is tested by downloading it to the FPGA and testing it there. To perform the steps here outlined and help with the simulation, the Vivado Design Suite by AMD was used.

3.2 Nexys A7-100T

The Nexys board is comparatively simpler than the Zynq board and was used to easily get started on the AWG design. Unlike the Zynq board it does not have an internal DAC which means it is only possible to read a digital output from it. The digital output can be sent through PMOD ports and collected by a digital analyzer.

The board has a single 100 MHz oscillator used for the system clock from which phase-locked loops (PLLs) can increase the frequency up to 450 MHz [11]. Since this is below the required output frequency as mentioned in Section 1.3, it would not be possible to satisfy the system requirements by only using the Nexys board. Due to both the lack of a DAC and the too low maximum frequency output of the Nexys board, it was necessary to use the Zynq board for the complete design.

3.3 Zynq UltraScale+ RFSoC ZCU216

The Zynq board is much more capable than the Nexys board which made it suitable for the final AWG design. It most notably has 16 DAC channels which can output frequencies in the GHz range as they have a maximum sample rate of 9.85 GSPS [12]. Note that the DAC sample rate in the data sheet is given for what RFSoC the board uses, which for the ZCU216 board is the ZU49DR. The Zynq UltraScale+ RFSoC RF Data Converter AMD LogiCORE IP has been used to interface with the DAC.

Included with the Zynq board as part of an evaluation kit were add-ons of which some were used. Most importantly was the CLK104 RF Clock Add-On Card used for the carrier frequency of the DAC as well as a synchronization signal for the different DAC channels. We also used the XM655 add-on card to break out the DAC output into cables from which the analog output could be measured with external instruments.

4

Design and Implementation

This chapter aims to describe the functionality of the implemented system. Section 4.1 presents the block diagram of the entire system. The implementation of NCO topology presented in the theory is shown in Section 4.2. Sections 4.3 to 4.4 show the implementation of the control modules. The communication between the host computer and the FPGA is explained briefly in Section 4.5.

4.1 Microwave pulse-generator using DDS

A simplified block diagram of the qubit control system is shown in Fig. 4.1. The system is divided up into two different sets of hardware. One is the host computer which acts as a control unit and the other is the digital logic in the FPGA. With instructions given to the host computer through a custom MATLAB-script, an arbitrary pulse-shape is generated which is sent to the waveform memory RAM-block in the FPGA via UART along with run-time configurable parameters. Once the data has been loaded, a start command is sent by the receiver to start feeding the data to the DAC. A confirmation signal is sent back to the host when the complete waveform stored in the memory has been played. The host may then send a new waveform or manipulate the parameters in order to create a pulse sequence.

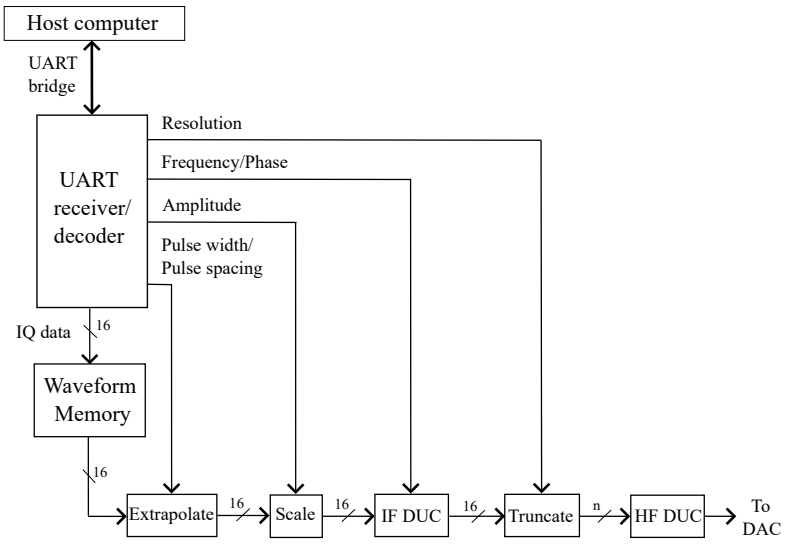


Figure 4.1: Simplified block diagram of the system.

Note that the above block diagram only displays the data path for one of the I- and Q-channels for simplicity. They are however identical. Resolution, carrier frequency, phase, amplitude, pulse width, desired number of pulses, and repetition rate between programs are all parameters that can be altered at run-time. That is, without having to reload the waveform memory.

The desired resolution is achieved simply by truncating the least significant bits (LSBs) of the output from the DUC. This means that all operations on the FPGA are always done at the maximum resolution of 16 bits, which preserves arithmetic accuracy. Note that the DAC output is only 14 bits, so some additional resolution degradation occurs there. The frequency of the carriers in the IF DUC is controlled using NCOs, as described in Section 2.4.1. Phase control is achieved as described in Section 2.3, that is, by adding a phase offset to the IF NCOs. The pulse width can be increased by duplicating the elements in the waveform memory. This means the pulse width can only be increased by an integer factor of the pulse width of the waveform in the waveform memory. Also controllable at run-time is the spacing in time between pulses. Control of pulse width and spacing has been summarized as "extrapolation" in the above block diagram.

4.2 NCO Implementation

The previously described NCO algorithm is used to generate sinusoidal waves to be used as LOs, the implementation strategy involves the utilization of a pair of ROM memories with pre-computed values, with the aim of minimizing the propagation delay of the system, thus enabling higher clock frequency utilization. The proposed topology requires a driving block that traverses through the memory address to produce the desired waveform, with both the memory and phase accumulator interconnected by a D-flip flop, as shown in Fig. 4.2.

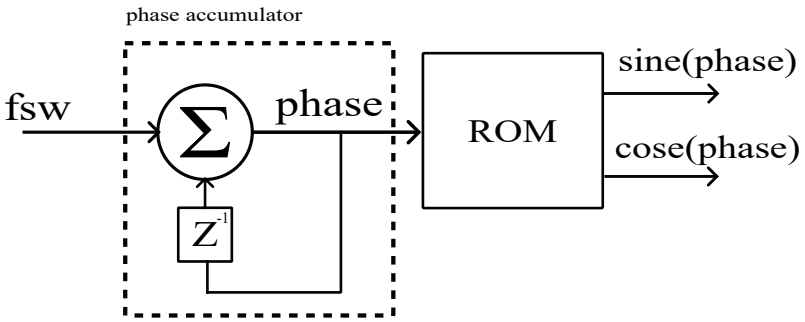


Figure 4.2: Memory based NCO functional block diagram

4.3 Frequency and Phase Control

The digital data-path of the DUC and RF up-converter is displayed in Fig. 4.3. The operations to the right of the multiplexers are carried out in the RF DAC on the Zynq board. It should be noted that the pipeline registers have been omitted for clarity. Also, only the parts relating to frequency and phase control are shown, i.e. other control modules such as amplitude and resolution are not included in the block diagram.

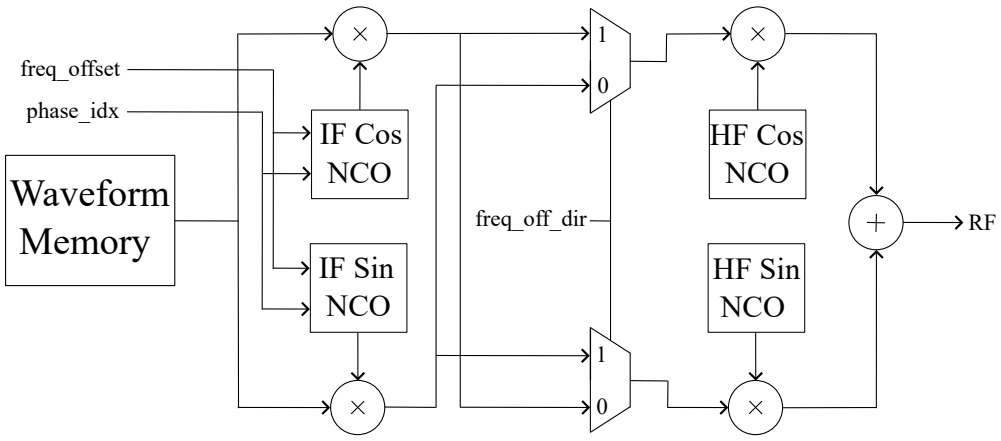


Figure 4.3: Digital data-path of the modulator. (Note: four-way crossings of wires do not constitute an electrical connection.)

4.3.1 Frequency

The frequency offset from the high-frequency carrier can be controlled at run-time using the input variables `freq_offset` and `freq_off_dir`. With `freq_offset` the 10-bit FSW to the IF NCOs are set. This value relates to the desired offset in Hz by Eq. (2.18). The 409.6 MHz system clock on the Zynq board yields a theoretical offset range of 400 kHz − 409.2 MHz. However, the highest useful frequency is ultimately limited by the waveform quality as the NCOs skip samples in order to increase the frequency. A waveform with 10 samples in a period, which is a reasonable amount, results in a maximum obtainable frequency offset of about ±40 MHz.

The direction of the offset is controlled with `freq_off_dir`. A value of '1' gives a negative offset (lower sideband) and '0' gives a positive offset (upper sideband) in accordance with the theory presented in Section 2.3.

The centre frequency, i.e. the high-frequency carrier, can be set by configuration of the RF DAC on the Zynq board to a frequency of up to several GHz. This is enough to cover the resonance frequency of the qubit which is assumed to be just shy of 4 GHz.

4.3.2 Phase

To set a phase offset, the run-time configurable variable `phase_idx` can be used. Its value corresponds to an offset given in number of samples which is added to the phase-accumulator value in the IF NCOs. For the lower sideband this value can be calculated as

$$phase\_idx = 2^N \cdot \frac{(360 - \varphi)}{360} \in \mathbb{Z} \tag{4.1}$$

and for the upper sideband

$$phase\_idx = 2^N \cdot \frac{(90 + \varphi)}{360} - 2 \in \mathbb{Z} \tag{4.2}$$

where N is the address depth of the NCO ROM-memory and φ is the desired phase offset in degrees. The -2 appearing in the expression for the upper side-band is an experimentally derived correction constant. With $N = 10$ we obtain a a phase resolution of about $0.35°$ or 6.1 mrad. This resolution is not affected by the fact that the NCOs dump samples in order to reach higher frequencies since the offset is set before the phase-accumulation process starts. In theory, the obtainable phase-resolution is only limited by the available memory. However in practice it will ultimately be limited by clock jitter and other non-ideal effects.

4.4 Pulse Control

This section describes the controllable parameters of the modulated pulses. The implemented control modules are three-fold: pulse width, amplitude, and resolution control. These are explained in detail in this section of the report.

4.4.1 Pulse Width

In order to control the width of the pulse we first need to define what we mean by pulse width. The common envelope used for the RF pulse is a Gaussian. Theoretically, the width of the Gaussian is of course infinite so it is necessary to restrict it in order to realistically create and store its values in the waveform memory. It was decided that limiting it to a minimum of three standard deviations ($\pm 3\sigma$) would be sufficient, as this would include 99.7 % of the Gaussian while still keeping it reasonably small.

The designed system can output waveform sample from the RAM once each clock cycle at the maximum, and as a result the minimum pulse width is determined by the clock period, t_{clk}, and the number of samples, N, stored in the waveform memory. Thus, the minimum pulse width $t_{p,\,\text{min}}$ can be determined by

$$t_{p,\,\text{min}} = t_{\text{clk}} \cdot N \tag{4.3}$$

A 409.6 MHz clock was used for the Zynq board design, which corresponds to a 2.44 ns clock period. Here we find a severe limiting factor to the system. If the 20 ns minimum pulse width specification should be fulfilled, this would imply a waveform memory size of only eight samples. This would later be an issue for the bit resolution control of the pulses, ranging from 16 down to 1 bits. With 16 bits we would expect $2^{16} = 65,536$ discrete levels in the pulse. Assuming a full Gaussian pulse is stored in the memory, a sample size of only eight samples could be viewed as a non-uniformly quantized signal with a maximum resolution of only 2 bits, and as a result, the extra 14 bits are not utilized effectively. This would lead to a very limited control of the resolution later on. However, to get full use of the maximum resolution, a memory size of $N \geq 2 \cdot 2^{16} = 2^{17}$ would be needed in order to represent the two sides of the pulse and get full use of the resolution. This memory size would result in $t_{p,\,\text{min}} \approx 0.32$ ms, which is far too wide to yield any sort of useful qubit fidelity. Thus, a compromise was needed.

It was decided that the minimum pulse width should be 1.0 µs as that is the maximum pulse width allowed by the project requirements shown in Section 1.3. Substituting this value of t_p in Eq. (2.6), together with the measured relaxation and decoherence times $T_1 = 79$ µs and $T_2 = 33$ µs, yields a theoretically possible fidelity of 98.79 %. From the product specification this would be the maximum allowed pulse width, but our design is fundamentally limited by the clock period so this compromise was unfortunately necessary. Given the minimum pulse width of the system of 1.0 µs and the clock period $t_{\text{clk}} = 2.44$ ns, the number of waveform samples are by Eq. (4.3) roughly 409.

The width control is fairly simple in its design. Each clock cycle one sample is output from an address in the waveform memory and the index is subsequently incremented

to fetch the next sample. By not changing the memory address for k clock cycles ($k \in \mathbb{Z}$) , it is possible to increase the pulse width by a factor k by simply clocking out each consecutive sample k times. This factor k is simply the desired pulse width set by the user. However, since k is the number of clock cycles per sample, we are limited to only changing the pulse width to a value which is evenly divisible by the minimum pulse width of 1.0 µs. In other words, we have a time resolution of 1.0 µs. This is of course dependent on how many samples are stored within the waveform memory. However, as previously mentioned, more samples improves the resolution of the envelope at the cost of longer $t_{p,\,\mathrm{min}}$ and worse F.

To control the indexing of the waveform memory at run-time, a VHDL module was implemented. A simplified block diagram of the design is shown in Fig. 4.4. The host computer sends the desired pulse width to the FPGA followed by a start signal, indicating that the FPGA should start transmitting the pulses. The `counter` block will increment each clock cycle until the value of the pulse width is reached. A trig signal indicates the end of the count duration and is sent as an input to the index counter block `inc_idx`. This block outputs the 10-bit index of the data sample that should be accessed from the waveform memory. When a rising-edge of the trig signal is registered, the current index is incremented by one and the next address from the memory is selected as the output signal. When the entire stored waveform has been outputted a done-flag is pulled high which resets the index counter. This flag is also used to keep track of the number of pulses transmitted.

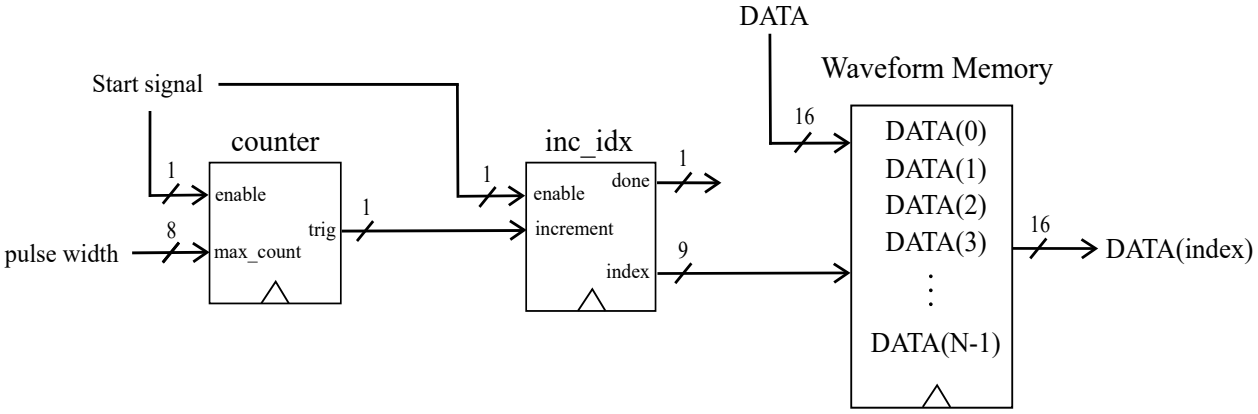


Figure 4.4: Block diagram of the pulse width control module. The clock and reset signal inputs to the registers have been omitted for simplicity.

It should be noted that the trig signal generated by the counter has a period of $2f_{\mathrm{clk}}$, and therefore the minimum achievable pulse width is 2 µs. If 1 µs pulse width is desired, the width control module will be circumvented and the RAM will be indexed as normal using the clock directly.

4.4.2 Amplitude

For the perfect AWG the amplitude control needs to be able to select at least 512 discrete levels for the amplitude, where 512 indicates 100 % of the maximum achievable amplitude from the DAC. We therefore have a percentage resolution of roughly 0.20 %.

Since the waveform memory stores a full resolution and full-scale envelope, an intuitive approach to lowering the amplitude would be to scale it by some factor, η, defined by

$$\eta \cdot (\text{Input data}) = \text{Output data} \tag{4.4}$$

Given that 512 represents the full-scale waveform, we get the expression

$$\eta = \frac{\text{steps}}{512}, \ \text{steps} \in \mathbb{Z} \cap [0, 511] \tag{4.5}$$

Thus, to get the desired output level we simply multiply the full-scale input with η. However, as the observant reader undoubtedly has noticed, η will be a fractional number. Hardware is implemented in binary logic, and typically binary numbers are regarded as integers. However, fractional numbers can be implemented in hardware, but it does requires some knowledge from the designer. The amplitude control implemented uses *fixed-point* representation for performing the fractional multiplication. As the reader might be unfamiliar with this binary representation of fractional numbers, a short explanation is in order.

The decimal value v of an n-bit fixed-point binary number $(a_{n-1}, a_{n-2}, ..., a_1, a_0)$ can be defined as [13]:

$$v = 2^{n-1} \sum_{i=0}^{n-1} a_i 2^{i-n} \tag{4.6}$$

A fixed-point number is commonly denoted in the format "IB.FB", where IB is the number of integer bits and FB the number of fractional bits. As an example, if we have a 4-bit number where we want one integer bit and three fractional bits, the format would be 1.3, or s1.3 if signed representation is needed. For normal integer representation 4 bits gives us a minimum value of 0 and a maximum value of $2^4 - 1 = 15$. However, with fixed-point the smallest fraction achievable given 1.3 format is

$$2^{-4} = 1/16 = 0.0625$$

and the maximum value is given by

$$1 - 2^{-4} = 15/16 = 0.9375.$$

Herein lies the explanation for the name "fixed-point". Each consecutive fractional value is of an equidistant step size, regardless of what value the current number has. This is in contrast to *floating-point* representation, where the step resolution scales exponentially, resulting in improved resolution for lower values and coarser steps for higher values. Observe that the maximum value in the 1.3 number above is strictly less than one. Thus, we can think of the representation as an integer, determined

by the n bits, being divided by 2^n. Recalling (4.5), this is exactly what we want to implement in order to define η in hardware. For 512 levels we need 9 bits, so if we define η to be of format s1.9, where the integer bit is the signed bit, we get the desired result. Since we want the scaling factor to always be positive, a '0' is always stored in the MSB.

The desired level is sent as a 9-bit input to the amplitude control module to be multiplied with the waveform. When multiplying fixed-point numbers a new format is formed, where the integer and fractional bits of the created product simply is the sum of the two formats. Since we can write the waveform memory in the format s16.0 (i.e. a 16-bit integer), we get the created product format:

$$\text{s16.0} + \text{s1.9} = \text{s17.9}$$

Thus, the product will be a 26-bit number. The number is then rounded to an integer by discarding the fractional bits, and setting the LSB of the integer bits to be the most significant bit (MSB) of the discarded fractional bits so that the output is in the correct 16-bit format.

The whole module is implemented as a pipelined design with three registers: a 16-bit input register, a 26-bit product register, and a 16-bit output register. Its block diagram is shown in Fig. 4.5. It receives one sample from the waveform memory each strike of the clock and due to the amount of registers it takes three clock cycles before the computed product is available at the output of the module. Even though it was not a requirement of the system, it is possible for the user to change the amplitude step value at any time during run-time, allowing for extremely flexible amplitude control in real-time. Note that the "steps" input is 9 bits, while the register output is 10 bits. This is the result of the '0' getting concatenated to the MSB of the vector to create the s1.9 format.

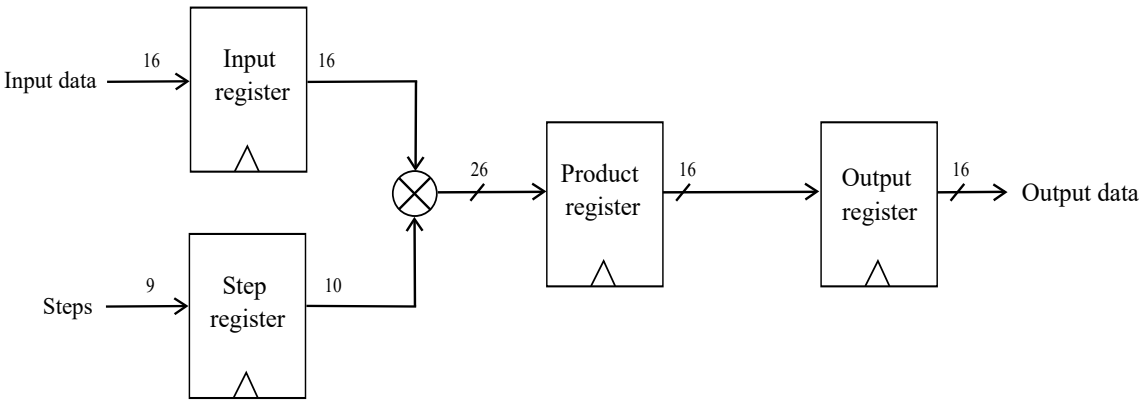


Figure 4.5: Block diagram of the amplitude control module. The clock and reset signal inputs to the registers have been omitted for simplicity.

4.4.3 Resolution

For the resolution control we are limited by the fact that the DAC input on the Zynq board is fixed to 16-bit resolution. The two LSBs are then dropped to form a 14-bit resolution output signal. Therefore, to control the resolution we need to emulate the degradation of the signal while still writing to all 16 bits. A way to achieve this is to simply "dump" the LSBs of the data before sending it to the DAC. To elaborate, if we desire N-bit resolution we will have to write zeros to the $16 - N$ LSBs. However, since our signal path is strictly in signed representation we cannot dump all bits below the MSB, since that will not create the desired result. Instead we are limited to dumping the $15 - N$ LSBs.

In digital hardware, N-bit resolution means there are 2^N discrete levels the signal can be quantized to. As an example, a resolution of 3 bits should give 8 discrete levels; determined by the value of the binary word $b_2b_1b_0$. Implementing this in the 15-bit format as mentioned above would gives the 15-bit number: $b_2b_1b_0000000000000$. This ensures that we still have 8 possible levels. However, rather than the value of these levels being in the integer range 0–7 they are now instead scaled to the available levels in the 15 bit representation. Given the "3-bit" resolution, these levels are in the range of $0–2^{14}$, with steps of 2^{12}. Concatenating the signed bit to the MSB of this vector then creates the correct representation.

The control implements this functionality be receiving the desired resolution as an 8-bit input set by the user, with allowed values ranging from 1-bit to 15-bit resolution. The module then uses this value to index the LUT presented in Table 4.1.

Table 4.1: Bitmask LUT

Index	Binary value
0	100000000000000
1	110000000000000
2	111000000000000
3	111100000000000
...	...
15	111111111111111

The input data's 15 LSBs is logical AND-bitmasked with this selected vector from the LUT, and thus creating a new 16-bit output with the undesired bits set to '0'.

4.4.4 Pulse Delay

In order to control the time between transmitted pulses, a pulse delay module was also implemented. The module consists of a simple state machine and counter. Every time the module detects that all samples of the envelope waveform have been transmitted, it sets the module's reset port to '1' and activates an internal counter. Thus, the time between pulses, τ, is determined by the input value of the module and the clock frequency, as shown in Eq. (4.7). Once the counter reaches the threshold value, it de-asserts the reset port, allowing the next envelope to be transmitted.

$$\tau = \frac{1}{f_{\text{clk}}} \cdot (\text{input value}) \tag{4.7}$$

4.5 Communication

This section describes how the user can communicate with the FPGA and control the pulse programs. Section 4.5.1 provides a condensed explanation of how the user interface works and Section 4.5.2 goes into detail how the FPGA receives and handles incomming data from the UART bus.

4.5.1 Host Computer

As previously mentioned, the pulse program and parameters are set from a host computer. The user interface created is simply a MATLAB-script that allows the user to input desired parameters of the pulses, enabling sweeping of parameters, and controlling the number of pulses and the stop time between pulse programs. It can also define and load the desired envelope to the waveform memory. The parameters are then sent via USB using the UART communication protocol to the FPGA, and the pulse program values are also stored in a text file after each run to ensure that the user is aware of what exact program was run. The interface was left relatively basic to allow simple testing of the full system. However, it is very flexible to changes and additions as long as the user is aware of how the receiver module handles data, which we will discuss in the following section.

4.5.2 Receiver

Sending, receiving, and storing data within the FPGA is a relatively simple task to implement in VHDL and was achieved early on in this project. However, interpreting this data correctly quickly increases the complexity – especially when there are so many different parameters and control values as in this project. To handle this decoding of incoming UART messages, a receiver module was implemented.

The main part of the receiver module (other than the UART module) was a Moore finite-state machine (FSM) used for decoding and storing the received values to the correct registers. A Moore FSM is a synchronous process that controls outputs depending on the current state of the FSM. UART transmits eight bits of data, or one byte, with one bit at a time. When a full 8-bit word is ready to be read, an

enable signal is pulled high. This enable signal is, other than the clock, the main controller of the FSM.

The FSM expects the user to always send a command value first, followed by the parameter data. What this means is that each parameter controlling the pulses that can be transmitted from the host is associated with an 8-bit word. The FSM is in an idle state until a command value is ready to be read from the UART bus. It then enters a state for reading the command value, and selects the next state depending on what has been received. The rest of the FSM is then simply a manner of waiting for the following parameter data to be received and stored in the correct registers, before returning to the idle state and wait for the next command value. To ensure that this does not induce any synchronization issues, all registers are updated synchronously in the same process regardless on how many parameters have been changed.

The main benefit of this design is that once the waveform has been loaded, which also is the most time-consuming operation, the control parameters may change freely during run-time, with very little risk for synchronization issues. Thus, there should be no issues sweeping parameter values as specified by the product specification.

Additionally, the receiver module has three outputs that act as trigger signals used for indicating when a new waveform is being loaded, as this is the most time-consuming task. It also ensures that the pulse program is not running without a stored waveform or while a new waveform is being loaded. For the reader interested in a more detailed explanation of the FSM we refer them to the documentation report of this project.

5

Results

In this chapter the results from the testing of the implemented system is presented. In Section 5.1 we present and discuss the results from synthesis and verification of the individual modules that form the entire system. In Section 5.2, we present the measurements and control tests from the lab after the full design had been loaded onto the Zynq board.

5.1 Verification and Synthesis

This section presents the results from functional and timing simulations after synthesis and implementation of each of the main modules in the system.

5.1.1 NCO Performance

The functional simulation results from the previously introduced NCO topology are shown in Fig. 5.1, it is observable that the number of samples in a period decreases for higher output frequencies, which is the main limitation in periodic signal generation because the number of samples in a period is directly related to the clock frequency. To increase the frequency of the desired waveform, a reduced amount of data is fitted to a smaller time window.

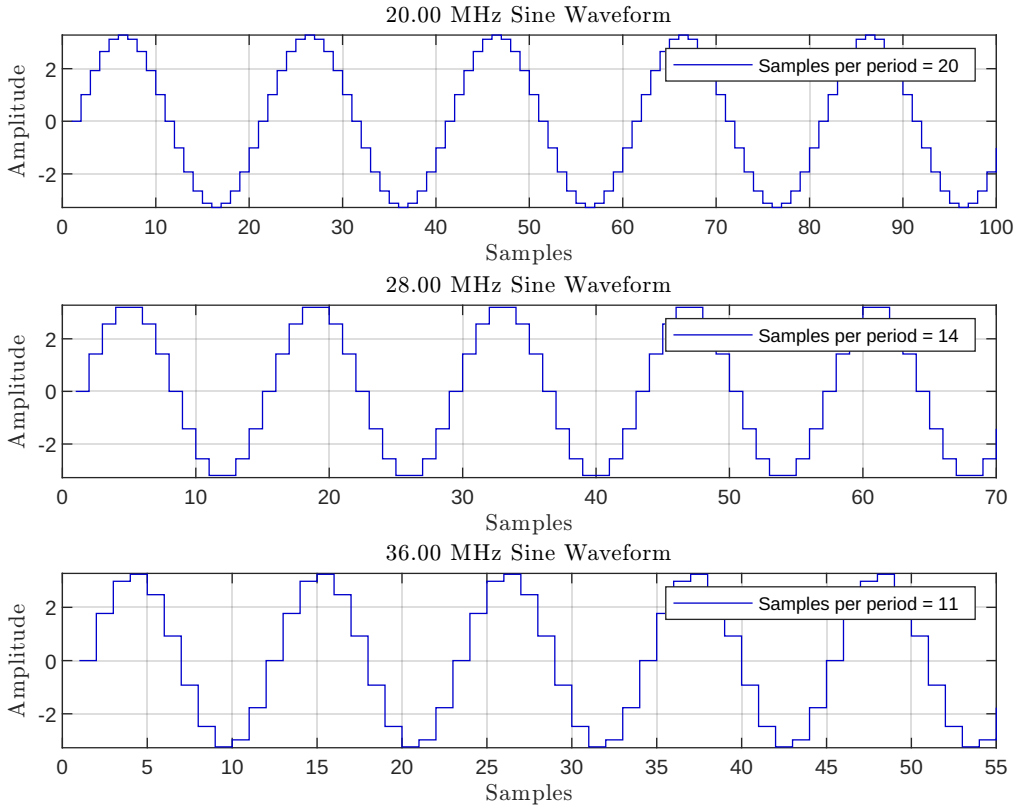


Figure 5.1: NCO functional simulation results for different output frequencies.

5.1.2 Pulse Width Control Simulation

The pulse width control module was synthesized and implemented in Vivado with the Zynq board as the target hardware. The module was tested for clock frequencies between 300–400 MHz, and it passed the timing constraints for both. Once implemented, verification of functionality was done by running a functional simulation using a testbench on the implemented design. The testbench provided stimuli in the form of Gaussian pulse of $\pm 3\sigma$ width consisting of 300 samples together with four different pulse width values ranging from 2–5 µs. The output of the resulting signal was then plotted in MATLAB and is shown in Fig. 5.2. Note that since the design initially had a 14-bit signal path, the Gaussian in Fig. 5.2 was defined with that resolution in mind.

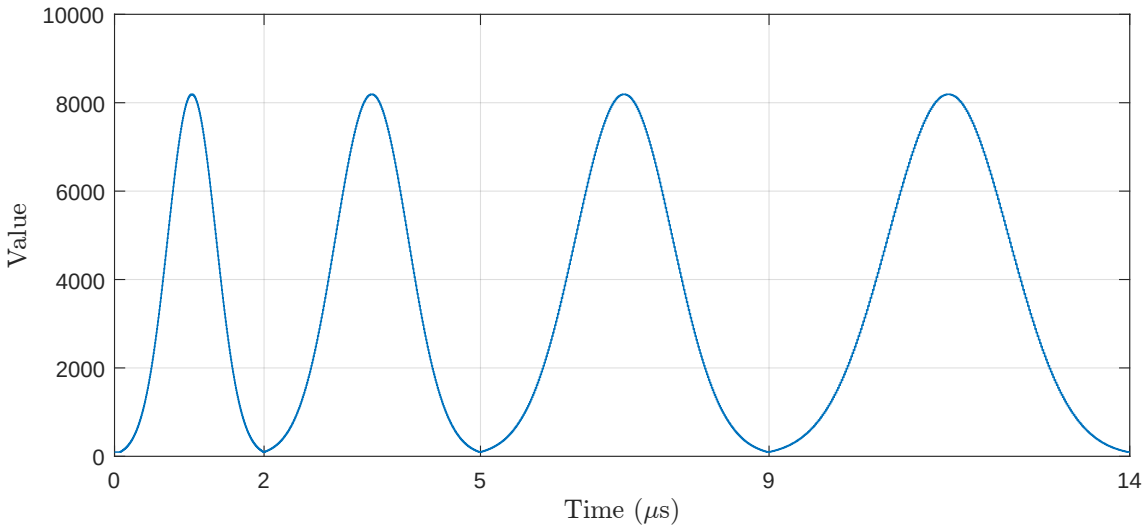


Figure 5.2: Results from post-implementation functional simulation of the pulse width control module. Pulse widths tested were 2–5 µs.

From the figure it can be easily verified by a simple ocular inspection that the pulse widths of each of the four pulses correspond well with the expected values.

5.1.3 Amplitude Control Simulation

Similar to the pulse width control, the amplitude control module was synthesized and implemented for the Zynq board with a clock rate of 300 MHz. Functional verification was performed using a testbench that provided the Gaussian pulse as stimuli together with a random amplitude step test vector for the controller. The fixed-point multiplication and truncation will generate some error (unless it would be possible to use an infinite amount of bits). The relative error, e_r, of some number x can be defined as [13]

$$e_r = \frac{|x_{\text{actual}} - x_{\text{expected}}|}{x_{\text{expected}}} \tag{5.1}$$

In Fig. 5.3 we see the results from this simulation. The input data is plotted together with the resulting output signal for different amplitude step values. Additionally,

a moving average of the error e_r of each output value was plotted as a function of amplitude step.

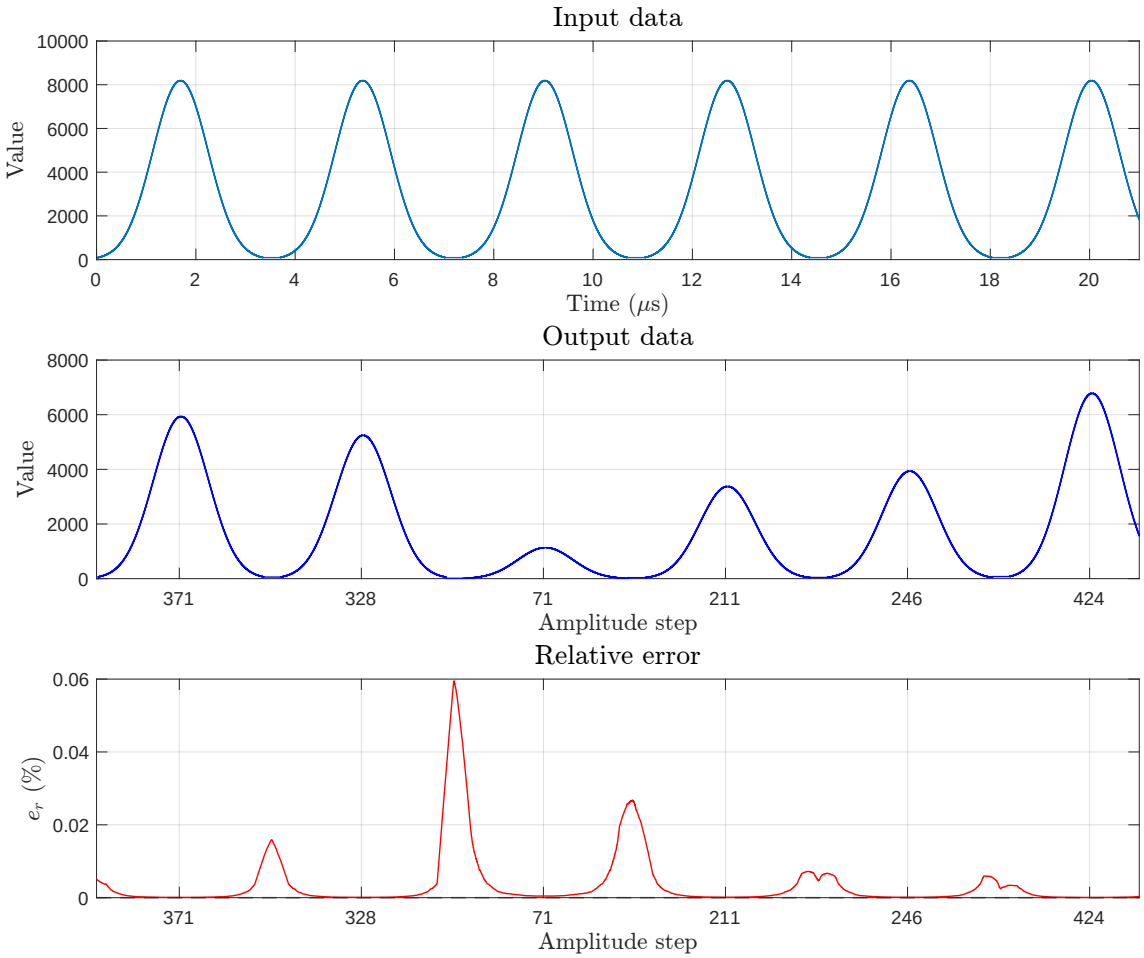


Figure 5.3: Results from post-implementation functional simulation of the amplitude control module, with random amplitude steps used as a test vector.

Since the fixed-point multiplication more often then not has some error, it was difficult to automate the verification process. Instead some output values were measured and verified using Eq. (4.4). Recall from Section 4.4.2, the output is delayed three clock cycles so comparison of the resulting output after multiplication of the input signal with the step value needs to bear this in mind.

From Fig. 5.3, we can see that the amplitude control definitely is scaling the input signal. Of course, we also see some relative error throughout the process, with a peak at 0.06 %. The error is maximized when multiplication is being performed on the low values of the Gaussian. This was expected as the truncation of the 26-bit product will impact the result much more for low input values. For example if two consecutive samples of input data are 1 and 5, and the step value is 100, the expected

result would according to (4.4) be

$$\frac{1}{512} \cdot 100 = 0.1953125$$
$$\frac{5}{512} \cdot 100 = 0.9765625$$

After truncation they will have the decimal value of 1 or 0. In other words, they can be interpreted as the same number even though one should be five times larger than the other. However, a maximum relative error of only 0.06 % of the expected value was deemed acceptable since the specification only requires that it should be possible to change the amplitude in 512 discrete steps, not that each value is perfectly accurate.

5.1.4 Resolution Control Simulation

The resolution control was synthesized for the Zynq board with a clock frequency of 300 MHz. It was tested by successively dropping the bits from full bit-resolution down to 1 bit-resolution. At the point of testing the data path in the design was 14 bits, however this would later change when the design was migrated to the Zynq board in order to match the 16-bit data path of the DUCs. As can be seen in Fig. 5.4, the intended functionality was achieved. It should however be noted that degrading the resolution by discarding LSBs results in a loss of amplitude, hence power, for lower resolutions.

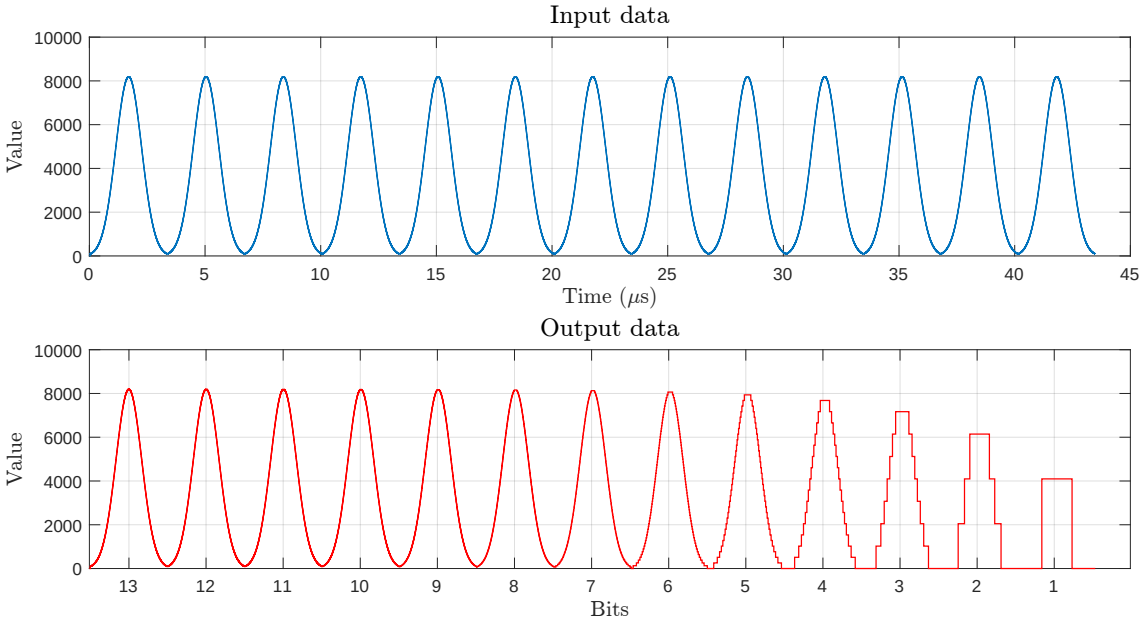


Figure 5.4: Results from post-implementation functional simulation of the resolution control.

5.2 Full System Measurements

The full system was tested in a lab environment by sweeping control variables for the different modules. This was achieved using the setup in Fig. 5.5. Using MATLAB-scripts, individual parameters were stepped through a set of values which were transmitted via USB to the Zynq board. The outputs from the I- and Q-channels were then captured in two separate channels of an oscilloscope, which in turn was connected by Ethernet to a network card. To achieve reasonable data acquisition time during sweeping of parameters, the number of samples were limited to 10,000. Another computer was connected to the network card in order to read the data from the oscilloscope, again, using MATLAB-scripts. This process was fully automated. That is, after the desired parameter and sweep range were set and the test program started from the transmitting computer, the experiment would run until output data for all the parameter values in the range had been obtained.

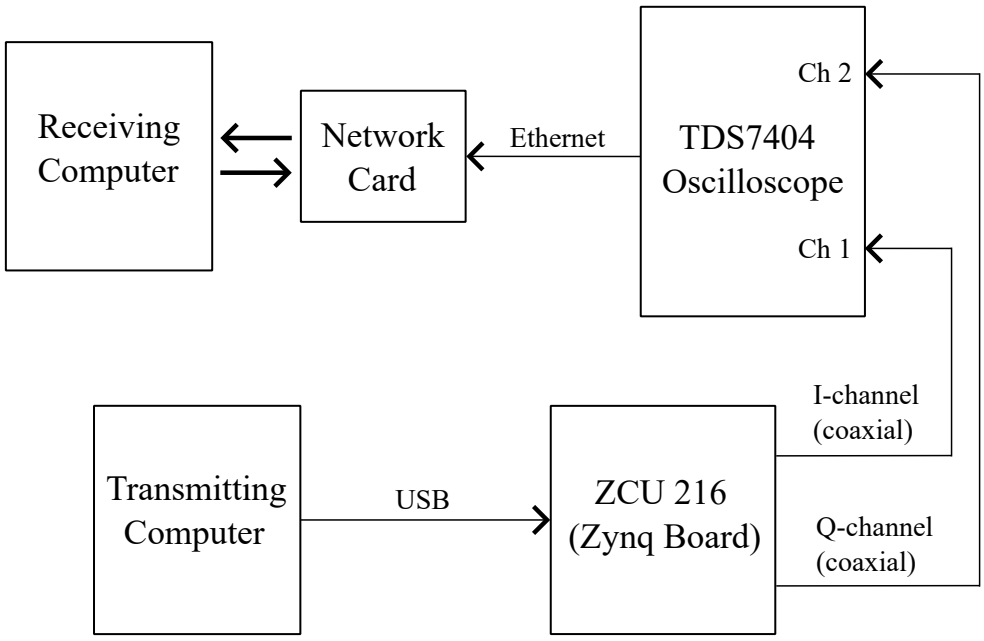


Figure 5.5: Measurement setup in the lab for sweeping parameters.

The envelope stored in the waveform memory was a Gaussian consisting of 409 samples as before, however with the slight change of having the pulse width defined from $\pm 4.4\sigma$ instead of $\pm 3\sigma$. This was to observe a smoother change closer to 0. Also note that the receiver module was not included during these measurements to limit the possible unforeseen errors of the design. Instead a simplified receiver was used, only capable of changing one parameter at a time determined before synthesis. The system was therefore re-synthesized before each control test. The receiver was however successfully tested later in the project while using a fixed waveform memory and changing all of the parameters during run-time from the host, although ultimately more testing is required.

5.2.1 Intermediate Frequency (IF) Measurements

To check to correctness of the IF data the mixers in the RF DAC were initially bypassed. Fig. 5.6 shows the IF output of a full-scale pulse with a pulse width of 2 µs generated at an IF carrier-frequency of 20 MHz. The resolution was set to the full resolution of the DAC.

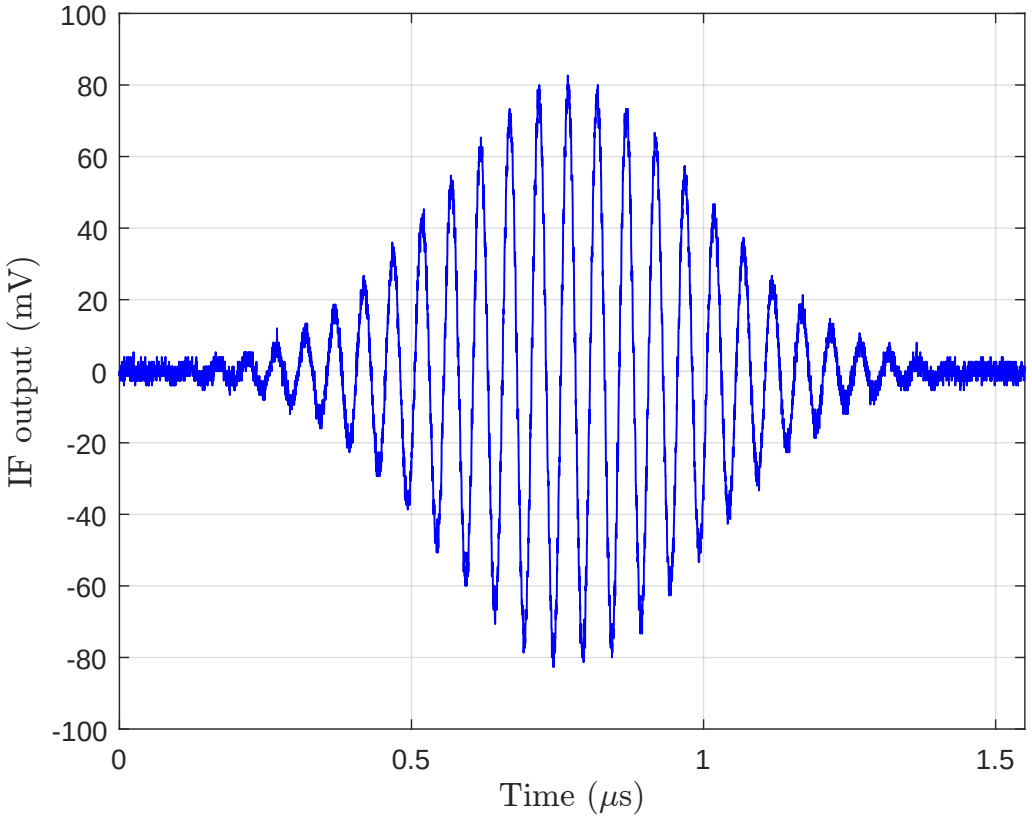


Figure 5.6: IF IQ modulator output. The RF mixer in the DAC was bypassed during these measurements.

From the figure we can see the expected modulated pulse. However, note that even though it looks like the pulse width is only about 1.5 µs the full pulse was actually correctly measured as 2 µs. The pulse data was captured from the oscilloscope in a way that did not include the entire pulse, so it was fitted symmetrically for the plot in Fig. 5.6. A maximum amplitude of around 83 mV was also measured and will be used as a reference from here on out.

To verify that the correct sideband had been generated, the power spectrum of the signal was plotted. Fig. 5.7 displays the resulting power spectrum after performing an FFT of the above signal. The spectrum was also normalized in relation to the fundamental tone. As can be seen from the spectrum, the fundamental tone is located at the carrier frequency of 20 MHz, as expected. Additionally, good noise suppression was achieved with the noise floor being centered at about −60 dB. Also seen is the presence of some harmonics, most notably the one just shy of 130 MHz at a power of about −30 dB. It should be noted that no analog filter was used for the

DAC output, which of course would have filtered out some of these non-linearities and noise.

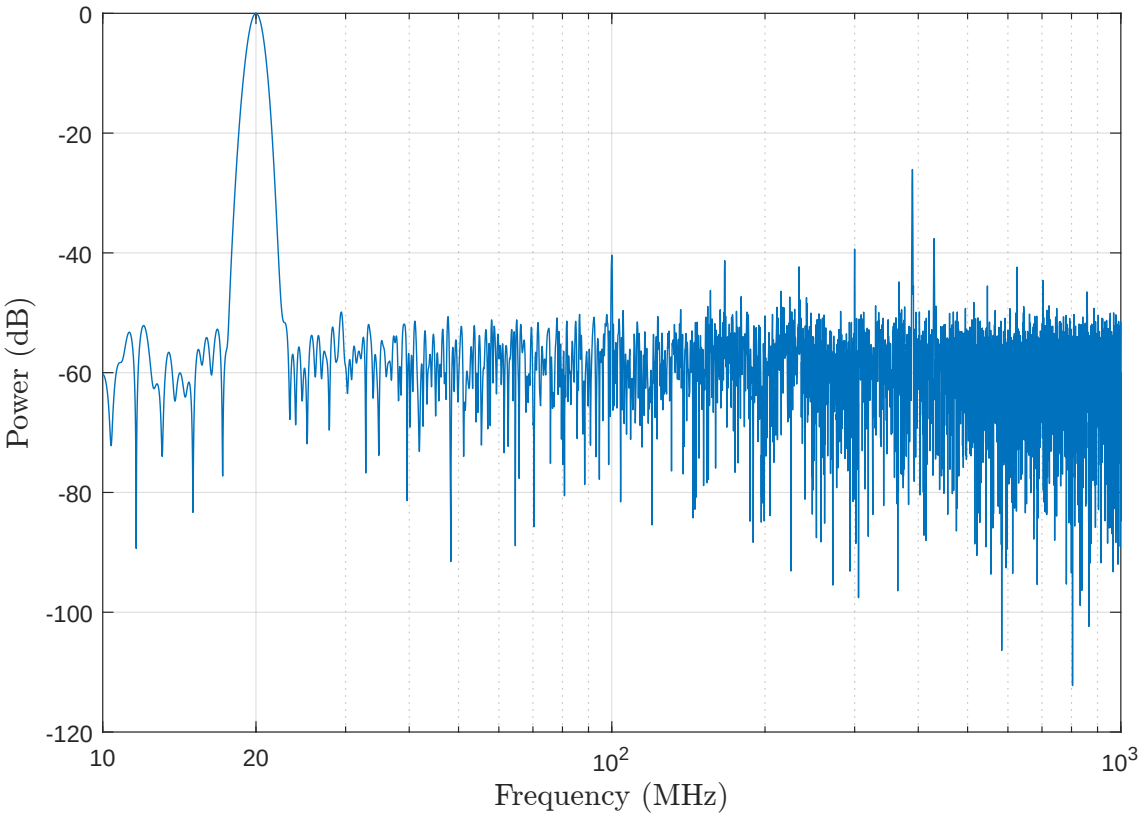


Figure 5.7: Power spectrum of the IF output signal. The 20 MHz IF carrier is clearly visible in the plot.

The first parameter to be swept was the resolution. As described in the theory section, this is done by dumping the LSBs. Fig. 5.8 shows the resulting IF modulator outputs for 2-, 3- and 4-bit resolution.

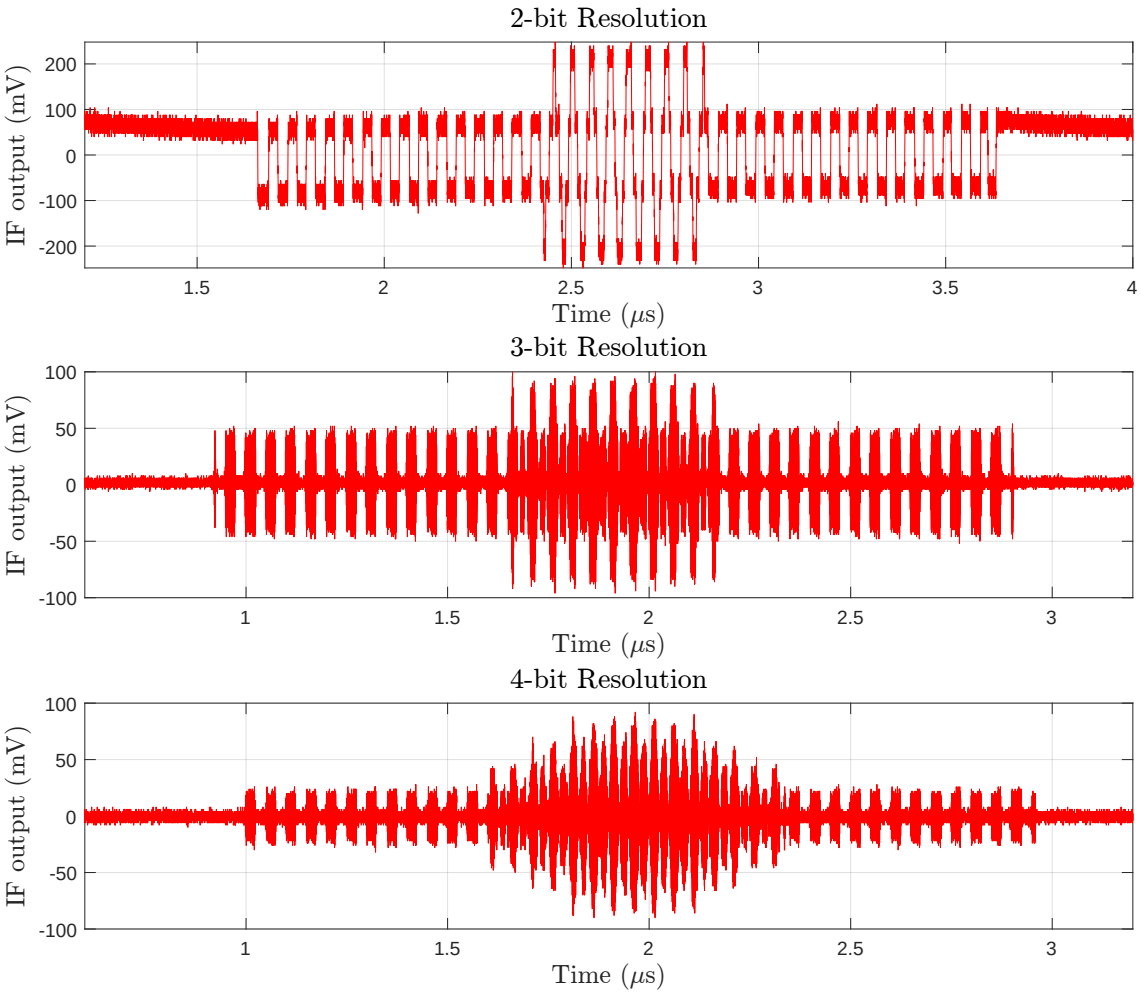


Figure 5.8: Results from resolution control test of the IF modulator. The plots show the measured IF IQ output with different bit resolution.

The top figure shows best that the resolution control was producing the expected output. It clearly shows four distinct voltage levels, as expected by the 2-bit resolution. However, the signal does have a higher maximum amplitude than expected. This result is somewhat unclear and was discovered after the lab session. It could be that the scale of the oscilloscope was off during these measurements, but further testing is needed to be sure.

The reason for analyzing the resolution at IF is that after the IF signal has been mixed with the high-resolution NCOs in the DUC of the DAC, we have no way of manipulating the signal quality. Thus we provide these images as a reference. In order to degrade the RF pulse as desired by the product specification it would be required to place the resolution control module after the HF mixer, which unfortunately was not possible from what we could tell from the RF DAC documentation.

5.2.2 High Frequency (HF) Measurements

In Fig. 5.9, the top-most image displays the outputs of the I- and Q-channels of the RF DAC. The IF NCOs where set to generate a 20 MHz negative offset and HF NCOs to generate a centre frequency of 2.048 GHz. The bottom image in Fig. 5.9 displays the resulting signal after summing the outputs from the I- and Q-channels in post-processing.

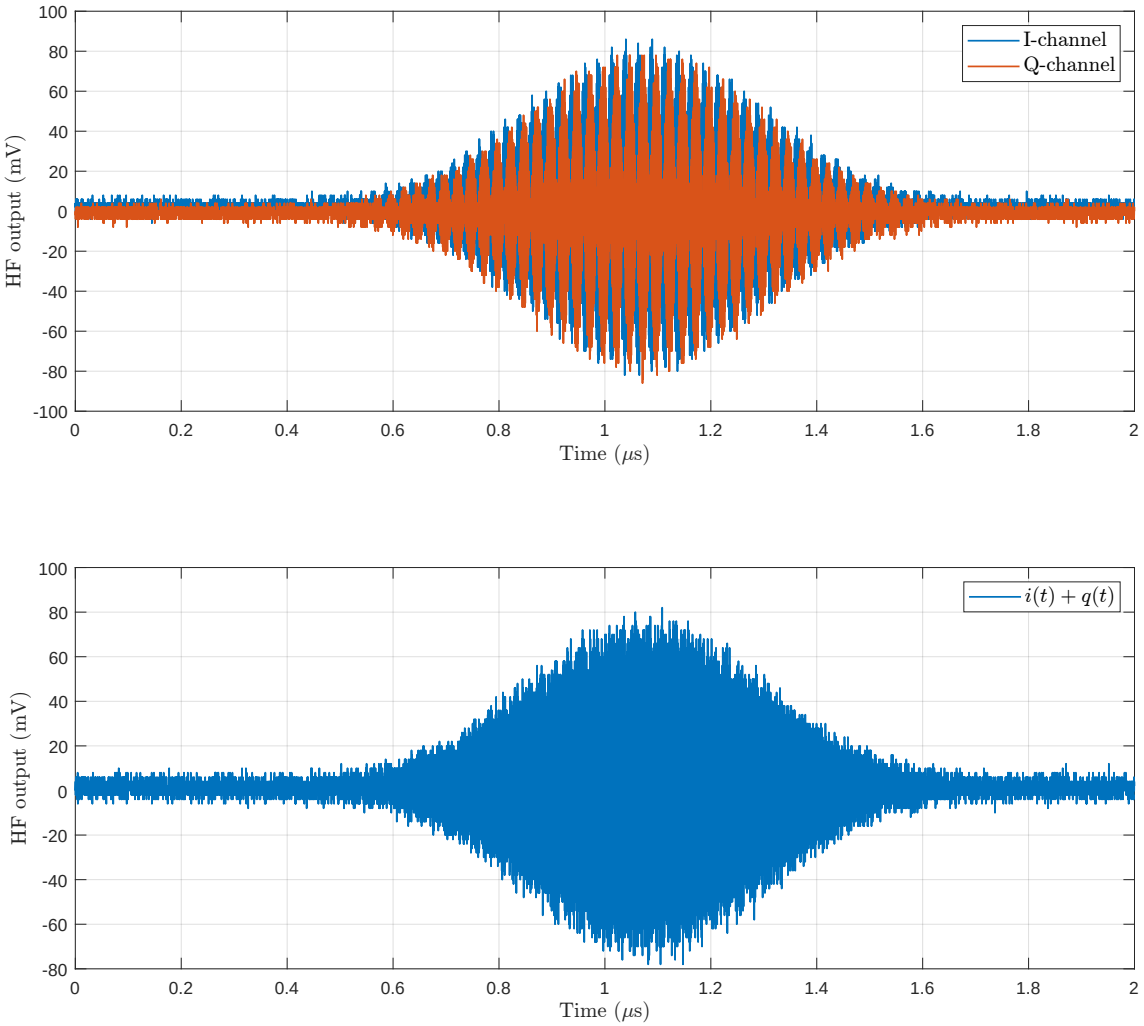


Figure 5.9: HF IQ output. The upper figure shows measurement of the I- and Q-channel of the DAC, and lower figure shows the summation of the signals done during post-processing in MATLAB. DAC was configured IQ to IQ during these measurements. The RF mixer on the DAC generated a 2.048 GHz carrier.

From Fig. 5.9 we can observe the expected 90° phase shift of the two channels in the upper plot, which was also confirmed during measurements in the lab. The bottom plot has the expected appearance of a HF IQ modulated pulse, although quite noisy. It should be noted that this waveform looks deceivingly degraded due to limited sampling frequency during data acquirement. During MATLAB simulations of the ideal IQ modulator, similar waveform degradation was observed when the

sample rate was reduced. It is therefore likely that the signal quality is far higher in reality than the plot suggests.

With the selected negative frequency offset, we expect a lower sideband to have been generated. To verify this, and that the carrier and upper sideband have been suppressed, the power spectrum of the signal was plotted as before and is shown in Fig. 5.10. From the specification the desired frequency range for sweeping is $f_{\mathrm{HF}} \pm 150\,\mathrm{MHz}$, we have therefore limited the power spectrum to this bandwidth in order to not include noise that later can be filtered out by a band-pass filter.

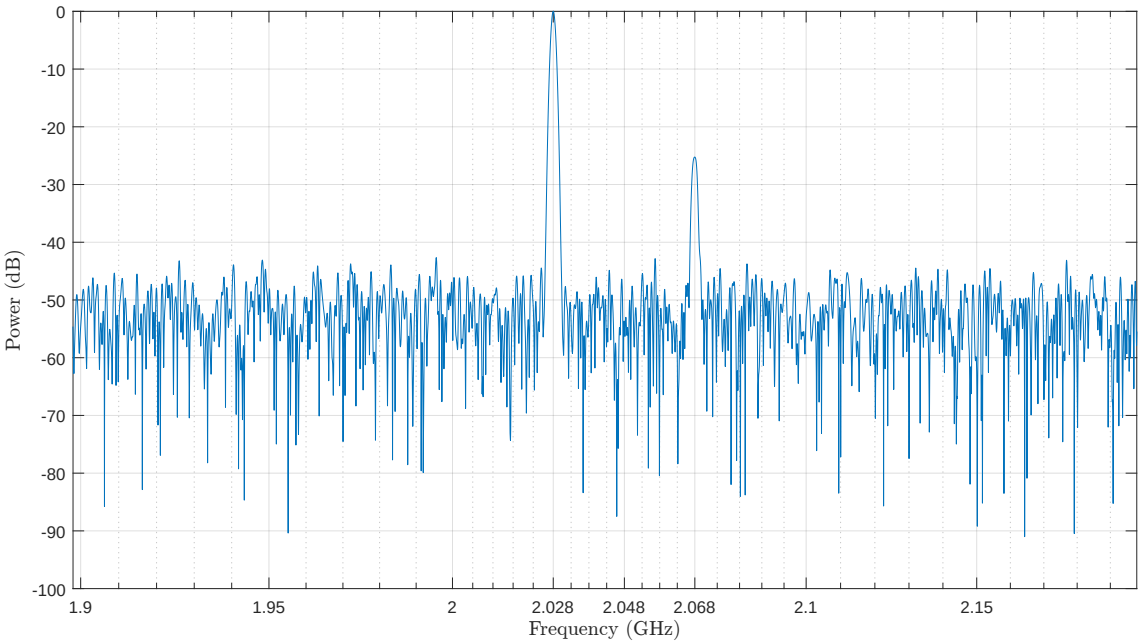


Figure 5.10: Power spectrum of the RF pulse. The spectrum is ideally "filtered" by omitting irrelevant frequencies outside of the bandwidth $B = f_{\mathrm{HF}} \pm 150\,\mathrm{MHz}$ with $f_{\mathrm{HF}} = 2.048\,\mathrm{GHz}$.

From the spectrum it is clear that very good carry suppression was achieved, with the carrier not being distinguishable from the noise floor. Furthermore, we see the lower sideband at around 2.08 GHz, exactly 20 MHz from the carrier as expected. However, we also see the upper sideband at the correct frequency, 2.068 GHz, not being fully suppressed. A lot of suppresion has been achieved however, with it having a power of $-25\,\mathrm{dB}$ relative to the lower sideband. This power could be reduced further by calibrating the IF modulator by sweeping the phase offset of just one of the NCOs.

The signal-to-noise-and-distortion ratio (SNDR) of the spectrum, including the upper sideband component as well, was calculated to be 22.78 dB. Omitting the sideband instead yielded an SNDR of 27.46 dB.

The pulse control was tested by providing some different pulse width values to the system and measuring the waveform on the oscilloscope. Fig. 5.11 shows the resulting output signal after a 10 µs was sent to the controller.

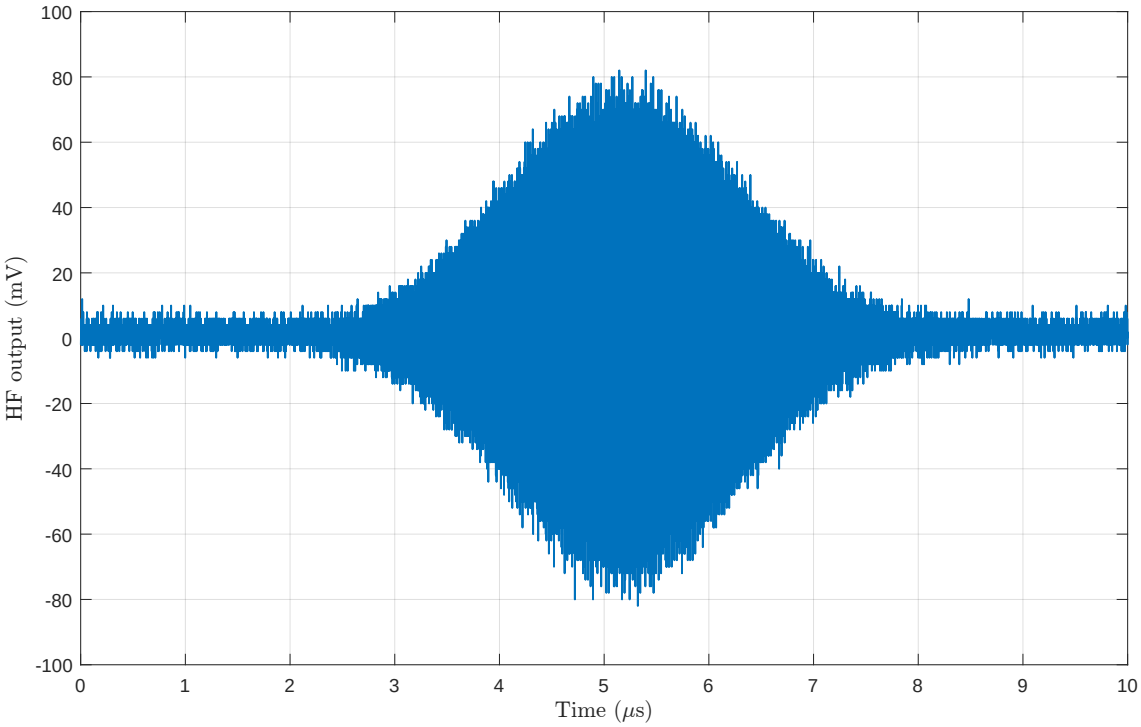


Figure 5.11: Result from pulse width control test. HF IQ output with $t_p = 10\,\mu s$.

The resolution was then swept in a similar fashion as explained in Section 5.2.1, but now with the HF mixer enabled to observe the effects of the resolution control on the full design. Fig. 5.12 shows the resulting outputs for 2-, 3-, and 4-bit resolution. Comparing the 2-bit resolution output with that of when only the IF modulator was used (Fig. 5.8), we can observe the same four voltage levels shaping the signal. However, this time the HF carrier is oscillating with full resolution within these levels. Fig. 5.13 shows a zoomed in version of the same plot where the HF carrier can be seen to clearly be in full resolution.

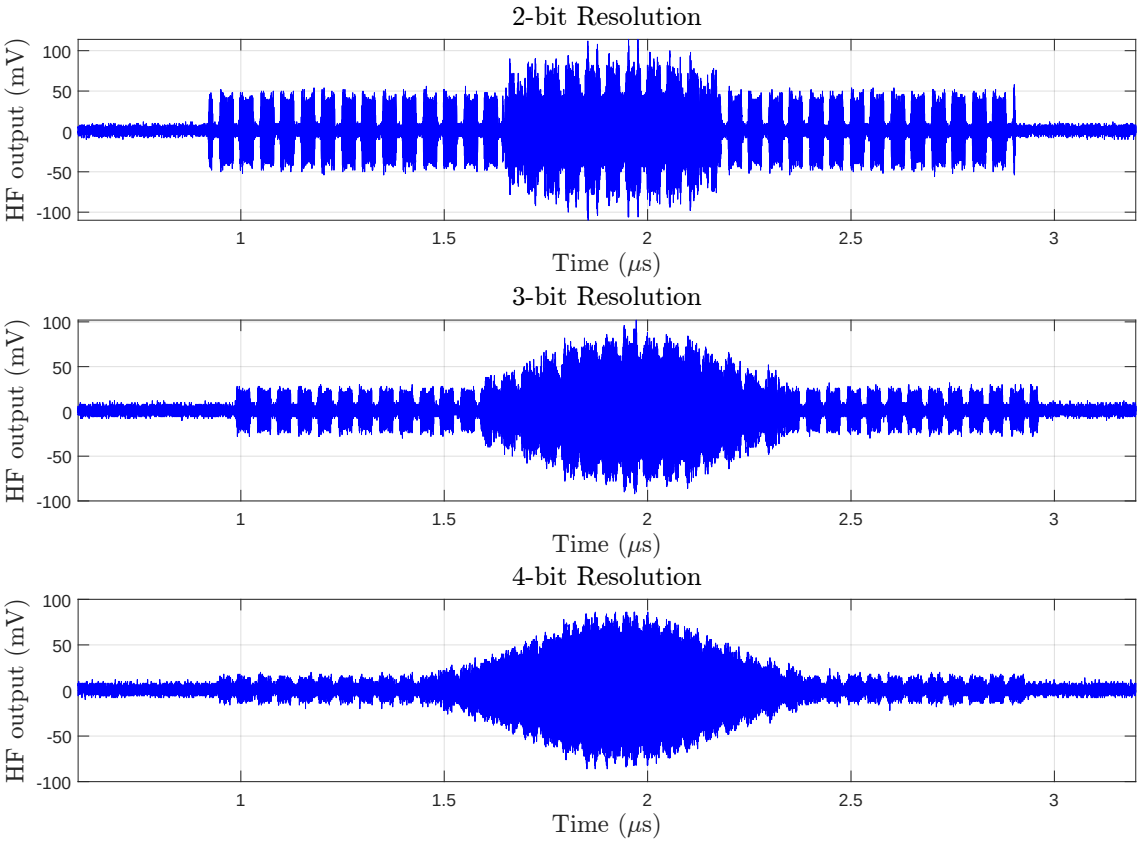


Figure 5.12: Results from resolution control test on the full system, including the HF mixer on the DAC. The plots show the measured HF IQ output with different bit resolution.

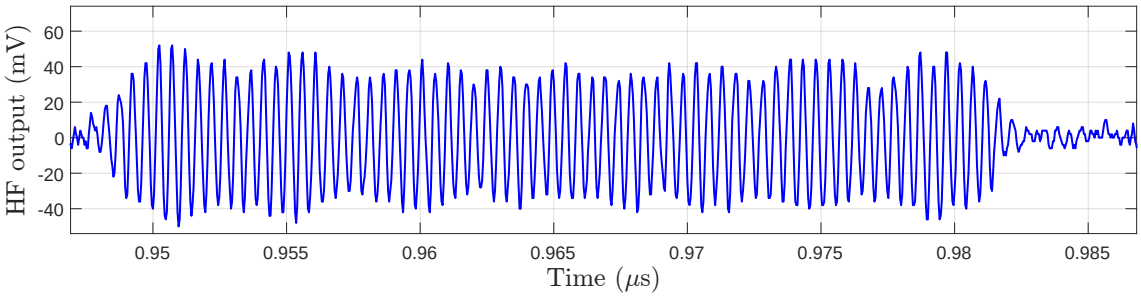


Figure 5.13: Zoomed in plot of the 2-bit resolution waveform. The HF carrier can be seen to still be in high resolution.

Fig. 5.14 shows the output signal after sweeping the amplitude step values for the full resolution signal. While the figure clearly shows that the amplitude can be controlled successfully, the step value of 255 in the bottom figure resulted in surprisingly high amplitude. With a step value of 255 we expect the output to be halved. As previously measured, the maximum achievable amplitude of the pulse was around 83 mV and the scaled version in Fig. 5.14 is showing similar amplitude, which was unexpected. There was unfortunately not enough time left on the project to thoroughly investigate this. It could be once again that the scale of the oscilloscope was off, although this needs to be confirmed.

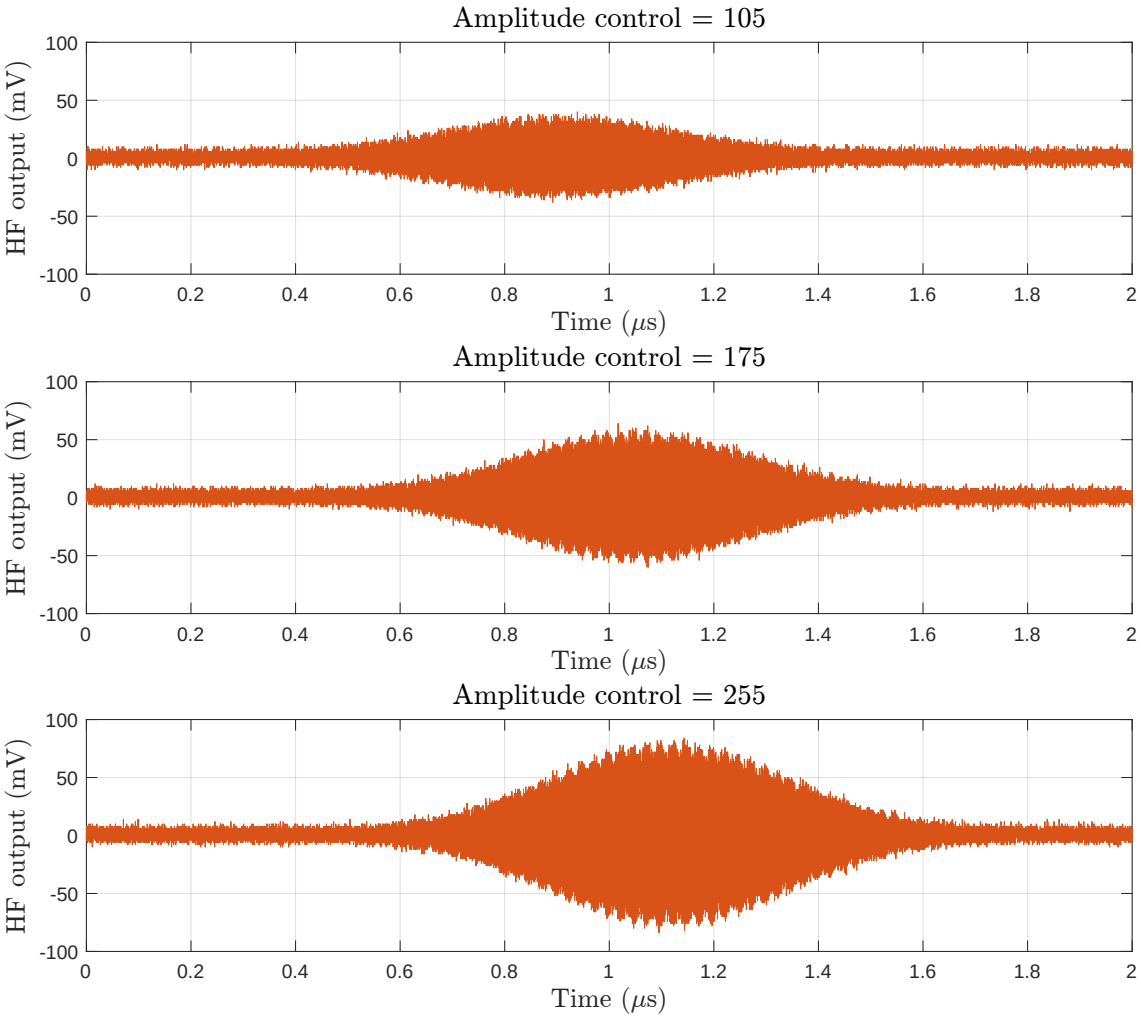


Figure 5.14: Results from amplitude control test on the full system. The plots show the measured HF IQ output with different amplitude control values.

The phase offset of the IF carrier was then swept and the I- and Q-channels were observed to see if they were shifting in phase in relation to each other as expected. Fig. 5.15 shows the resulting waveforms when the phase offset was set to 0 and 60 respectively.

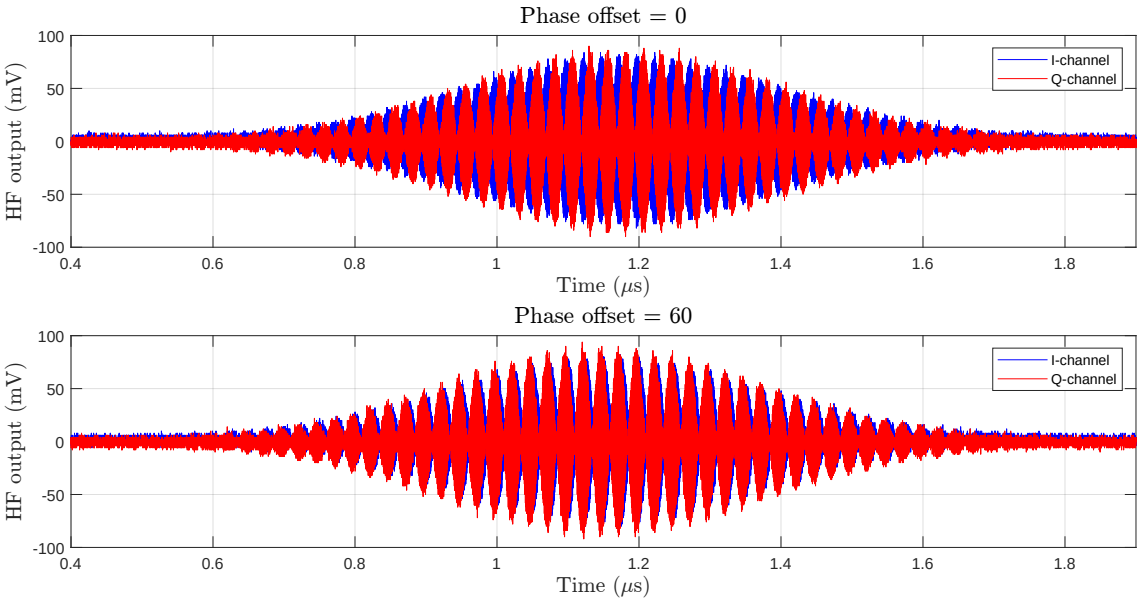


Figure 5.15: Results from phase control test on the full system. The plots show the measured HF IQ output with different phase offset.

With a phase offset of 0 we expect the I- and Q-channels to be 90° out of phase, which the upper plot in Fig. 5.15 verifies. For a phase offset of 60, we can rewrite Eq. (4.1) to calculate the expected phase $\varphi \approx 381° = 21°$. Measurement of the phase between two of the peaks in the 60 offset plot in Fig. 5.15 yielded:

$$\varphi = \frac{(1.25026 - 1.24726)\mu s}{1/20\,\mathrm{MHz}} \cdot 360° = 21.6°,$$

which is clearly very close to the expected value.

Next the frequency offset control was tested. The system was configured to first generate a lower sideband with an FSW of 61, which by Eq. (2.18) should yield a frequency offset of 24.4 MHz. In other words, a lower sideband located at

$$f_{\mathrm{lower}} = 2.048\,\mathrm{GHz} - 24.4\,\mathrm{MHz} = 2.0236\,\mathrm{GHz}.$$

In Fig. 5.16 the power spectrum of the measured output signal is shown.

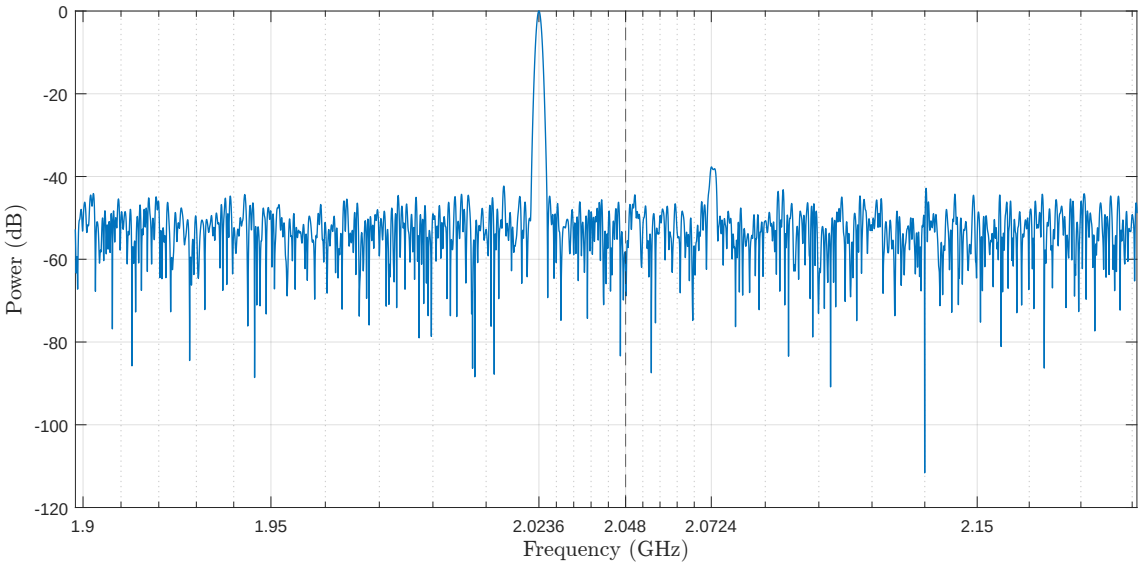


Figure 5.16: Results from frequency offset test on the full system. The plot shows the power spectrum of the RF pulse when frequency control was set to generate a lower sideband with an FSW of 61. The HF carrier is indicated by the dashed line.

From the spectrum we can observe that we indeed have a lower sideband at 2.035 GHz. Furthermore, the upper sideband at 2.0724 GHz has been well suppressed this time, with a power of only −37 dB in relation to the lower sideband power.

The experiment was repeated, but this time the system was set up to generate an upper sideband with an FSW of 66, which should generate a frequency offset of 26.4 MHz. The expected upper sideband will therefore be at 2.0744 GHz. The power spectrum of the output signal was once again plotted and can be seen in Fig. 5.17. From the plot it is clear that the correct upper sideband has been generated. However, the lower sideband was now only attenuated by −19.8 dB. The suppression of the lower sideband could possibly have been improved by sweeping one of the NCO phase offsets as mentioned in Section 5.2.1, however it was not calibrated between the measurements of Fig. 5.16 and 5.17.

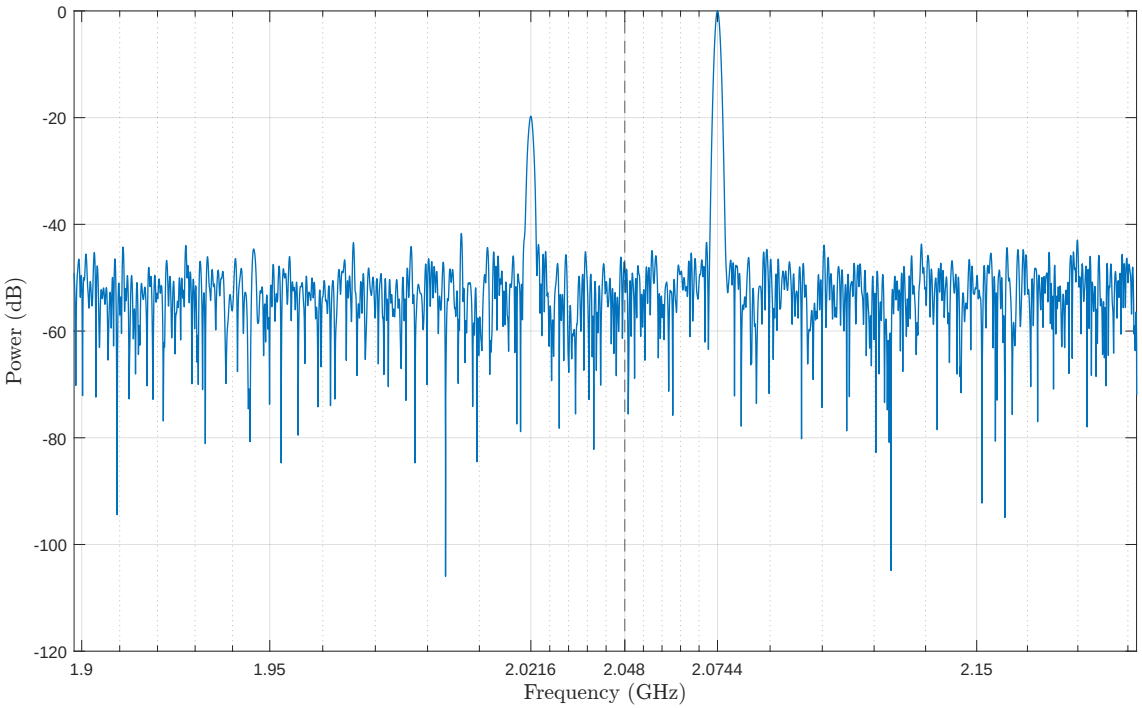


Figure 5.17: Results from frequency offset test on the full system. The plot shows the power spectrum of the RF pulse when frequency control was set to generate a upper sideband with a FSW of 66. The HF carrier is indicated by the dashed line.

Finally, the control for increasing the time between pulses was tested by sending some values to the modules. The result of two different control values are shown in Fig. 5.18. Since this control was not significant part of the product specification, not a lot of time was spent on verifying that the delay between pulses was correct. However, Fig. 5.18 does clearly show that the delay can be controlled by the host in some way, but it needs some further testing in order to produce predictable behavior.

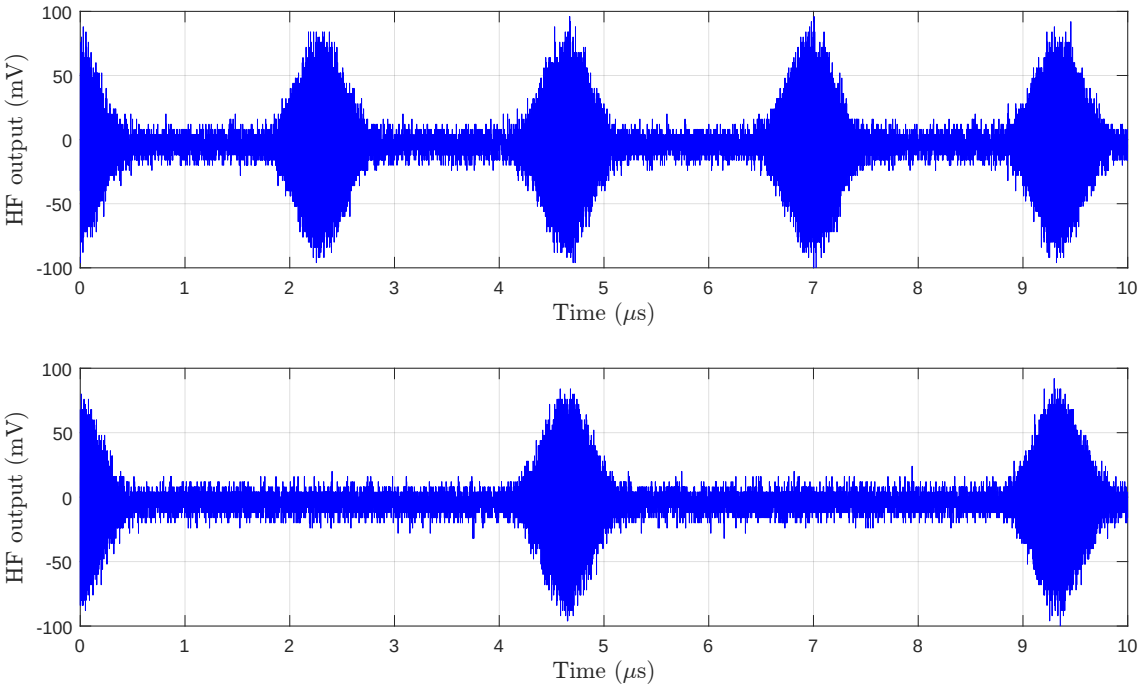


Figure 5.18: Results from testing different values for time between pulses.

6

Conclusion and Future Development

This report has presented an implementation of a novel AWG allowing generation of microwave pulses with tunable frequency, phase, pulse width, time between pulses, amplitude and resolution. These parameters are all configurable at run-time via a UART connection to a host computer which can run test scripts in MATLAB. Below follows a review of how well the control of each parameter was implemented in relation to the specification.

6.1 Carrier Frequency and Frequency Offset

The HF carrier is tunable via configuration of the DUC in the RF DAC. Offsetting from this value should ideally be done in a range of $\pm 150\,\text{MHz}$. In theory this is possible with the proposed design, however as mentioned earlier this is practically limited by the quality of the waveforms provided by the IF NCOs. A practical limit could be set to about $\pm 20\,\text{MHz}$, as this gives roughly 20 samples in a period.

One of the biggest problems in the design still left to be solved is that the summation of the I- and Q-signals was not successfully implemented in digital logic. The documentation of the DUCs in the Zynq board suggests that this should be possible after mixing with the HF NCOs. However, after many failed attempts a workaround where the signals where summed in post-processing was implemented. The fact that we are not performing the addition directly in the digital logic is most likely a cause for not being able to fully suppress the undesired sideband, as seen in the previous section, since sending the I- and Q-signals over separate channels might cause additional skew. That is, causing the channels not be perfectly in quadrature. We have cause for believing that this is the case as it was seen in the lab that manually calibrating the phase offset in one of the channels resulted in improved sideband suppression.

6.2 Phase Offset

As stated in the report, the phase resolution for phase offsets ranging from 0 to 2π is limited by the number of waveform samples in the ROMs of the IF NCOs. For the proposed implementation the achieved resolution was about 6.1 mrad corresponding to memory size of $2^{10} = 1024$ samples. In theory the specification of a phase resolution of less than 0.12 mrad could be achieved by increasing the size of the memory to $2^{16} = 65536$ samples, resulting in about 0.096 mrad resolution. However, this was not tested.

The correctness of the phase control was tested in the lab by sweeping the phase of the Q-channel and observing the change in relation to the I-channel. This is however not the intended use and was done only for demonstration purposes. Ideally, an additional DAC-tile on the Zynq board could be used to display a waveform with the phase offset added to *both* IF NCOs and compared with an output without the additional phase. There was however a problem with the number of channels available as we had to output both the I and Q waveforms for reasons explained above.

6.3 Pulse Width and Time Between Pulses

The envelope stored in the SRAM waveform-memory is completely arbitrary as the UART receiver/decoder reads and stores whatever data that is transmitted from the host, which is then used for amplitude modulation of the IF NCOs. Most commonly, however, a Gaussian is used. The pulse width of a Gaussian should according to the specification be configurable in the range 20 ns-1 µs. For the proposed design a lower limit of 1 µs was used due to limitations imposed by the available clock frequency. A pulse width of 20 ns results in a Gaussian envelope of only about 8 samples, which was determined to be insufficient. This limitation of pulse width will yield a maximum possible qubit fidelity of 98.79 % for this design, which unfortunately does not scale well for QPUs with more qubits than one.

The separation in time between consecutive pulses in a program can be set to be as low as a single clock period, that is $1/409.6\,\text{MHz} \approx 2.44\,\text{ns}$. This fulfills the specification of a maximum time between pulses of 5 ns. A module was also implemented to increase this time if desired. This allows the user to conduct Ramsey experiments, for example.

6.4 Amplitude Scaling

Control of the amplitude was implemented by means of fixed-point multiplication resulting in 512 possible values, ranging from 0 to max amplitude. This was done with a maximum observed relative error of 0.06%. Thus, our proposed design satisfies the specification of controlling the amplitude in 512 discrete steps. However, the measured voltage level for certain step values were higher than expected, and further testing is required to conclude that the design is 100 % correct.

6.5 Waveform Resolution

The specification states that the bit-resolution of the RF pulse should be able to be adjusted from 1 to the maximum bit-resolution of the DAC, i.e. 14 bits. This was not quite achieved as we found no way of controlling the number of bits after mixing the IF outputs with the NCOs of the DUC in the Zynq board. However, full control of the resolution of the IF waveforms was implemented. This might still provide some useful insights with regards to the effects of waveform quality on qubit fidelity. For example, HF DUCs are very power hungry which makes external analog mixers a viable option. Our design could then be used to observe how low-resolution IF data when combined with an analog mixer, or another high-resolution NCO, affects the state of a qubit.

6.6 UART communication

A UART receiver was implemented as a Moore FSM and successfully tested in post-implementation simulations as well as with changing some parameters of a preloaded waveform during run-time in the lab. Unfortunately, there was no time to test this module fully in the lab. Consequently we did not obtain a measurement of how fast program repetition rate could be achieved. Testing and debugging this module is therefore obvious next step in future development as this would enable running tests such as Rabi measurements and randomized benchmarking on an actual QPU.

6.7 Additional Future Work

The main goal of the project was to design an AWG with the capability of degrading waveform quality to observe the effects on qubit fidelity. We chose to focus on the resolution control, however other types of distortion and non-idealities could be emulated in software and/or hardware. This could include modeling clock-jitter, thermal noise, component mismatch and other noise and distortion introduced by data conversion.

It was also observed during testing in the lab, that increasing the amount of σ one included to the Gaussian envelope reduced the power of the harmonics in the frequency spectrum. Further testing of this phenomenon might provide some insight to how the Gaussian's pulse width should be defined for best results.

Bibliography

[1] B. Ehsan Baaquie and L.-C. Kwek, *Quantum Computers - Theory and Algorithms*. Singapore, Singapore: Springer, 2023, Accessed: Mar. 17, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-981-19-7517-2

[2] C. Krizian, "Instrument and measurement automation for classical control of a multi-qubit quantum processor," Master's thesis, Dept. Comput. sci. and Eng., Chalmers Univ., Gothenburg, Sweden, 2019.

[3] D. Castelvecchi, "IBM Releases First-Ever 1,000-Qubit Quantum Chip," *Scientific American*, Accessed: May 22, 2024. [Online]. Available: https://www.scientificamerican.com/article/ibm-releases-first-ever-1-000-qubit-quantum-chip/

[4] P. Krantz, M. Kjaergaard, F. Yan, T. Orlando, S. Gustavsson, and W. Oliver, "A Quantum Engineer's Guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, July, 2021.

[5] Z. Chen, "Metrology of quantum control and measurement in superconducting qubits," Ph.D. dissertation, Uni. of Calif., CA, USA, 2018.

[6] T. Abad, J. Fernández-Pendás, A. F. Kockum, and G. Johansson, "Universal Fidelity Reduction of Quantum Operations from Weak Dissipation," *Physical Review Letters*, vol. 129, no. 15, Oct., 2022, Art. no. 150504.

[7] D. M. Pozar, "Modulation techniques," in *Microwave and RF Design of Wireless Systems*. Hoboken, NJ, USA: Wiley, 2000, ch. 9.

[8] R. Lyons, "Quadrature signals: Complex, but not complicated," *IEEE Long Island Section*, 2008, Accessed: May 22, 2024. [Online]. Available: https://www.ieee.li/pdf/essay/quadrature_signals.pdf

[9] J. Mercade, "Rf signal generation with digital up-converters in AWGs," Tabor Electronics, White Paper, 2021, Accessed: May 22, 2024. [Online]. Available: https://www.taborelec.com/RF-Signal-Generation-with-Digital-Up-Converters-in-AWGs-White-Paper

[10] M. Welborn, "Direct waveform synthesis for software radios," in *WCNC. 1999 IEEE Wireless Communications and Networking Conference (Cat. No.99TH8466)*, vol. 1, 1999, pp. 211–215.

[11] Digilent, Pullman, WA, USA, *Nexys A7 Reference Manual*, Accessed: May 19, 2024. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

[12] AMD, Santa Clara, CA, USA, *Zynq UltraScale+ RFSoC Data Sheet: Overview*, 14th ed., 2023, Acessed: May 19, 2024. [Online]. Available: https://docs.amd.com/v/u/en-US/ds889-zynq-usp-rfsoc-overview

[13] W. Dally, C. Harting, and T. Aamodt, "Fixed- and floating point numbers," in *Digital Design Using VHDL: A Systems Approach*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2016, ch. 11.

A

Product Specification

A.1 Minimum Requirements

- User interface for setting a pulse program
- Envelope may be fixed as a Gaussian
- the AWG should generate a trig signal when the program is finished
- Fixed wait time of 400 µs between programs
- Each pulse should last for between 20 ns to 1 µs
- Time between pulses should not exceed 5 ns
- The phase should be adjustable from 0 to $\frac{\pi}{2}$ rad
- A carrier frequency of about 3–4 GHz is necessary to control the qubit
- Carrier frequency sweep range of ±2 MHz
- Amplitude settings of 0 %, 50 %, and 100 % of full-scale waveform

A.2 "Perfect" AWG Requirements

- Envelope may be completely arbitrary
- Pulses can change during run-time
- At most one clock cycle delay between pulses
- Adjustable wait time between pulse programs [2.7 µs, 10 ms]
- 2 ns time resolution
- Carrier frequency sweep range of ±150 MHz
- Phase resolution ≤ 0.1 mrad
- Amplitude range: 512 steps between 0 % and 100 % of full-scale waveform
- Adjustable bit resolution: 1–14 bits