

Master-Arbeit

Short Time Fourier Transform Pulse Generator for Trapped Ion Quantum Gates

erstellt von

Norman Krackow

Matrikel: 399214

Berlin, Februar 2020

Hochschullehrer: Prof. Dr.-Ing. R. Orglmeister, TU Berlin

Betreuer: Dr. Robert Jördens, QUARTIQ GmbH

Alexandru-Gabriel Pielmus, TU Berlin

Technische Universität Berlin, Fachgebiet Elektronik und medizinische Signalverarbeitung
Institut für Energie- und Automatisierungstechnik

Contents

Abbreviations	iii
Abstract	1
Zusammenfassung	3
1 Quantum Information Processing	5
1.1 Introduction	5
1.2 Quantum Logic	7
1.3 Trapped Ions	9
1.4 Mølmer-Sørensen Gate	11
1.5 Existing Hardware Approaches	13
2 ARTIQ Framework	15
2.1 Instrumentation for Quantum Physics	15
2.2 Sinara	16
2.3 ARTIQ Overview	17
2.4 Development Tools	19
3 STFT Pulse Generator	23
3.1 Phaser hardware	23
3.2 Description	25
3.2.1 Abstract	25
3.2.2 Functional	25
3.2.3 Interface	27
3.2.4 Practical Considerations	28
3.2.5 Specifications	31
3.3 Signal Processing	32
3.3.1 FFT	32
3.3.2 Interpolator	41
3.3.3 Upconverter	51

Contents

3.4	ARTIQ integration	53
3.5	Testing	56
4	Conclusion and Outlook	59
A	API Documentation	61
	List of Figures	67
	List of Tables	71
	Bibliography	73

Abbreviations

AOM	Acousto-Optic Modulator
API	Application Programming Interface
ARTIQ	Advanced Real-Time Infrastructure for Quantum physics
AWG	Arbitrary Waveform Generator
dBFS	dB relative to Full-Scale
BSD	Berkeley Source Distribution (license)
CIC	Cascaded Integrator Comb
COM	Center Of Mass
COTS	Commercial-Of-The-Shelf
CPU	Central Processing Unit
CSR	Control/Status Register
DAC	Digital to Analog Converter
DDS	Direct Digital Synthesis
DFT	Discrete Fourier Transform
DIF	Division In Frequency
DIT	Division In Time
DMA	Direct Memory Access
DSP	Digital Signal Processing

Contents

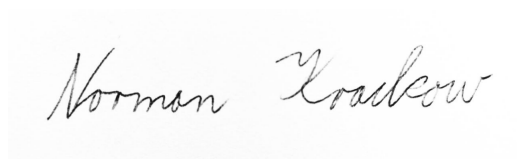
DUC	Digital Up-Converter
FIR	Finite Impulse Response (filter)
FFT	Fast Fourier Transformation
FHDL	Functional Hardware Description Language
FPGA	Field Programmable Gate Array
FTW	Frequency Tuning Word
GUI	Graphical User Interface
HBF	Half-Band Filter
HDL	Hardware Description Language
IC	Integrated Circuit
IF	Intermediate Frequency
ILA	Integrated Logic Analyzer
IO	Input/Output
I/Q	Inphase/Quadrature
LGPLv3+	GNU Lesser General Public License Version 3+
LSB	Least Significant Bit
LUT	Look-Up Table
LVDS	Low Voltage Differential Signaling
MSps	Mega Samples per second
NIST	National Institute of Standards and Technology
NCO	Numerically Controlled Oscillator
RF	Radio Frequency
SOPOT	Sum Of Powers Of Two
SED	Scalable Event Dispatcher

SerDes	Serializer-Deserializer
SFDR	Spurious Free Dynamic Range
SNR	Signal to Noise Ratio
SoC	System on Chip
STFT	Short-Time Fourier Transform
TTL	Transistor-Transistor Logic
PCB	Printed Circuit Board
PHY	Physical Layer
PLL	Phase-Locked Loop
POW	Phase Offset Word
PTB	Physikalisch-Technische Bundesanstalt
RAM	Random Access Memory
RTIO	Real Time Input/Output

Erklärung

Die selbständige und eigenständige Anfertigung versichert an Eides statt.

Berlin, den April 11, 2021

A handwritten signature in black ink on a light gray background. The signature reads "Norman Zadeow" in a cursive script.

Unterschrift

Danksagung

Der Autor dankt hiermit allen, die durch ihre Unterstützung zur Entstehung dieser Arbeit beigetragen haben. Besonderer Dank gilt Herrn Dr. Jördens von QUARTIQ.

Abstract

Quantum information processing is a rapidly evolving field at the intersection of physics and computer science. Its goal is to harness the inherent computational power of quantum mechanics for information processing.

In recent years, arrays of trapped atomic ions have emerged as a vehicle to store and manipulate quantum information. The information is therein represented as states in the atomic structure and manipulated by the interaction with electromagnetic radiation. Due to the natural fragility of quantum information, this interaction must be mediated in a precisely controlled fashion.

This work presents the development of a device for the generation of signal pulses for quantum logic gates. The pulse can be specified using a set of configuration parameters and emitted with nanosecond precision. The Short-Time Fourier Transform (STFT) is employed as the key concept for signal computation. High spectral purity and optimal signal to noise ratio are achieved via the use of custom Digital Signal Processing (DSP) circuits in a Field Programmable Gate Array (FPGA).

The device was integrated in the Advanced Real-Time Infrastructure for Quantum physics (ARTIQ) framework used by leading groups in the field.

Zusammenfassung

Die Quanteninformatik ist eine rasch fortschreitende Disziplin zwischen Physik und Informatik. Sie hat zum Ziel, die computationale Stärke der Quantenmechanik zur Informationsverarbeitung zu nutzen.

In den letzten Jahren haben Anordnungen von gefangenen Ionen gezeigt, dass sie als Medium zur Speicherung und Verarbeitung von Quanteninformation dienen können. Hierbei ist die Information durch Zustände in der atomaren Struktur repräsentiert und kann durch die Interaktion mit elektromagnetischer Strahlung manipuliert werden. Aufgrund der inhärenten Sensitivität von Quanteninformation muss diese Interaktion präzise dirigiert werden.

Diese Arbeit präsentiert die Entwicklung eines Geräts zur Erzeugung von Signalpulsen für logische Quantengatter. Der Puls kann durch einen Satz Konfigurationsparameter spezifiziert und mit Nanosekunden-Präzision ausgesendet werden. Die Kurzzeit-Fouriertansformation (STFT) bildet hierbei das zentrale Konzept der Signalberechnung. Eine hohe spektrale Reinheit und optimaler Signal-Rausch-Abstand werden durch maßgeschneiderte Schaltungen der digitalen Signalverarbeitung (DSP) in einem programmierbaren Logikschaltkreis (FPGA) erreicht.

Das Gerät ist in ein weltweit führendes System zur Steuerung von Quantenexperimenten, der “Advanced Real-Time Infrastructure for Quantum physics (ARTIQ)”, integriert.

1 Quantum Information Processing

This first chapter addresses a reader without a physics background and gives context about the reasoning and design decisions that went into developing the STFT pulse generator. Due to the complexity of the application background, this chapter has no claim to completeness.

1.1 Introduction

While often perceived as an abstract concept, information always corresponds to the state of a physical system. This is true for the contents of a computer memory, the DNA of a living organism or the modulation of electromagnetic radiation coming from a distant star. So does computation and the time evolution of a physical system. A classical computer subjects the content of its memory to some logical function in order to calculate an outcome. Underlying this is a physical process in which some physical quantity like the flow of electrical current is mediated by some physical interaction like the dynamics of a transistor.

Starting in the 1980s, physicists wondered if they could simulate physics on a computer with Richard Feynman recognizing the exponential resource demand of a classical computer to emulate quantum mechanics [Fey82]. He went on to formulate the idea of a “Quantum Mechanical Computer”[Fey86], laying out the fundamental framework and elementary building blocks for such a device. Other theorists followed suit with David Deutsch defining a Quantum Turing Machine and generalizing the Church-Turing thesis to physical systems[Deu85]:

“Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means.”

1 Quantum Information Processing

Bernstein and Vazirani [BV97] subsequently went a step further, theorizing that a universal computing machine would be able to *efficiently*¹ simulate any other physical system, sometimes called the *strong* Church-Turing thesis.

It is currently unknown whether the (strong) Church-Turing thesis is true for a quantum computer. While fundamental problems arise with general relativity, recent works by John Preskill suggests that universal quantum computers can efficiently simulate quantum field theories and probably the full standard model of particle physics[Pre18]. Interestingly, although conjectured by information theorists, it is yet to be proven that a quantum computer is strictly more powerful than a probabilistic classical computer.

While still an open debate, quantum information theory has given rise to actual quantum algorithms with superior performance to known classical algorithms (Shor[Sho99], Grover[Gro96], etc.) as well as fundamentally secure communications (quantum teleportation[Ben+93], BB84[BB20]). Since technologies capable of performing those schemes would be very disruptive for many of today's communication and cryptography systems, there has been considerable interest from government agencies, funding much of the early experimental work on quantum computers and quantum communications. While secure quantum communication has been demonstrated over large distances [Lia+17], a quantum computer capable of executing Shor's or Grover's algorithm on technologically relevant scales does not yet exist.

However, experimentalists have made considerable progress towards realizing bigger and better quantum systems with programmable dynamics. A number of competing platforms and systems have emerged with a growing number of researchers making progress on each. Since the STFT pulse generator presented in this work is intended for use with trapped ions, only this platform will be discussed in section 1.3.

Today quantum computers, communications and metrology are often aggregated to the term quantum technologies. With rising public and private investments,

¹Efficiently in this context refers to a polynomial increase in computing resources for a polynomially increased problem size.

the field is transitioning from laboratory to industrial scale. There are a fast-growing number of companies with many specializing in different aspects of the technology “stack”. In 2020, the German government pledged investments of $2 \cdot 10^9$ € for quantum technologies in conjunction with the “Konjunktur- und Krisenbewältigungspaket”[Bun20] and an industry consortium has framed a German “Quantum Roadmap”[Gmb21].

1.2 Quantum Logic

Quantum information processing is usually described using the quantum circuit model. In a similar way to classical logic circuits, inputs are subjected to a number of logic gates, altering their states and producing an output. While a classical computer works with bits, the fundamental unit of information in a quantum computer is a *qubit*. Unlike a classical bit, which can be in one of two states, a qubit can be in any *superposition* of two states. However, the state of a qubit can not be directly observed. Instead one needs to perform a measurement on the qubit, collapsing it into one of two basis states. Basis states are states in which the qubit can be observed to be in. Therefore every quantum algorithm always ends with a measurement, revealing the outcome of the computation.

The other fundamentally different concept in quantum computing is *entanglement*. While the state of a classical system is fully described by describing each element individually, this is not the case for a quantum system, where the state of a system can only be fully described as a whole, incorporating mutually shared information. The consequence is, that the amount of information contained in a system rises exponentially with a linear increase in size. Furthermore, the state of an entangled quantum system can only be changed globally, with operations on each individual qubit affecting the global state.

A quantum circuit can exploit these quantum phenomena by applying quantum logic gates. A single qubit logic gate can manipulate the state of a single qubit, for example taking it from a basis state into a superposition state. A multi qubit logic gate can take a number of qubits into or out of a state of entanglement.

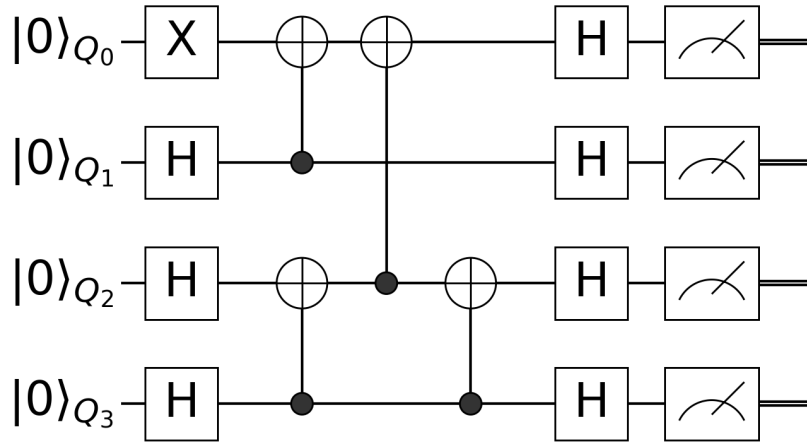


Figure 1.1: Exemplary quantum circuit[yao] that generates a highly entangled state between 4 qubits (before the measurement). After state preparation on the very left, single and two qubit gates are applied, followed by a measurement at the end which reveals the final quantum state. In this circuit the qubits will have a 50/50 chance to all be 1 or all be 0, an outcome not possible with classical logic.

By exploiting both of those properties, it is possible to find algorithms with better performance than known classical algorithms[Sho99][Gro96].

For entanglement and superposition to be conserved, the system must not interact with the outside during computation. Unfortunately, since a real system is never completely isolated from the outside², real qubits and quantum gates are hard to realize. The time a real qubit can retain its quantum information is called the qubit coherence time, with the best platforms now achieving many seconds or even minutes. The other defining metric is the gate fidelity which gives the probability of successfully applying the gate. Current records are around 99.999% for single qubit gates and 99.99%[Gae+16] for two qubit gates.

While these values are not sufficient to successfully perform big computations natively, the concept of quantum error correction gives hope that complex algorithms

²At least not when it is programmable from outside.

can be performed on noisy computers by distributing the information involved in the computation in the entanglement between the real qubits. However, while theoretically possible, quantum error correction comes with a big increase in system size for systems with subpar gate fidelity. As of today, a fully error corrected (“logical”) qubit has not yet been realized.

1.3 Trapped Ions

Compared to other platforms such as superconducting, solid-state or optical qubits, trapped ion qubits currently show the best coherence times[Wan+17], single qubit and two qubit gate fidelity[Gae+16] and benefit from all-to-all connectivity. All-to-all refers to the fact, that multi qubit gates can be performed in arbitrary constellations, which is not possible on other systems. Furthermore, qubit initialization and readout can be performed with near perfect accuracy in a trapped ion system. On the other hand, other platforms are often simpler to operate, whereas trapped ion setups demand precision lasers and optics.

In many trapped ion systems a qubit is represented by a hyperfine electronic transition in the atomic structure. Hyperfine transitions are small transitions within an electron orbital that arise from electron-nucleus interactions. A number of individual ions are trapped in a vacuum via static and oscillating electric fields³. In modern traps, the electrodes are arranged in 2D on a microfabricated chip with the ions forming a linear array above the chip. The linear arrangement stems from the trapping fields asserting strong confinement in two spatial dimension and weak confinement in the third. The charged ions also repel one another and therefore form an ion crystal with coupled motion between the ions.

To control their motion, the ions are first cooled to nanokelvin levels using various “laser cooling” techniques. Laser cooling uses the fact that the interaction of atomic states can be dependent on the ions motion⁴. Using this property, ions can

³The invention of the paul ion trap in the 1950s in Bonn was awarded with a Nobel Prize for Wolfgang Paul.

⁴Also awarded with a Nobel Prize for Steven Chu, Claude Cohen-Tannoudji and William D. Phillips in 1997.

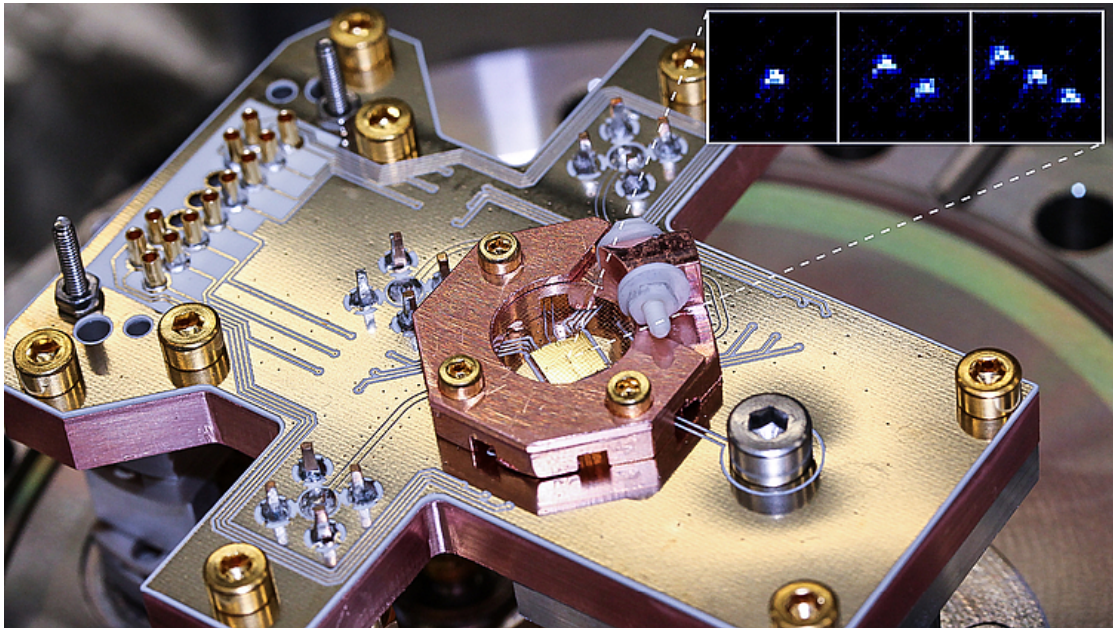


Figure 1.2: Ion trap fabricated at the “Physikalisch-Technische Bundesanstalt” (PTB) Braunschweig[Bra]. The inset shows arrays of fluorescing beryllium ions.

even be cooled to their motional ground state. The qubits are initialized using “optical pumping”, where the electron is first excited into a high energy state which then coherently decays into the qubit ground state. Readout of the qubit state is achieved via fluorescence, where the electron is continuously excited into a higher energy level, quickly decaying back to the original level, emitting a photon. If the ion is prepared by a process called “shelving”, it will only fluoresce if the qubit was in the ground state⁵.

Once the qubit is initialized, arbitrary single qubit gates can be applied by illuminating the ion with radiation resonant to its qubit transition frequency. Since the qubit transition is usually in the microwave region, it can either be driven directly using near-field electrodes or via a Raman process. The Raman process involves two laser beams with frequencies that differ by the qubit transition frequency. The illumination is called a π pulse if the qubit exactly flips its state and a $\frac{\pi}{2}$ pulse, if

⁵Another Nobel Prize was awarded to David J. Wineland in 2012 for developing experimental methods for quantum experiments with trapped ions.

the state is changed by 90° , which would bring a qubit from the ground state into a superposition state.

Multi qubit gates are more difficult since the hyperfine states of different ions cannot directly interact with one another. However, using similar techniques to laser cooling, the internal qubit states can couple to the motional modes in the ion crystal, thereby enabling communication between the internal ion states. A motional mode refers to the degrees of freedom for coupled vibrations of the ions, forming a complex structure in large ion crystals. The first such technique was invented by Cirac and Zoller[CZ95] in 1995, representing the last fundamental building block for a quantum computer based on trapped ions.

1.4 Mølmer-Sørensen Gate

Mølmer and Sørensen significantly improved the multi qubit entanglement process via shared motional modes[SM99]. Instead of requiring the ions to be in the groundstate for their coupling mode, as is the case for Cirac-Zoller, the ions can already share several quanta of motion (phonons) in their mutual vibrations.

The internal qubit states are coupled to the motional modes by driving the qubit transition using a light-/microwavefield with a slightly off-resonant frequency. Due to their quantized nature, the ion modes can only couple to the field if the energy difference due to the off-resonant frequency is accommodated in the motional state. This way the qubit becomes entangled with the motional mode, accommodating the energy difference. If the frequency is slightly higher than the qubit “carrier” frequency, the mode is driven in the “blue sideband”, for a lower frequency it is driven in the “red sideband”.

When driven at red and blue sideband simultaneously, the ion is “virtually” entangled with the mode, which simultaneously gains and loses a “virtual” quantum of energy. By illuminating more than one ion with a lightfield which simultaneously drives the red and blue sideband of a shared motional mode, *all* the ions become entangled with the “virtual” mode simultaneously. If the ions are illuminated

1 Quantum Information Processing

by a pulse with exact timing and detuning from the mode frequency (which is itself detuned from the qubit frequency), their internal states become maximally entangled while the “effective” motional mode does not change.

However, a real Mølmer-Sørensen gate in a large ion crystal is complicated by several imperfections. The first is that ideal pulses are hard to achieve and small timing, frequency and intensity fluctuations can lead to degraded gate fidelities. To achieve robustness against these degradations a reliable pulse shaping method needs to be employed.[Zar+19]. Moreover, it is impossible to exclusively drive the common “bus” mode in the ion crystal without also coupling to other modes (an exemplary ion mode spectrum is depicted in Figure 1.3). Although small, this coupling would leak quantum information to the ion motion and therefore impair the gate fidelity. By intentionally coupling to many ion modes simultaneously and ensuring their disentanglement with the transferred quantum information at gate termination, one can accommodate for the complex mode spectrum in large ion crystals[Sha+20].

This complex subject is similar for various quantum systems and has attracted a lot of research focus in recent years. Theorists have proposed various schemes to

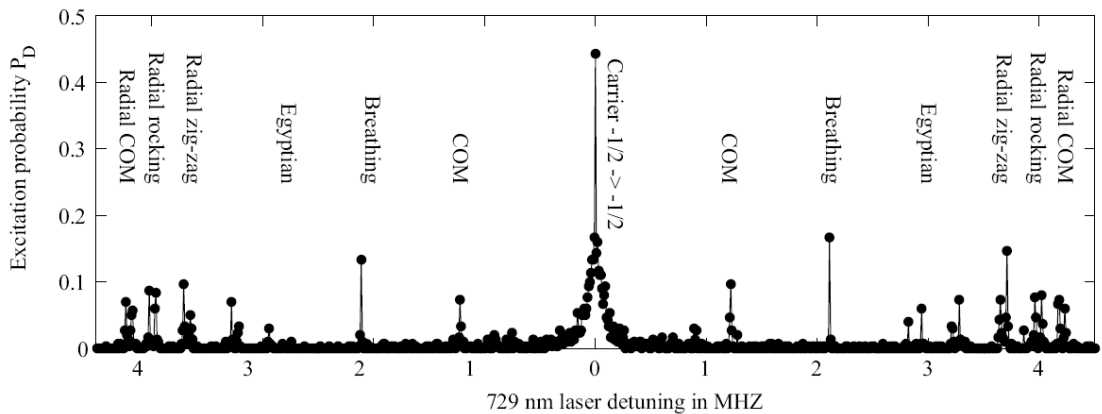


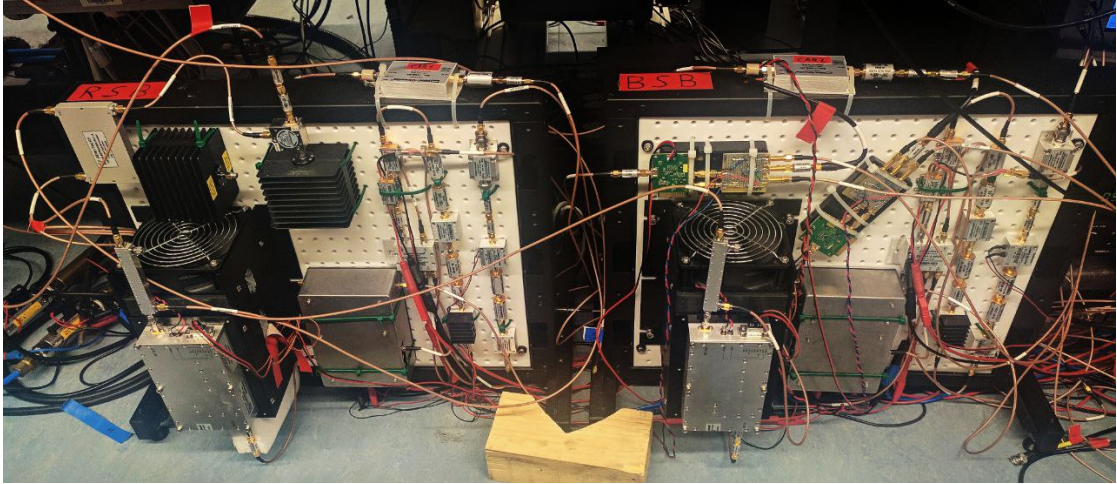
Figure 1.3: Exemplary ion mode spectrum[Cal]. The symmetrical structure of the red (left) and blue (right) sideband is due to various vibrational modes in the ion crystal. The modes are named by their movement pattern like Center Of Mass (COM) where the ions all move back and fourth in unison.

optimally control the gates and make them robust. Recently the FU Berlin physics department also established a working group on quantum dynamics and control[19]. Yet, experimentally achieved gate fidelities still lag far behind the theoretically predicted values.

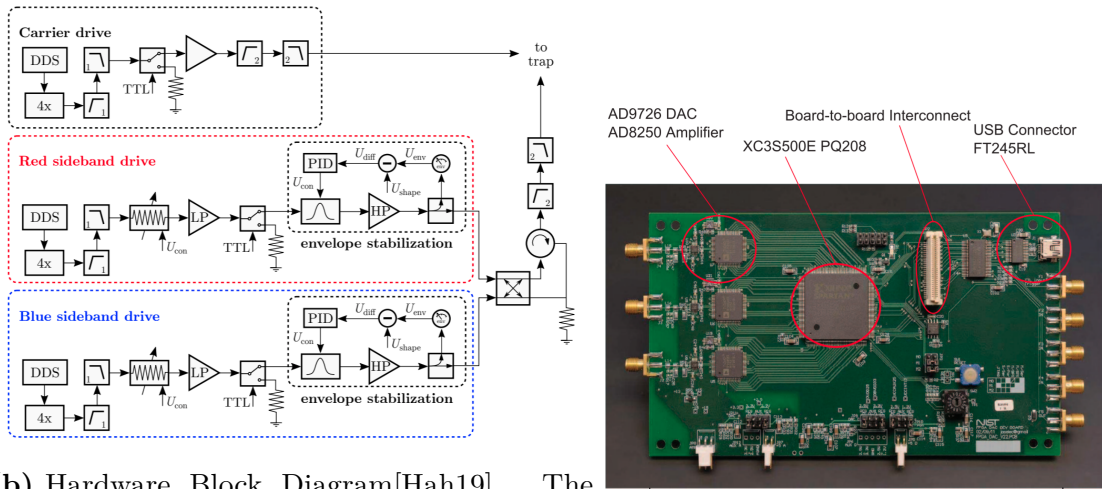
1.5 Existing Hardware Approaches

With a growing number of researchers pushing for higher gate fidelities in trapped ion systems, several hardware approaches for pulse generation are being explored. While some of the schemes make use of Commercial-Of-The-Shelf (COTS) products like Arbitrary Waveform Generators (AWGs), many systems are custom built by the experimentalists. Since the focus is usually on publishing the result of an experiment, scalability and generality are less of a priority. Furthermore, as the schemes become more sophisticated, specific engineering knowledge like RF design and DSP become necessary.

As a general survey of the hardware landscape for pulse generation is outside the scope of this work, the system developed in [Zar+19] at the PTB is given as an example. As depicted in figures 1.4b and 1.4a, many bulky and discrete components are necessary to drive both sidebands and the carrier. The pulse shape is processed by the AWG shown in Figure 1.4c[Bow+13]. While appropriate for basic tasks, the limited digital bandwidth, memory and FPGA size make scaling up to more complex pulses or bigger ion crystals challenging.



(a) Foto of the RF components for driving the red and blue sideband (labeled RSB and BSB in the picture).



(b) Hardware Block Diagram[Hah19]. The U_{shape} signal is provided by the NIST AWG.

(c) NIST AWG PCB[Bow+13]

Figure 1.4: PTB Hardware for Mølmer-Sørensen microwave gates.

2 ARTIQ Framework

2.1 Instrumentation for Quantum Physics

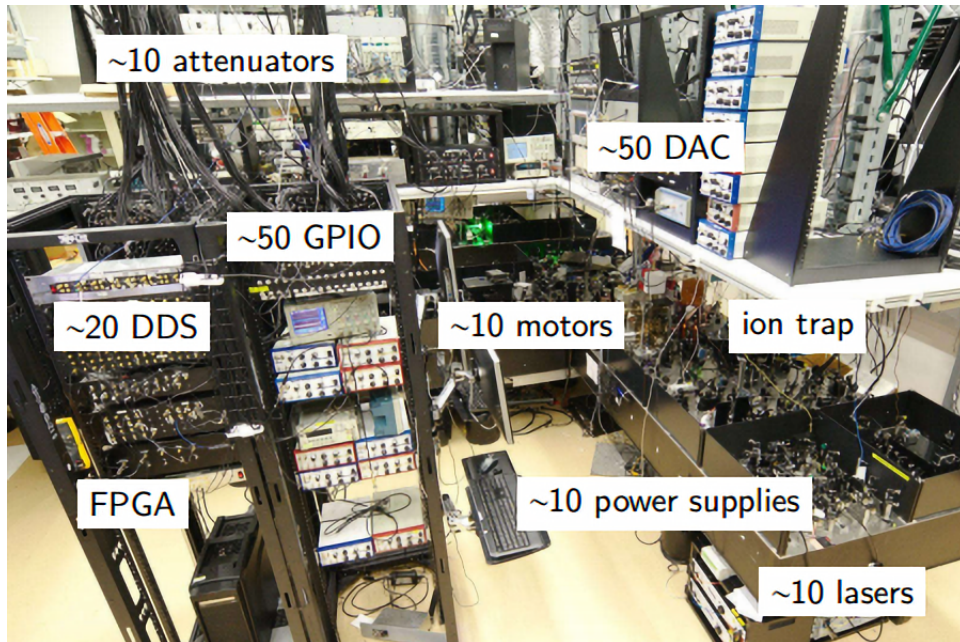


Figure 2.1: An image of a quantum laboratory with several instrumentation devices annotated[M-Lb].

As depicted in Figure 2.1, laboratories for quantum physics often contain a vast array of optical, electrical and mechanical systems. Even for simple experiments like changing the state of a hyperfine qubit, many inter-dependencies emerge. Just loading the ion-trap is a many-step process and often requires several tries. Therefore all of these systems have to be precisely orchestrated while maintaining

the ability to be reconfigured for different experiments. Since quantum experiments are statistical in nature, many “rolls” have to be performed and experiment logic has to be built in a dynamic way using loops and branching. Furthermore, the short-lived nature and dynamic behavior of quantum systems impose constraints on timing and jitter in signal sources.

2.2 Sinara

The Sinara ecosystem is a collection of hardware modules for quantum physics. Each module is designed for one or several functions in a quantum experiment. Examples include:

- 4 Channel, 1-400MHz DDS “Urukul” for driving AOMs
- 8 Channel, 16-bit, 1.5MSps ADC “Sampler” for photo-diode current digitization
- 32 Channel, 16-bit, 1MSps DAC “Zotino” to drive trap electrodes
- 4 Channel, 13.6GHz Microwave Synthesizer “Mirny” for RF generation

The modules are connected to the “Kasli” main module using ribbon and coaxial cables. Kasli acts as the central controller for the system and can itself be connected to a host computer using Ethernet. It features the Artix-7 100T FPGA containing custom Central Processing Units (CPUs) and gateway to communicate with the host and drive the various daughter cards. Several Kaslis can be connected as core and satellite systems, sharing a common time-base for experiment control. All of the modules are designed to the Eurocard standard and can be assembled in a Eurocard 3U chassis.

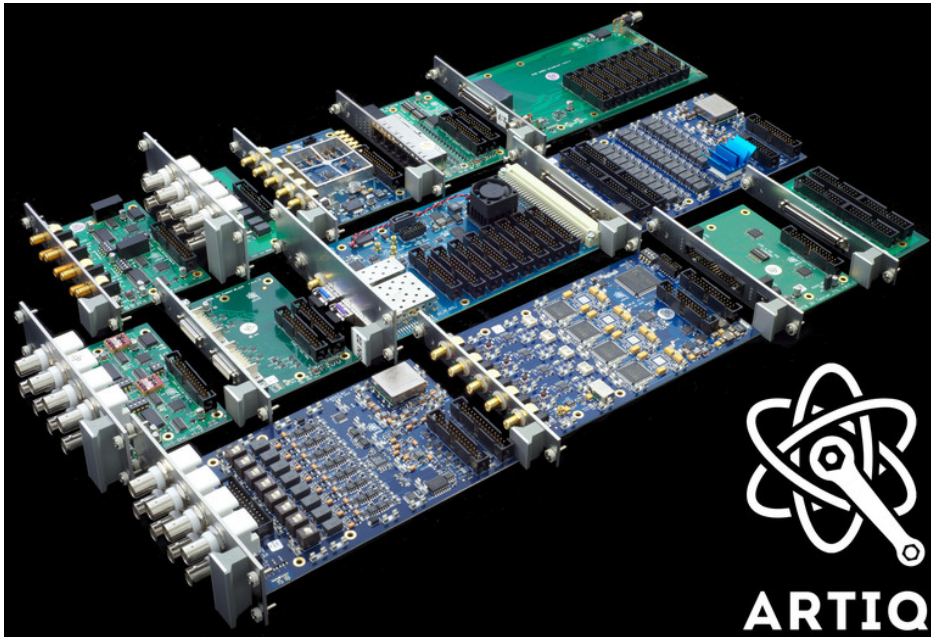


Figure 2.2: A selection of Sinara hardware modules[M-Ld]. In the standard configuration the modules are mounted in a Eurocard 3U chassis.

The hardware is “CERN OHL v1.2” open-source licensed and developed by a multi-institutional collaboration of companies, universities and research institutions including the National Institute of Standards and Technology (NIST), Warsaw Technical University, University of Oxford, M-Labs Ltd. and QUARTIQ GmbH, with the latter two companies distributing the devices.

2.3 ARTIQ Overview

ARTIQ is built specifically to address the challenges in quantum experiments described in section 2.1. As the split between performance and reconfigurability necessitates sophisticated systems from various disciplines, a technical description of ARTIQ is outside the scope of this work. The following will give a brief overview of the main components with a focus on the segments relevant for integrating the STFT pulse generator.

2 ARTIQ Framework

The user can describe an experiment in the Python programming language. The program flow is split between code that runs on the host computer and code which is executed on the embedded kernel CPU in the Kasli FPGA. The latter has to be placed in a function with a *@kernel* decorator. When starting an experiment, the Python interpreter will dispatch this function to the ARTIQ backend which will then compile the code to be run on the kernel CPU¹. When the experiment reaches the point where a kernel function is called, the ARTIQ backend sends the code to the embedded Random Access Memory (RAM) via a separate communication CPU where it is executed by the kernel CPU. During execution the kernel can communicate with the host through the communication CPU. Similarly, a (standard Python) host function can be called from within a kernel function with communication being handled automatically. The subset of Python which can be compiled for execution on the kernel is called ARTIQ Python.

To interact with the Sinara hardware modules used in the experiment the physicist can use the specific driver Application Programming Interface (API) from within a kernel function. The API abstracts the underlying transactions between the CPU and the modules via the Real Time Input/Output (RTIO) system. The RTIO basically decouples the dynamical program flow from “wall clock” time. Transactions with the modules are not evoked at some time in the program, but instead scheduled to an exact point in “wall clock” time. The Scalable Event Dispatcher (SED) handles all of the necessary logic for sorting the transactions and acts as a data buffer².

The RTIO PHYs³ accommodate the physical interfaces to the modules. Those can range from simple I/O lines to custom gigabit interfaces like fastlink. Additionally, some RTIO PHYs feature real-time logic for control and signal processing and can write and read the main RAM via Direct Memory Access (DMA). On the module side the communication is handled either by another FPGA or directly relayed by the hardware.

¹A description of the various software tools and compilers is omitted

²Several other logical functions have been omitted in the interest of brevity.

³short for “Physical Layer”

ARTIQ is developed and maintained by M-Labs in collaboration with QUARTIQ and several other institutions. It is free software under LGPLv3+. A more detailed documentation can be found in the manual [M-La].

2.4 Development Tools

ARTIQ relies upon a suite of electronic design tools for development and operation. This section briefly mentions a selection used in the development of the STFT pulse generator. All of the tools except for Vivado are open-source software under various BSD licenses.

Python

Over the last decades, the Python programming language has become ubiquitous across many fields. Popularized by being readable, open, and portable, it now forms the core for tools in many emerging disciplines like machine learning, quantum computing or bioinformatics. Using the well-established NumPy, SciPy and Matplotlib packages, Python can be an engineering tool for scientific computing and signal processing. ARTIQ uses Python for scripting experiments and as the basis for the MiGen Hardware Description Language (HDL).

MiGen

MiGen[**MiGen**] is a Functional Hardware Description Language (FHDL) originally developed by Sébastien Bourdeauducq for the Milkymist video synthesizer. In contrast to standard HDLs such as Verilog and VHDL it provides a more generic way to describe hardware. As all of the basic Python functionality like branching, loops, functions and objects, as well as Python extensions remain usable, hardware can be described in an abstract form. For example one could have a FIR filter module that takes as inputs the design specifications and then uses SciPy and MiGen to generate the optimal filter hardware.

2 ARTIQ Framework

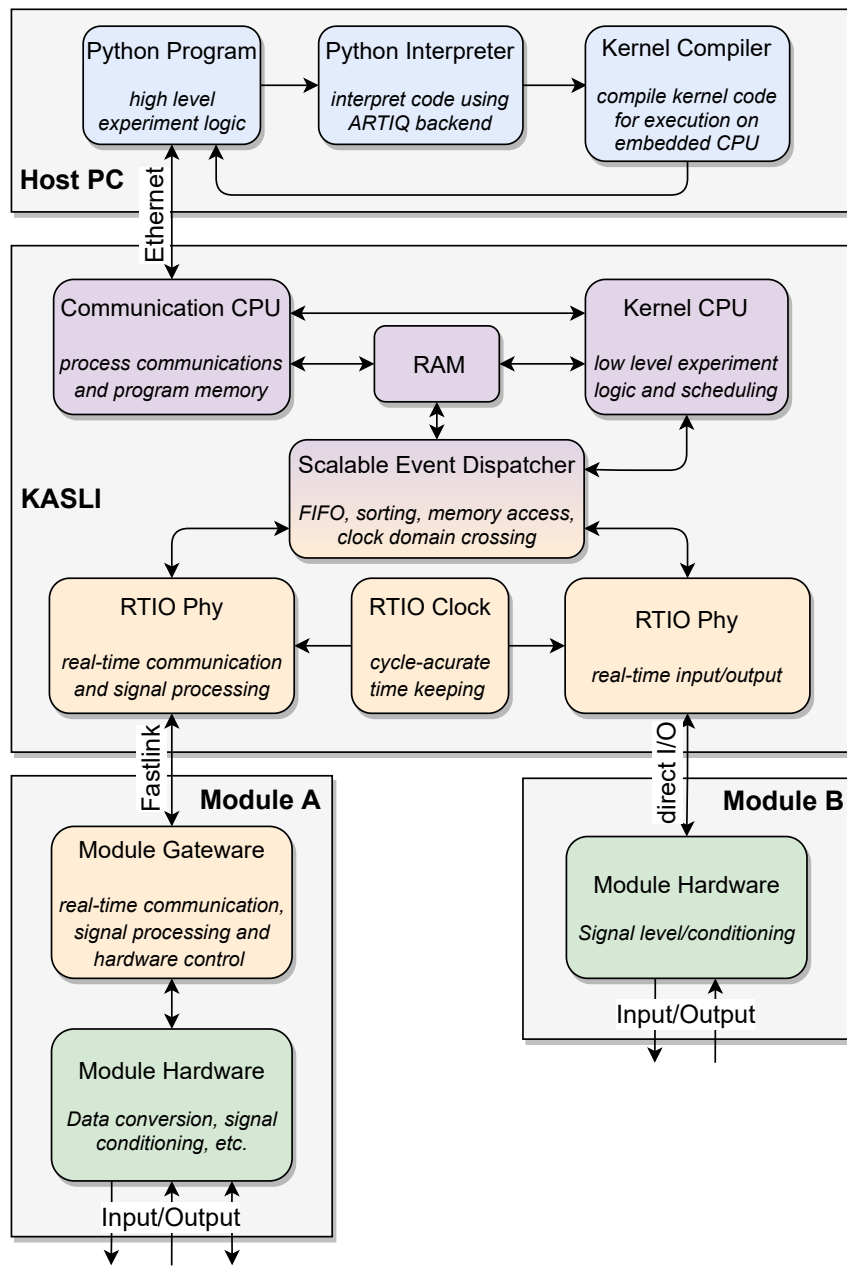


Figure 2.3: The ARTIQ architecture spans several devices and abstraction layers. Shown in blue are high level software functions running on the host PC. Purple are low level software functions on the Kasli FPGA. Gateway is shown in orange and hardware is depicted in green. The unannotated arrows represent various interfaces, communication or logical connections.

The same is true for the built-in simulator. Stimulus and simulated response can be scripted and processed with the full Python toolkit. Circuit and Model can be co-developed and numeric equivalence can be checked automatically. This enables more software-like design flows like unit-testing and continuous integration.

There are a number of difficulties and pitfalls in hardware development with MiGen. First, the use of Python requires the developer to be familiar with the syntax, style and object-oriented paradigm of the language. Coupled with the complexity of gateway design, this can be overwhelming for beginners. Secondly, the built-in simulator sometimes behaves differently to synthesized gateway from different toolchains. This is partly due to ambiguities in the intermediate Verilog code generated by MiGen and interpreted by downstream tools. But also several hardware descriptions such as signed fixed-point arithmetic or memory initialization are simulated differently to what is described in the output Verilog code. This can lead to time-consuming debugging.

LiteX [Ker+19] and MiSoC are examples of System on Chip (SoC) design frameworks using MiGen. In addition to QUARTIQ and M-Labs, companies like Oxford Ionics and Riverlane employ MiGen for quantum technologies.

MiSoC

Extending the FHDL functionality, the MiSoC framework provides designers with several utilities and modules for SoC development. Infrastructure such as buses, streams, timers, Integrated Logic Analyzers (ILAs) and connectivity can be specified in Python while repetitive tasks such as bus topology and memory mapping are automated. Several RISC-5 and OpenRISC soft-CPU's can be connected to an ecosystem of included peripherals including: UART, GPIO, NOR flash controller, SPI flash controller, Ethernet MAC, etc.

Additionally MiSoC contains an increasing number of customizable DSP cores including CORDIC, DDS, FIR and CIC.

Vivado

As most of the Sinara hardware uses XILINX FPGAs for the digital systems, the framework relies on Vivado to actually synthesize the gateway. However, instead of using the Vivado Graphical User Interface (GUI), MiGen comes with tools that invoke synthesis with the right parameters.

The designer can inspect the Vivado build logs to see if the synthesis process produced the intended gateway at the necessary performance. Correctly interpreting the logs does however require specific experience. Finding the critical path in an under-performing circuit can be very time-consuming as Vivado shuffles the primitives around during optimization and therefore reports random paths as being critical. Similar problems arise with other synthesis tools and timing optimization remains a challenging task for gateway designers.

3 STFT Pulse Generator

3.1 Phaser hardware

As a new addition to the Sinara ecosystem, “Phaser” takes the role of a versatile AWG with integrated RF upconversion. It could also be seen as a transmit only Software Defined Radio (SDR). Being in a similar frequency range as modern 4G/5G wireless systems, it can make use of high performance COTS components from the communications industry. The Phaser Printed Circuit Board (PCB) is depicted in Figure 3.1 with the main Integrated Circuits (ICs) annotated.

As the main digital IC, the XILINX Artix-7 XC7A100T FPGA [20] provides the following resources:

- 101,440 6-LUT logic cells
- 240 DSP48E1 DSP slices
- 4860Kb block RAM
- 144 high-speed differential or 300 single-ended IOs

Those resources have to be shared between communications, control, interfaces, glue logic and digital signal processing, with the latter taking up the lion’s share.

The four channel, 16-bit, 1.25 Gsps DAC34H84 Digital-to-Analog Converter (DAC) [15], originally developed for cellular communications, features a range of additional functionality. Built specifically to drive I/Q RF modulators, it can perform internal digital interpolation and upconversion to a wide frequency range. Integrated

3 STFT Pulse Generator

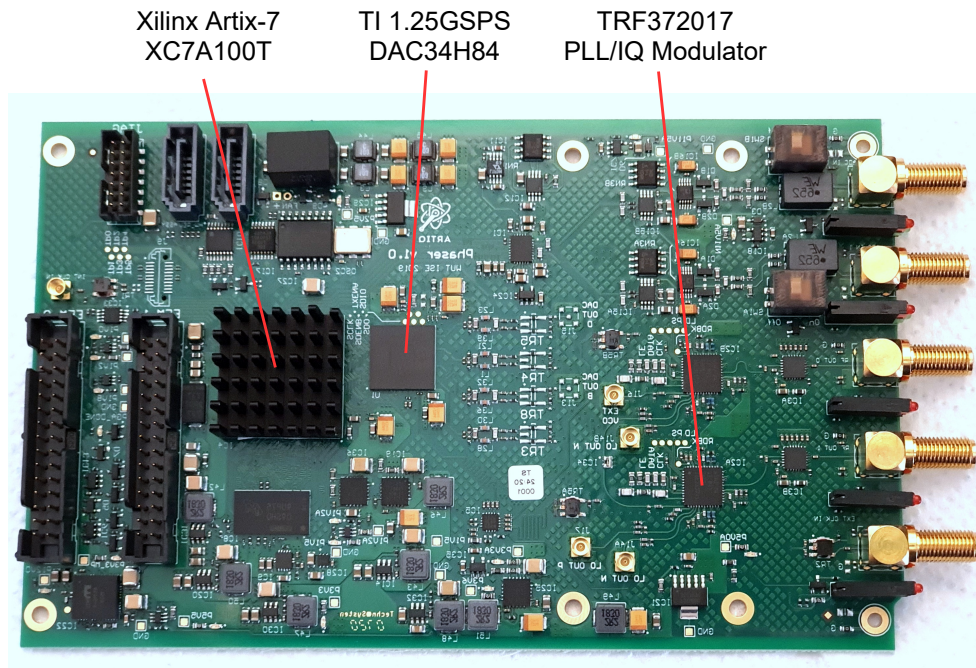


Figure 3.1: Phaser[Kas] PCB with the main ICs annotated.

quadrature modulator correction enables compensation for I/Q imbalance that would lead to undesired spectral components.

Also developed for wireless infrastructure applications, the TRF372017 integrated IQ modulator/PLL/VCO [16] is characterized by its low phase/amplitude noise and high linearity. As it packs several functions into one package, board space is reduced and parameters can be easily changed via a single interface.

Phaser was designed as a hardware platform for different signal generation schemes. In the “classic” configuration the FPGA gateway hosts fixed interpolators with subsequent Digital Up-Converters (DUCs) for the two RF output channels. The interpolators are fed with samples from the main Kasli module via a Fastlink interface.

As hardware development was not part of this work, it is treated as a commodity and analysis of the analog performance is omitted.

3.2 Description

3.2.1 Abstract

As discussed in section 1.3, novel and future approaches for Mølmer-Sørensen-style gates can require complex pulse spectra on several frequency bands. The main design goal of the STFT pulse generator is therefore to synthesize such spectra. Furthermore, the pulse should be narrow-band windowed and emitted with precise timing.

The architecture makes use of a number of combined signal sources with several Intermediate Frequencies (IFs). By modulating complex signals onto a final analog RF carrier, the spectrum can be shaped in the baseband with minimal undesired spectral components in the RF.

As a compromise between complexity and functionality, 3 sets of 1024 tones with customizable tone-spacing can be placed in a 400MHz window. This window can itself be placed somewhere within $[-400, 400]$ MHz from the $[0.3, 4.8]$ GHz carrier. This combination allows for a spectral composition in which carrier leakage and I/Q imbalance artifacts will show up far away from the desired spectral components.

3.2.2 Functional

Figure 3.2 depicts the pulse generator block diagram with the parameters listed in table 3.1. The main signal computation consists of 3 STFT branches. Each branch contains one block-FFT¹ core with repeater, two interpolators for real and complex signal components and one DUC, illustrated as sinusoidal signal source and multiplier. Each FFT core gets a set of 1024 complex frequency parameters $[a_{-512}, a_{511}]$ which are transformed into a time-domain representation. The FFT output is then continuously repeated and fed into the variable interpolators. As short temporal blocks are stitched together, we call this a Short-Time Fourier Transform

¹Since an FFT and an inverse FFT are basically equivalent from an engineering point of view, “FFT” will refer to either in the context of the STFT pulse generator.

3 STFT Pulse Generator

(STFT). The interpolators compute the missing samples to the full 500MSps at which the DAC is driven. Valid interpolation rates N are 2 and multiples of 4. The output of each interpolator is finally upconverted by the frequency f_0 which can be specified at mHz resolution in a $[-250, 250]$ MHz interval.

The outputs of the STFT branches are summed up and then multiplied with the output of the window signal path. As the window path is only real-valued, real and complex signal parts of the branches are both multiplied by the same signal. The window features the same block-FFT core as the branches with the imaginary time-domain output being discarded. The real part is interpolated by another interpolator with a maximum rate-change of $K = 8192$. This window can be gated and triggered with deterministic latency. After being transferred to the DAC, the signal is interpolated again to 1GSps and upconverted by $f_1 = [-500, 500]$ MHz. The DAC then converts the real and complex signals to the analog domain and drives the I and Q inputs of a final, analog I/Q modulator, in fig. 3.2 shown as an analog mixer. The RF input of the modulator is fed by a Phase-Locked Loop (PLL), shown as an analog signal source, which can synthesize a frequency in the $f_2 = [0.3, 4.8]$ GHz range.

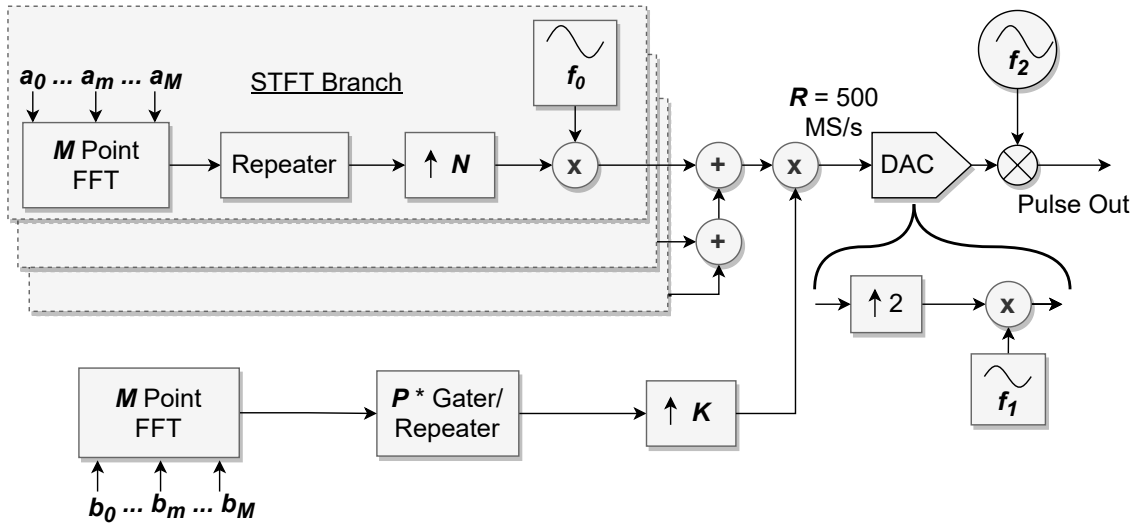


Figure 3.2: STFT pulse generator Block-Diagram

Parameter	Description	Constraint
R	base sample rate	fixed 500MHz
M	FFT size	fixed 1024
P	number repeated windows	integer
N	branch upsampling factor	2, 4, 8, 12, ..., 1024
K	window upsampling factor	2, 4, 8, 12, ..., 8192
a_m	branch FFT tone parameters	complex 2x16bit fixed-point
b_m	window FFT parameters	complex 2x16bit fixed-point
f_0	internal DDS frequency	$[-250, 250]$ MHz
f_1	DAC DDS frequency	$[-500, 500]$ MHz
f_2	PLL LO frequency	$[0.3, 4.8]$ GHz
t_{pulse}	pulse time	$t_{pulse} = M \cdot P \cdot N \cdot \frac{1}{R}$
f_{Δ}	frequency spacing	$f_{\Delta} = \frac{R}{M \cdot N}$
B_{branch}	branch bandwidth	$B_{branch} = \frac{R}{N}$
f_m	tone frequency after upconversion	$f_m = f_0 + f_1 + f_2 \pm m \cdot f_{\Delta}$

Table 3.1: STFT Pulsegenerator Parameters

3.2.3 Interface

The STFT pulse generator driver Application Programming Interface (API) exposes the gateway configuration in convenient kernel functions. This section will give a functional overview over the API functions and their usage. For the exact parameter definitions see API documentationA.

Each FFT in the pulse generator can be loaded, cleared, started and configured individually. The `send_full_coef()` function can be used to send a full 1024 point coefficient vector to a specific FFT core. `stage_coef_adr()` and `stage_coef_data()` will stage FFT address and data on Kasli until a `send_frame()` call transfers them to their destination FFT memory. This allows for up to 13 consecutive coefficients to be written to an offset in a specified FFT core. `start_fft()` will trigger the

3 STFT Pulse Generator

FFT computation in a core. `set_shiftmask()` allows the specification of a scaling schedule for each FFT².

Signal processing parameters can be specified for each STFT branch individually. The API provides `set_duc_frequency()` to set the DUC frequency, `set_duc_phase()` to set the DUC phase and `set_interpolation_rate()` to set the branch interpolation rate. The window FFT and interpolation rate can be configured in a similar manner, being identified as a fourth FFT and interpolator. `set_number_repeats()` allows the window to be repeated a number of times.

`set_pulsesettings()` allows the pulse generator to either:

1. continuously output the accumulated branch signals.
2. continuously output the windowed branch signals, infinitely repeating the window.
3. output the windowed signal a specified number of times when the pulse is triggered.

If the pulse generator is configured to the third option, the `trigger()` function will trigger the emission of a pulse as soon as possible – either as soon as all pending FFT computations are done or as soon as the trigger register in the Phaser FPGA is written. However, because Kasli and Phaser are connected using a frame-based real-time interface, the exact frame timing is crucial for a deterministic latency. Using the `get_frame_timestamp()` function, the exact timing of a frame can be stored in `self.frame_tstamp` and the `trigger()` function can schedule the trigger flag to be sent an exact integer multiple of `self.tframe` later, resulting in deterministic latency.

3.2.4 Practical Considerations

There are a couple of important considerations when configuring the STFT pulse generator. The combination of various DSP components and their inherent dynamics impose constraints on the valid parameters. Since some of the constraints are

²For technical detail refer to 3.3.1

very dynamic, it is not practical to inhibit all erroneous constellations. The user therefore has to be cognizant of the limitations and interactions in the architecture.

Digital circuits are limited in their ability to represent signals. If signal amplitude/frequency fall outside the allowed intervals, overflow/aliasing will occur. In an aliasing condition, spectral content outside the representable bandwidth will appear somewhere within the representable region. If a signal overflows (i.e. a sample is outside the allowed value range), the output will be discontinuous, resulting in strong harmonic distortion. While these effects can be tolerated or even useful if accounted for, interpreting the output of an overflowing/aliasing circuit can be questioning if not taken into account.

Overflow can happen at three places in the STFT pulse generator. The first and most nuanced is within the FFT computation. Since the usual computation of an inverse Discrete Fourier Transform (DFT) involves a $\frac{1}{N}$ factor and the FFT works in several stages, the $\frac{1}{N}$ factor is usually achieved by bitshifting the intermediate signals by one in every stage. The $\frac{1}{2}$ factors in every stage compound to $\frac{1}{N}$ at the end of the FFT. With only a limited number of tones or small tone coefficients, it is usually desirable to scale the output for maximum signal power and hence SNR. This can be achieved by omitting the $\frac{1}{2}$ factor in some FFT stages. However, depending on the combination of coefficients and shiftmask, the signal might overflow at the output or during the computation. While better shifting schedules can be designed for special cases, the following usually yields adequate performance without the risk of overflow:

1. The total scaling should be lower (next power of two) than the sum of the coefficient magnitudes.
2. Scaling must generally happen in the later stages to avoid overflow in the intermediate stages.
3. However, the first two stages can always be scaled (unless more than 256 tones are present).
4. The magnitude of each individual coefficient should not be greater than 1 (i.e. $1 + 1j$ should be avoided).

3 STFT Pulse Generator

The second place overflow can occur is at the adders summing up the branch outputs where it will happen if the sum of the branch outputs does not fit into the 16 bit representation. Lastly, the FIR filters in the interpolators display inevitable overshoot for abrupt signal changes. If the time-domain output of an FFT is such that there is a step change close to the highest or lowest representable amplitude, the signal will overflow³.

Aliasing has to be considered at the two digital upconverters in the architecture. The “baseband” signal after the branch interpolator will occupy a bandwidth proportional to the interpolation rate (see table 3.1) and defined by the FFT coefficients. If spectral content is upconverted to outside the $[-250, 250]$ MHz window representable in a complex 500MSps signal, it will alias. Similarly, if spectral content falls outside the $[-500, 500]$ MHz window representable after the DAC upconverter (f_1), the signal will alias at this point.

Finally, the digital filters in the branch interpolator can only be built with a transition band, attenuating the highest frequencies. For both branch- and DAC interpolator, the transition band starts at 80% of the input Nyquist frequency, meaning frequencies higher than this should not be present at the input. For the branches and window this simply means, that the highest and lowest 10% of the coefficients should be set to zero. Unfortunately the transition band of the DAC interpolator is more difficult to avoid and the user has to make sure that none of the upconverted tones of either branch lie outside the $[-200, 200]$ MHz DAC interpolator passband.

³With interpolation rates greater than 4, the effect will be suppressed by an additional Cascaded Integrator Comb (CIC) filter. It will still produce unintended spectral components.

3.2.5 Specifications

A summary of the specifications is given in table 3.2. Further details can be found in the respective DSP module description.

Module	Parameter	Specification
General	output sample rate	500MSps
	data width	16 bit
FFT	size	1024
	arbitrary single tone SNR/SFDR	>90dB/100dB
	arbitrary 8 tone SNR/SFDR	>70dB/90dB
	arbitrary 128 tone SNR/SFDR	>60dB/70dB
Interpolator	image rejection	>89.5dB
	passband droop	<0.9dB/10%
	input cutoff frequency	$0.8f_N$ (80% Nyquist)
	supported branch interpolation rates	2, 4, 8, 12, ..., 1024
	supported window interpolation rates	2, 4, 8, 12, ..., 8192
Upconverter	frequency resolution	58mHz
	frequency range	[-250, 250]MHz
	SNR/SFDR	>83dB/84dB

Table 3.2: Specifications

3.3 Signal Processing

This section will give a technical description of the main signal processing blocks. As the theory for blocks implemented or used in this thesis is well-established and can be found elsewhere[Org07], the focus is on motivating the utilization, specific design decisions and implementation.

3.3.1 FFT

The design of the FFT core reflects a compromise between performance, resource requirements and complexity:

1. As undesired spectral components can contribute to ion heating or even drive unintended modes in the ion crystal, SNR and Spurious Free Dynamic Range (SFDR) performance need to be adequate.
2. Since the FFT is used to generate samples which are further processed by fixed-point DSP circuits and a floating point FFT is resource inefficient, a fixed-point FFT should be employed.
3. The FFT output should be persistent and accessible in FPGA RAM after computation.
4. An FFT computation should be reasonably quick.
5. Resource requirements should be reasonably low.
6. Due to limited development time, highly sophisticated schemes are unfeasible.

Performance

The ubiquitous FFT algorithm significantly reduces the computational complexity of a DFT by successively applying elementary “butterfly” computations in a divide-and-conquer scheme. The whole FFT is thereby broken up into stages, each of which iterates the butterfly operation over the data. A butterfly computation consists of additions, subtractions and multiplications with “twiddle” factors.

A 1024 point, radix-2, Division In Time (DIT)⁴ algorithm was chosen for the FFT core. This variant has the best SNR performance of possible fixed-point FFT implementations[CN08], requires the least resources and is simple to implement. It is, however, also the slowest algorithm. DIT is favorable over Division In Frequency (DIF) since trivial computations are performed first, which leads to better performance⁵. The number of FFT points is a compromise between number of possible tones per STFT branch and resource requirements (maximizing specific FPGA primitives as discussed later).

The fixed-point arithmetic entails several consequences and particularities in theory of operation. Since the usual inverse Discrete Fourier Transform (DFT)

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{\frac{2\pi}{N} jkn}, \quad (3.1)$$

where x_n is the n 'th time domain sample, X_k the k 'th frequency domain sample and N the size of the inverse DFT, features a $\frac{1}{N}$ factor in the computation, a straight-forward fixed-point inverse DFT would occupy only a small fraction of the output dynamic range.

In order to make use of the full output dynamic range we need to scale the computation of the FFT core. Because a single point of scaling would either require the use of a large intermediate fixed-point representation, or introduce significant error, a distributed scaling scheme is employed. Furthermore, a specific shifting schedule can be supplied for each FFT computation so as to maximize the SNR in various scenarios.

This additional control is desirable because we want scaling to happen as early in the computation as possible, so as to not lose precision in the intermediate rounding. This intermediate rounding is inevitable since each input sample is subject to several multiplications during the computation. After each multiplication the output has to be rounded to the internal computation width again. For the FFT core this internal computation width is fixed to 18 bit. Therefore each rounding operation

⁴Unfortunately the literature is inconsistent with the DIT and DIF terms. Here DIT will always refer to a division in the input data, agnostic of domain.

⁵Computation errors will be discussed at a later point.

3 STFT Pulse Generator

introduces, on average, 0.25 Least Significant Bit (LSB)⁶ of error, if we assume the error to be uniformly distributed. This error is propagated and potentially exacerbated by downstream scaling operations. A more sophisticated error analysis for a fixed-point radix-2 DIT FFT can be found in [Ma97].

An inverse DFT can be seen as a sum of complex sinusoids. Depending on the spectral distribution, the sinusoids can “constructively” accumulate in some time-domain samples or cancel out. For a general vector of frequency coefficients it has to be assumed that the maximum time-domain sample x_{max} will be

$$x_{max} = \frac{1}{N} \sum_{k=0}^{N-1} X_k. \quad (3.2)$$

This has to be considered when choosing a scaling schedule for the FFT core as too much scaling will lead to arithmetic overflows. The scaling schedule should always be optimized so that the output uses the full dynamic range, but not more.

Figure 3.3 shows the performance of the FFT core for a number of different scenarios. 3.3a displays the naive fixed-point FFT performance for a single full-range input coefficient. The noise exhibits some structure as the errors are distributed in the spectrum during the computation. 3.3b shows an FFT with the same input, but with scaling in every stage. The output tone is now also at full-scale and the noise shows a different pattern. 3.3c and 3.3d depict a full-scale output 8-tone FFT with different scaling schedules. 3.3d shows significantly improved SNR and SFDR, since the early scaling is favorable. 3.3e portrays an optimized scaling scheme for an arbitrary set of 128 input coefficients. An interesting special case is detailed in 3.3f, where 8 tones are distributed in a bit-reversed ascending order. Bit-reversing refers to the index written out in binary and then interpreted right-to-left to yield the bit-reversed index. This leads to equidistant tones in natural order, also called a frequency comb. Here the FFT algorithm happens as such that the samples are only shuffled around and never multiplied with non-trivial twiddle factors. The resulting time-domain pulse comb is perfectly expressed in fixed-point representation.

⁶LSB refers to the signal value represented by a single bit.

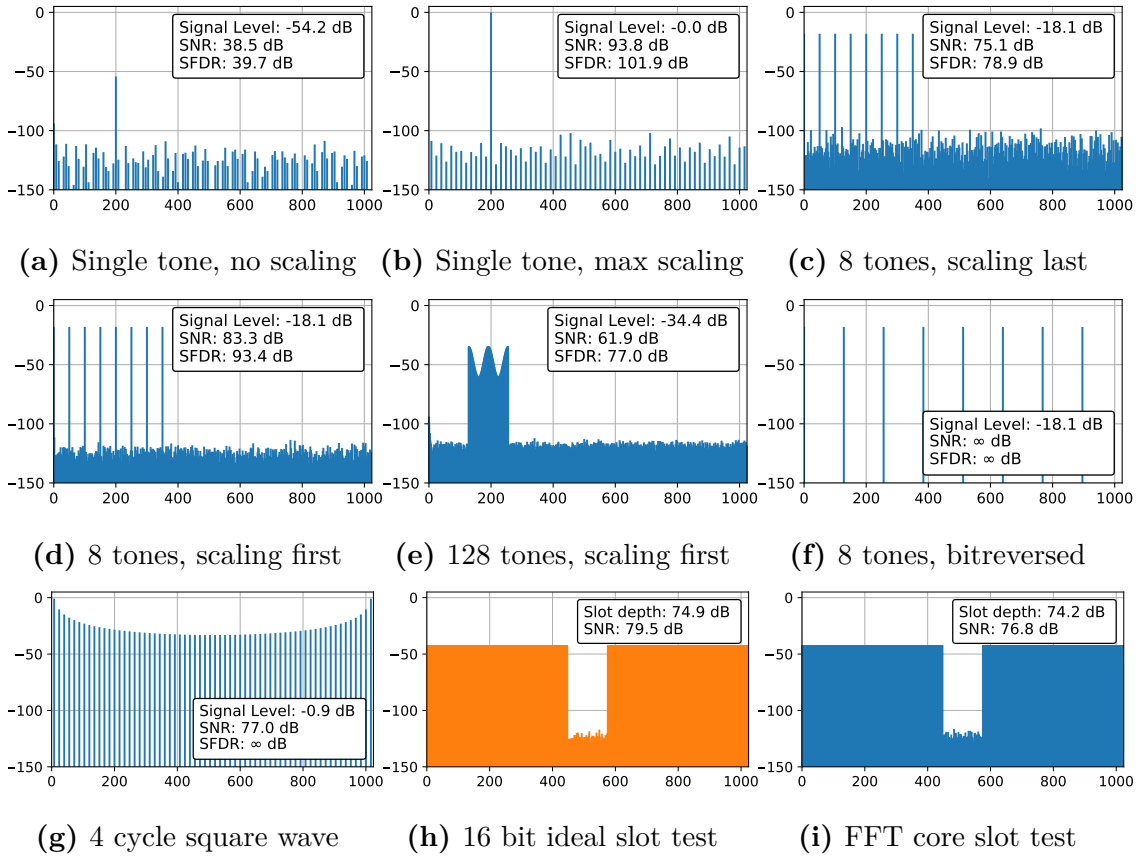


Figure 3.3: 9 FFT computation scenarios for the developed core. For each scenario an ideal 1024 point FFT of the time-domain output is depicted. The y-axis is the spectral magnitude in dB relative to a Full-Scale sinusoid (dBFS) and the x-axis are the FFT bins. For each scenario additional information is given. Signal level refers to the magnitude of the strongest signal bin. The SNR is calculated as follows: Signal power is defined as the ideal signal power output by an ideal FFT (scaled to be equivalent to the core output). The error signal is determined by subtracting the ideal signal from the FFT core output. The noise power is defined to be the power of this error signal, with the SNR being the ratio of the two terms. The SFDR is simply given as the difference of the highest signal and the highest noise bin. Since 3.3h is not computed by the actual FFT core, it is plotted in a different color.

3 STFT Pulse Generator

Another interesting case is demonstrated in 3.3g. The scaling scheme allows for the synthesis of a full-scale time-domain squarewave, even though two full-scale tones at arbitrary positions would lead to overflow. 3.3h and 3.3i show the output of a “slot test”. For a slot test all of the FFT frequency bins are initialized with maximum amplitude, random phase samples. Then a “slot” is cut out by setting a number of consecutive samples to zero. Afterwards the FFT of the remaining “depth” of the slot is compared. 3.3h has been computed with an “ideal” double precision float FFT and then time-domain quantized to 16 bit. 3.3i shows that the output of the FFT core has comparable performance.

Architecture

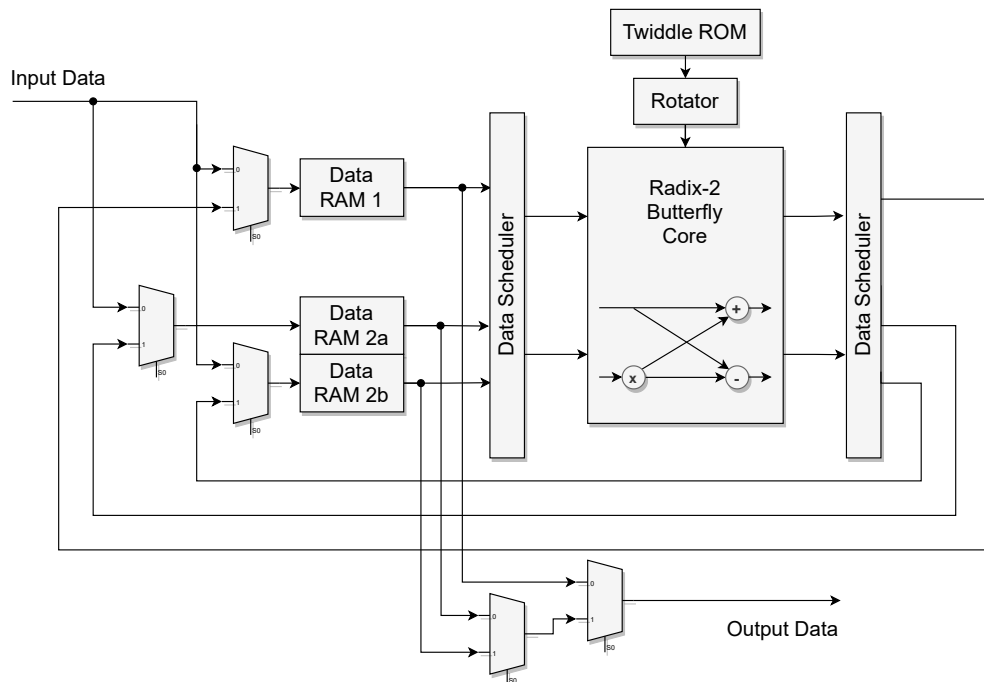


Figure 3.4: FFT Block-Diagram

Since the FFT will be used in the STFT scheme with the output repeated several times, a block-FFT is the natural form of implementation. In a block-FFT the data is always stored in the same location and iterated over to compute the FFT.

At the end of computation, the data is transformed into time-domain and samples can be randomly accessed. The overall block diagram is depicted in Figure 3.4 with the individual blocks explained later in the text.

As we want the computation to happen as fast as possible with reasonable resource overhead, the design features a single, fully pipelined butterfly computation core. This core ingests two input samples a, b and one twiddle factor w and expels two output samples c, d in each clock-cycle. Separating the computation of the output samples

$$c = a + wb \quad (3.3)$$

$$d = -a + wb \quad (3.4)$$

into their real/imaginary components

$$c_r = a_r + b_r w_r - b_i w_i \quad (3.5)$$

$$c_i = a_i + b_r w_i + b_i w_r \quad (3.6)$$

$$d_r = -a_r + b_r w_r - b_i w_i \quad (3.7)$$

$$d_i = -a_i + b_r w_i + b_i w_r \quad (3.8)$$

and then rewriting the complex outputs as

$$c_i = a_i + (b_r + b_i)(w_r + w_i) - b_r w_r - b_i w_i \quad (3.9)$$

$$d_i = -a_i + (b_r + b_i)(w_r + w_i) - b_r w_r - b_i w_i \quad (3.10)$$

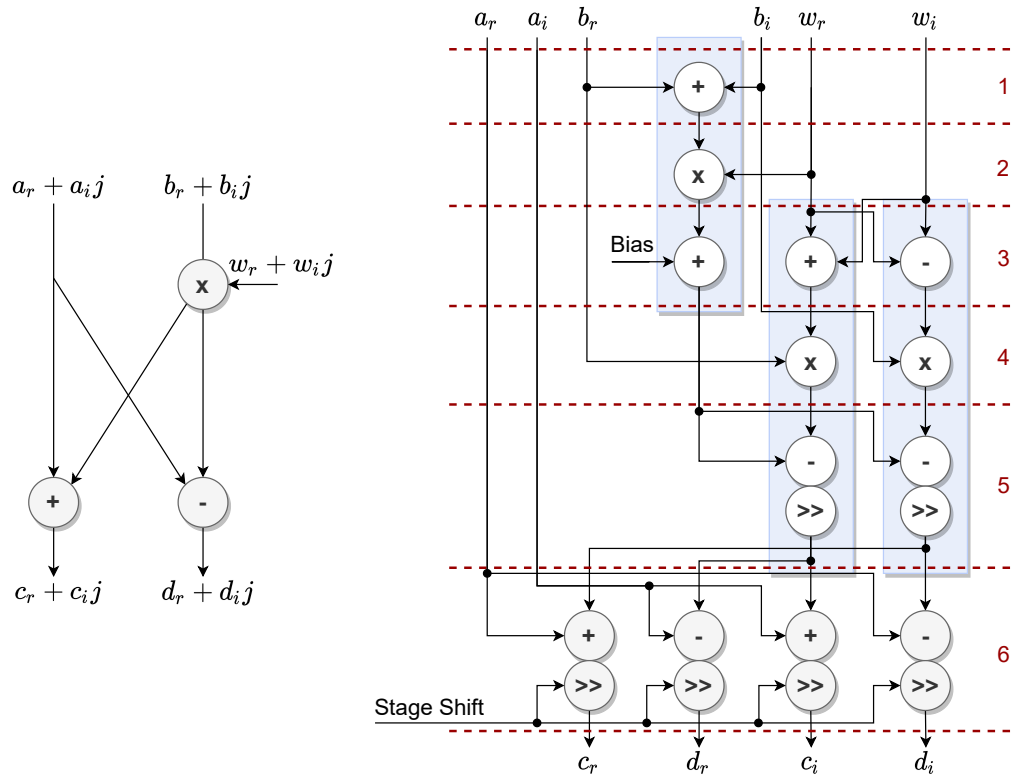
yields a set of equations with 3 real multiplications in total⁷.

Allocating the equations to arithmetic primitives in the FPGA yields the datapath shown in Figure 3.5. The full computation can be accomplished using only 3 DSP blocks and 4 fabric adders. To achieve the 250MHz fabric clock, a pipeline register is located after every elementary arithmetic operation. The core thereby uses 6 computational steps plus one step at the input and output to pipeline the memory

⁷Note that the same multiplication is present in several equations and therefore only requires a single instantiation in the circuit.

3 STFT Pulse Generator

access. There are shifting operations in two of the computational steps. The second shift is configurable by an additional shift input and actualizes the stage scaling. When there is no scaling, the output is shifted by one, hence applying a factor of $\frac{1}{2}$. Doing this in every stage leads to the “natural” $\frac{1}{N}$ factor in the inverse DFT computation. If the output should be scaled in the current stage, the output is not shifted which can lead to bit growth on the output.



(a) Butterfly Signal-Flow

(b) Butterfly Datapath

Figure 3.5: Butterfly Signal-Flow and Datapath. The dashed lines in the datapath diagram illustrate register stages in the dataflow. The blue boxes signify the underlying DSP block actualization.

The first shifting operation, at the output of two DSPs, is used in conjunction with the bias to implement rounding. The rounding makes use of the fact that multiplication output width is the sum of the widths of the inputs. By adding a constant equivalent to 0.5 input LSBs to the output, where one input LSB is represented by many bits, the output gains this offset. After shifting behind the output, the

offset is negated by the negative offset introduced by the shifting operation. This procedure actualizes the “round half up” rounding scheme.

The butterfly computation iterates over the data vector once per FFT stage. Since the data is stored in a RAM block in the FPGA, the correct addresses of the inputs and outputs must be computed. Because we want a total of four memory locations to be accessed in one cycle, we need at least two dual-port memories. However, because the order of access changes in every stage, we need a predictive scheme which sorts the outputs of the butterfly core into memory in a way that allows a future butterfly operation to access inputs from different memories.

The matter is further complicated by the fact that the butterfly core has an 8 cycle pipeline delay. This delay leads to memory access congestion at the stage transitions, where input data from the new stage is accessed, but the output data is 8 cycles delayed and therefore still from the last stage. To remedy this problem, the FFT core alternates two separate memories for the second butterfly input and output. There are better schemes which do not require for an extra memory bank by additionally accounting for the pipeline delay and altering the computation order within a stage [Ric+15]. As these schemes come with significantly increased complexity and memory was not a limiting resource, the simpler scheme is employed. The memory access pattern is illustrated in Figure 3.6.

The memory logistics as well as the FFT input and output access are aggregated into the data scheduler circuitry. This module also keeps track of the computation flow and sets the butterfly scaling signal in the relevant FFT stages. The scaling schedule is simply a register in the data scheduler accessible from outside. The scheduler has the additional task of allocating the FFT input data to the memories and bit-reversing the addresses, if configured to do so. The bit-reversing is an integral part of the FFT algorithm. At the output, the scheduler acts as an address decoder, abstracting the multi-memory architecture for downstream modules.

Reflecting the 36 bit architecture of the XILINX RAMB36 primitives, the FFT core uses 36 bit for one complex sample. Therefore, the real and imaginary parts are each 18 bit wide, thus also maximizing one input of the DSP48E1 18x25 multipliers.

3 STFT Pulse Generator

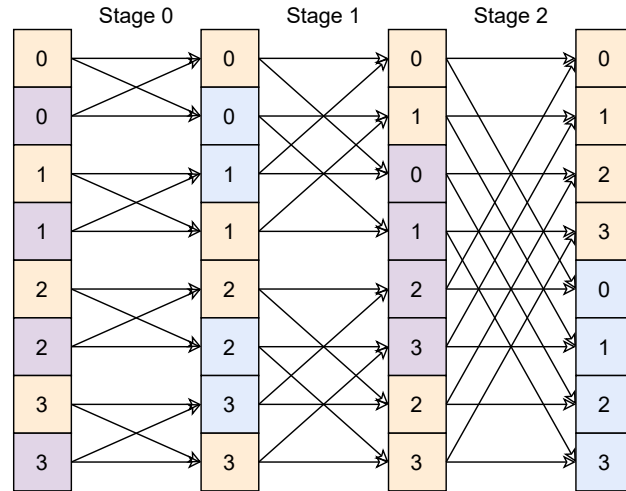


Figure 3.6: Simplified 8 point FFT memory banking scheme. The memory banks are illustrated with different colors and the respective entry address is shown. RAM1 is shown in orange, RAM2a in purple and RAM2b in blue. Using this scheme, neither the butterfly inputs nor the outputs ever access the same memory bank.

The data memories fully occupy the 9 bit address space of a RAMB36 block for a 1024 point FFT. The same data structure is used for the twiddle factors, hence occupying a single RAMB36 block. Additionally the twiddle factors are compressed, exploiting the symmetry of the unit circle. The mapper uses the leading three bits of the twiddle address to rotate the twiddle factors stored at the address defined by the remaining bits to the correct eighth of the circle.

There is a nice synergy between the 16 bit input data, the 18 bit internal precision and the radix-2 DIT scheme. In this FFT variant, all of the twiddle factors in the first two stages point in the direction of the real or imaginary axis and are therefore perfectly represented in fixed-point arithmetic. In conjunction with the fact that we can always scale the first two stages because we can let the inputs grow to 18 bit, we avoid rounding errors in the first two stages. As stated before, errors in the beginning of the computation are the most critical.

3.3.2 Interpolator

The development of the interpolators was driven by the following design considerations:

1. For the same reasons as in the FFT, undesired spectral components are to be avoided and therefore interpolation performance has to be adequate.
2. Adaptive tone-spacing of an STFT branch requires a changeable interpolation rate R without FPGA reconfiguration.
3. As a dispersion of the pulse waveform could change the phase-space trajectory described in section 1.4, a constant group delay is required.
4. The interpolator has to compute two output samples in one clock-cycle.
5. Since there are several interpolators in the design, DSP and fabric resource requirements matter.

Performance

Finite Impulse Response (FIR) Half-Band Filters (HBFs) are efficient linear phase filters. By imposing the constraint

$$H(e^{j\Omega}) + H(e^{j\Omega-\pi}) = 1, \quad (3.11)$$

where H is the frequency response and Ω represents the normalized angular frequency in radians per sample, and

$$N = (4 \cdot n) - 1, \quad (3.12)$$

where N is the number of filter taps and n is a natural number, the frequency response is centered around $\Omega/2$. As a result, every second coefficient, except for the center tap becomes zero. Additionally, the symmetrical impulse response halves the number of multiplications and makes use of DSP pre-adders. A polyphase decomposition of a $R = 2$ HBF interpolator leads to the aggregation of the non-zero

3 STFT Pulse Generator

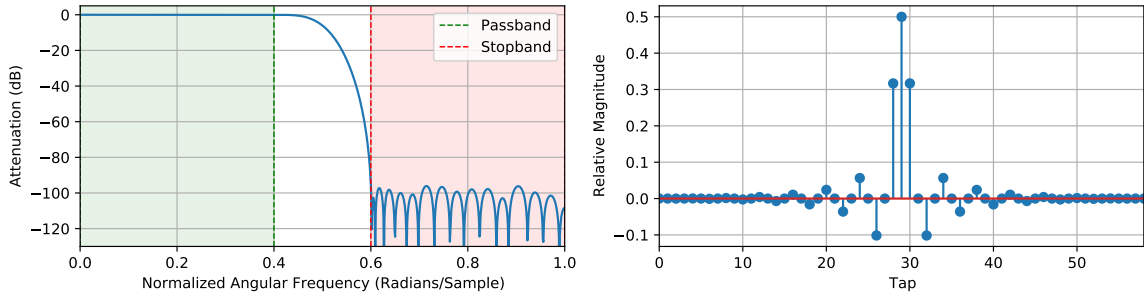
multiplications in one polyphase path. The other polyphase part reduces to a unity-gain delay [Göc11].

A Cascaded Integrator Comb (CIC) filter is a special case of a multiplier-less FIR filter. As the name suggests, instead of delaying and multiplying the input, it constitutes a cascade of integrator and comb stages. While this arrangement makes for a very resource efficient implementation, the filter has a limited range of possible frequency responses. However, as no new filter coefficients have to be provided, the response can be adapted easily during operation.

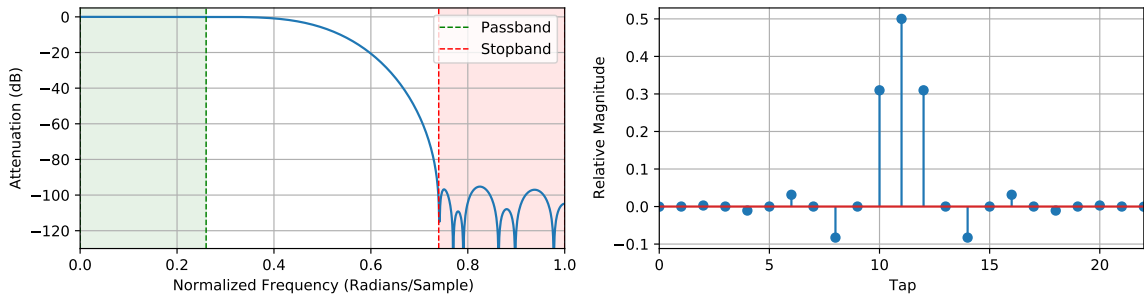
Since a monolithic filter for the whole interpolator would not be very flexible or efficient, a variable filter cascade is employed. At $R = 2$ the interpolator uses a single, sharp HBF with a transition band from $\omega_p = 0.4\Omega$ to $\omega_s = 0.6\Omega$, with ω_p being the passband frequency and ω_s the stopband frequency. The minimum HBF size for an image rejection of higher than 90dB was found to be 59 taps using the Parks–McClellan/Remez algorithm. The design parameters were chosen as a compromise between performance and size. Due to the high stopband attenuation and the symmetrical nature of the frequency response 3.11, passband ripple is not a concern. A zero stuffer inserts a zero between each sample and the filter input⁸. The frequency magnitude and impulse response is shown in Figure 3.7a.

The $R = 4$ interpolator uses the $R = 2$ interpolator described above with a second $R = 2$ stage. The second stage has the same structure as the first. However, the design constraints for the second HBF are relaxed in comparison to the first. Since the first stage already suppresses images above $\frac{1}{2}\Omega_1$, with Ω_1 being the normalized angular frequency of the first HBF, there can also be no significant aliases between $\frac{1}{4}\Omega_2$ and $\frac{3}{4}\Omega_2$, with Ω_2 being the normalized angular frequency of the second HBF. As displayed in Figure 3.7b, a HBF with 23 taps is sufficient for an image rejection of >90dB in the stopband.

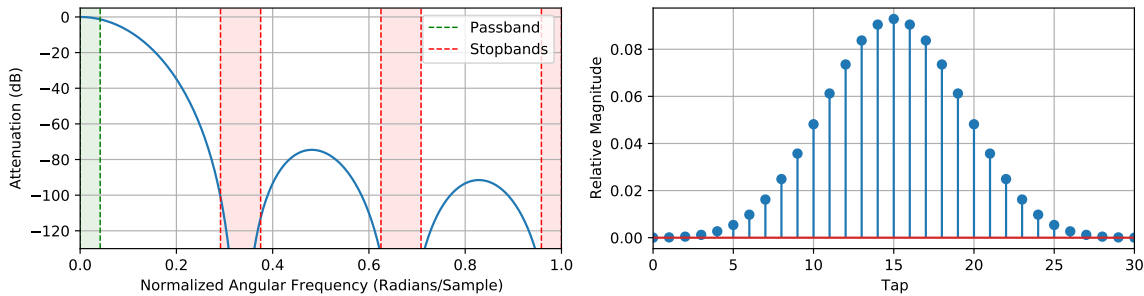
⁸In the actual implementation this zero stuffer is abstracted away. See 3.3.2.



(a) HBF1 - 59 Taps



(b) HBF2 - 23 Taps



(c) CIC6 - 31 Taps

Figure 3.7: Frequency magnitude (left) and impulse response (right) for the three interpolator filters. Relevant frequency bands are highlighted in each frequency response. The CIC response is plotted with an exemplary rate-change of $R_{CIC} = 6$.

3 STFT Pulse Generator

For interpolation rates greater than four, the interpolator constitutes the two HBF stages and a third CIC stage. Because the second HBF stage has halved the passband again, the design constraints are further relaxed. As the CIC frequency magnitude response

$$|H(e^{j\Omega})| = \left| \frac{\sin(\Omega D/2)}{\sin(\Omega/2)} \right|^M, \quad (3.13)$$

where M and D are the CIC order and differential delay of the comb stages, has troughs around

$$T_n = \frac{n}{D}, \quad (3.14)$$

with n being a natural number, we prefer the passband to alias into those troughs. This is given when

$$R_{CIC} = D. \quad (3.15)$$

An example for $R_{CIC} = 6$ is depicted in Figure 3.7c.

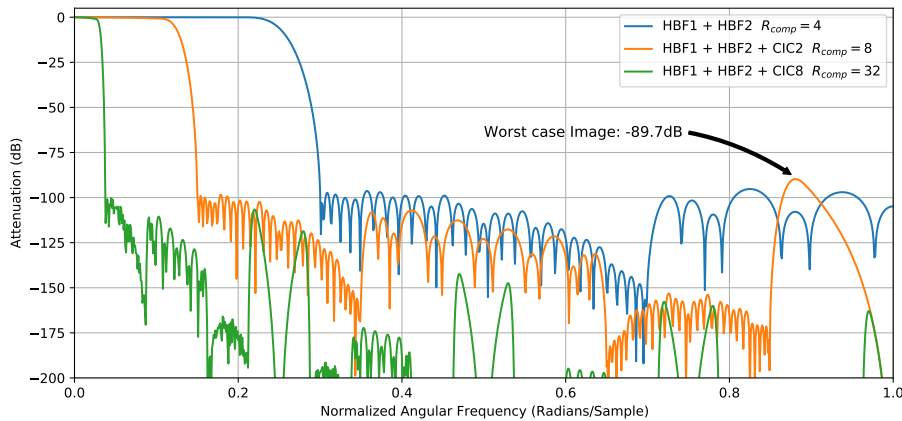


Figure 3.8: Composite frequency responses for various interpolator configurations. The worst case image for $R_{comp} = 8$ is highlighted. As R_{comp} increases, the images are suppressed further.

As we need the CIC stage to realize rate-changes of 2 to 1024, the CIC has to provide $>90\text{dB}$ image rejection for all configurations. It was found that a 6th order $M = 6$ CIC at $R = D = 2$ is just barely short of this performance. Nonetheless,

89.7dB of image rejection was deemed sufficient. As shown in Figure 3.8, the aliases are attenuated further at higher rate changes.

Unfortunately, a CIC filter exhibits inevitable drooping of the passband. This droop is exacerbated as the rate-change and thus differential comb delay increases. For the interpolator, this effect is mitigated by the fact that the passband is compressed by the two leading HBF stages. The droop for a selection of rate-changes is plotted in Figure 3.9. As remaining droop can easily be accounted for in the FFT coefficients, a worst-case of -0.86dB was deemed tolerable.

Another downside of the cascaded interpolator architecture is that the rate change can not take any value. R_{CIC} can be any natural number greater than 1. However, as the composite rate-change is multiplicative

$$R_{comp} = R_{HBF1} \cdot R_{HBF2} \cdot R_{CIC}, \quad (3.16)$$

this entails a rate-change granularity of 4 for the whole interpolator at $R > 2$.

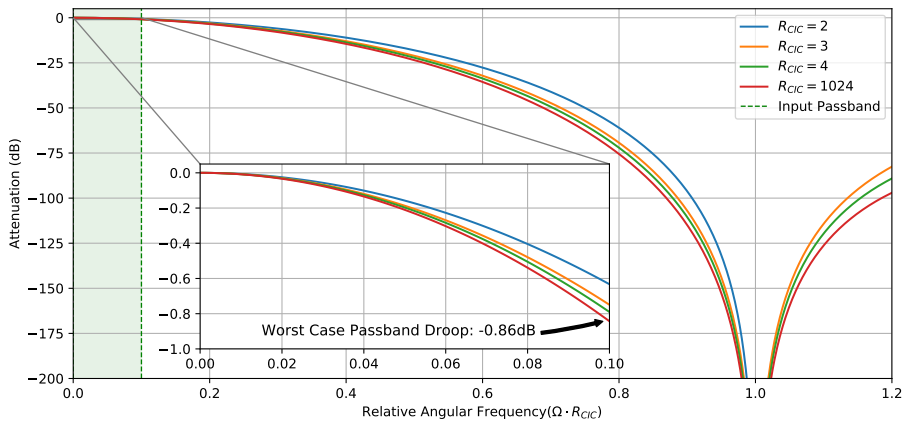


Figure 3.9: CIC passband droop for various rate-changes R_{CIC} . As the normalized angular frequency is different in every case, the attenuation is plotted over a frequency relative to R_{CIC} . The relative passband width is constant for all rates.

Architecture

As the interpolator must be synthesized with the necessary performance⁹, further optimizations in the implementation are necessary. Though the first stages often run at a sample rate R much slower than the FPGA fabric clock $R_{CLK} = 250\text{MHz}$, every stage can potentially be the last and therefore have to serve the DAC at $R_{DAC} = 500\text{MHz}$. Consequently, a 2x super-sampled architecture is necessary for every filter. Super-sampled refers to the fact that the circuit processes more than one sample per clock-cycle.

For the HBFs, we can exploit the multirate decomposition for supersampling. Since every second output sample is just a delayed version of the input and the other polyphase path runs at the input sample rate, we can run them both at R_{CLK} to achieve a throughput of $2R_{CLK}$. The nontrivial path is fed with one input sample per cycle and no stuffed zeros, leading to a standard filter.

To stay within the 240 DSP limit of the FPGA, a custom, DSP multiplexed architecture with several optimizations is employed for the HBFs. First it is realized, that the number of operations per cycle stays roughly constant whether just HBF1 or HBF1 and HBF2 are engaged. This is because HBF2 is approximately half the size of HBF1 and HBF1 always operates at $R_{HBF1} = R_{HBF2}/2$. Hence, it is possible to time-multiplex the resources of HBF1 when HBF2 is engaged, freeing up the rest of the DSP blocks for HBF2 computation.

This results in the multiplexed DSP architecture illustrated in Figure 3.10. The XILINX DSP48E1 DSP circuitry¹⁰ is shown in blue with all of the internal registers and arithmetic enabled. The d delayed filter inputs $x_{[n-d]}^A$ for filter A and $x_{[n-d]}^B$ for filter B are multiplexed into the pre-adder inputs, while the b_d^A (filter A) and b_d^B (filter B) filter coefficients are multiplexed into the multiplier input. When the multiplexing signal h is true, the DSP computes one output of one filter tap

$$p_{d+1} = ((x_{[n-d_1]}^A + x_{[n-d_2]}^A) \cdot b_d^A) + p_d \quad (3.17)$$

in one clock-cycle for filter A. When h is false, the t signal alternates between

⁹Performance in the architectural context refers to the maximum clock rate for the gateware.

¹⁰See XILINX DSP Usage Guide[18] for details

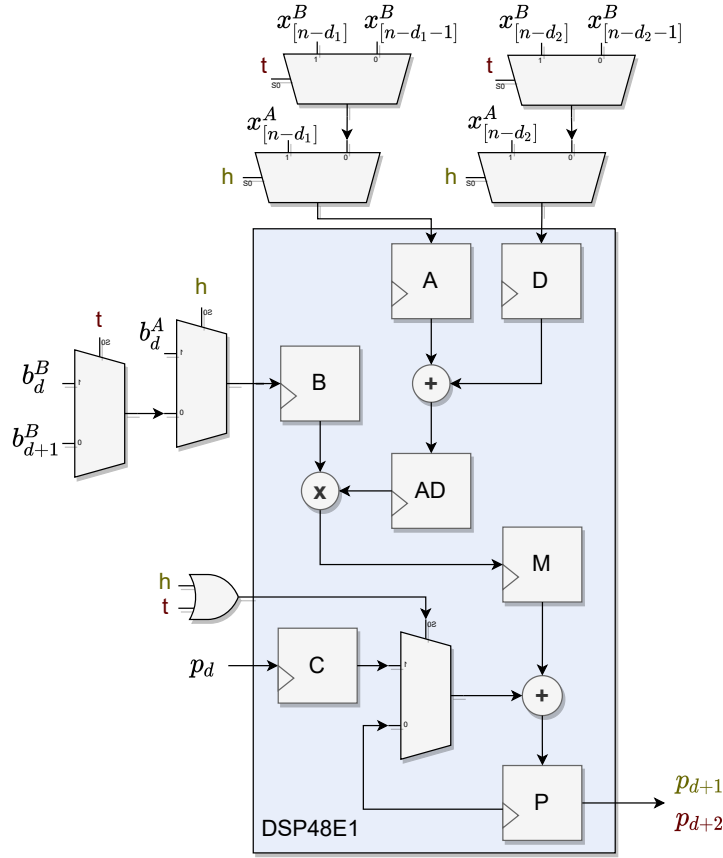


Figure 3.10: Signal flow of the multiplexed DSP architecture. Depending on the control signals different filters are realized.

true and false with a two cycle period. This leads to the computation of one accumulated output of two filter taps

$$p_{d+2} = ((x_{[n-d_1]}^B + x_{[n-d_2]}^B) \cdot b_d^B) + ((x_{[n-d_1-1]}^B + x_{[n-d_2-1]}^B) \cdot b_{d+1}^B) + p_d \quad (3.18)$$

every second cycle.

With all of the optimizations, HBF1 requires 15 multiplications to compute two new output samples and HBF2 requires 7. Therefore, the whole DSP cascade consists of 15 multiplexed DSP blocks. When only HBF1 is engaged, h is true and filter A takes the role of HBF1 for all DSPs. If both HBF1 and HBF2 are engaged,

3 STFT Pulse Generator

h is false for the first 8 DSPs and filter B now becomes HBF1. In the latter 7 DSPs h is still true with filter A now actualizing HBF2.

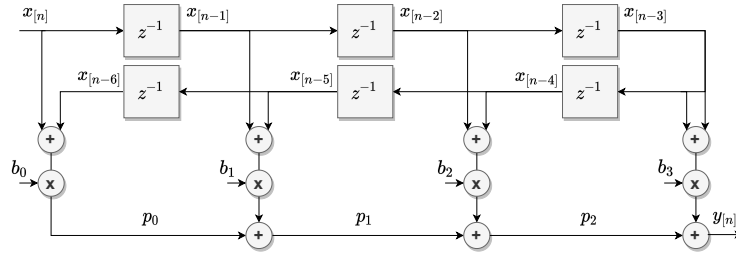
There are a few additional adjustments necessary to make the architecture execute the desired function. First, the trivial polyphase sample from HBF1 has to be piped into the input of HBF2 with the correct delay. Second, a zero multiplication has to be injected in the last time-multiplexed DSP because HBF1 requires an odd amount of multiplications. Third, as filter rounding for the DSPs is performed by piping an offset value into the C input of the first DSP block, this offset must be applied every second cycle for filter B.

Using this arrangement, the second HBF can be implemented at zero DSP cost. The downside is that the multiplexers use more fabric logic, though this was not the limiting resource for the STFT pulse generator.

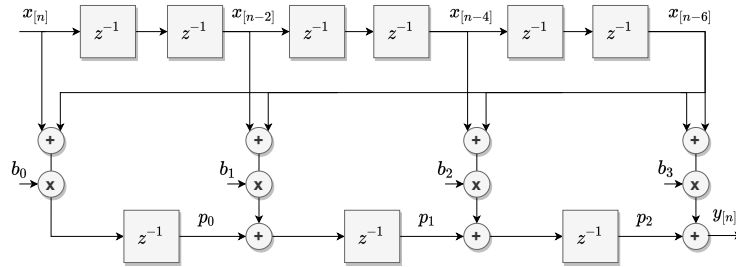
In order to make the interpolator meet the $R_{CLK} = 250\text{MHz}$ timing requirements, pipelining has to be used between each computation and DSP block. During development it was found that the C register of the DSPs was not necessary to meet timing in a design containing only one interpolator. Yet, with the full STFT pulse generator and additional circuits, it was found that the C register is required. As the C register actualizes a unit delay between the postadders in the DSP chain, the filter topology had to be adapted. Figure 3.11 shows three iterations of HBF topologies with the last one being utilized in the final interpolator.

For most interpolation rates the CIC stage is last and hence it is required that the CIC operates super-sampled as well. Since we use the CIC for interpolation, only the integrator stages have to run at the output sample rate and thus super-sampled. This is accomplished by operating two separate interpolators in each stage, one of which accumulates two input samples per cycle. Both integrator outputs are passed onto the next stage where the first integrator accumulates the first input and the second integrator accumulates both. Aside from the accumulator register there is one more pipeline register in each stage for pre-adding the two inputs.

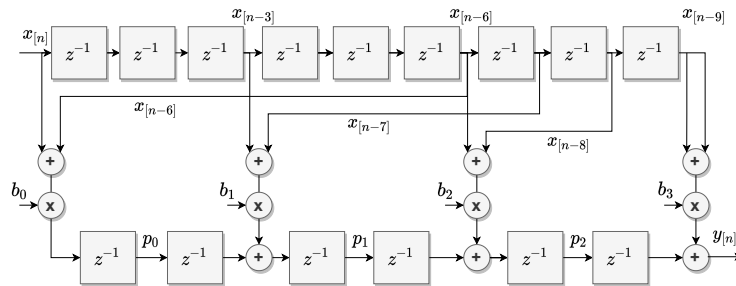
Another consequence of the super-sampled CIC scheme is that the interpolator stage does not ingest samples at evenly spaced intervals for uneven interpolation



(a) Naive HBF architecture. All the multiplier outputs have to be accumulated in one cycle.



(b) HBF architecture 1. A single unit delay in the adder cascade.



(c) HBF architecture 2. Two unit delays between each adder. The pre-adder inputs have to be restructured accordingly.

Figure 3.11: Three iterations of the HBF topology.

rates. For $R_{CIC} = 3$ the core ingests on average 0.66 samples per cycle and expels exactly two. In other words, the stage needs to wait on average 0.33 cycles between every input sample. To account for this, the stage alternates the waiting periods between the closest natural numbers so that the average rate is correct. As a consequence, all of the differentiators have to be run at this staggered clock. Furthermore, additional logic is necessary to ensure the correct output stalling behavior in the upstream HBFs.

3 STFT Pulse Generator

Owing to the high $M = 6$ filter order, the integrator accumulator widths become very large for the latter stages. Since the register bit width is given by

$$W_{reg} = W_{in} + ((M - 1) \cdot \log_2(R)) \quad (3.19)$$

with W_{in} being the input bit width, it follows that

$$W_{reg} = 66\text{bit} \quad (3.20)$$

for a maximum rate-change of 1024 with 16-bit inputs. Fortunately, thanks to fast carry logic in the FPGA slices, a 66 bit adder is realizable at $R_{CLK} = 250\text{MHz}$.

Also, since

$$|H(0)| = R^{M-1}, \quad (3.21)$$

the DC gain of the filter becomes very substantial and not a power of two in many cases. It can thus not be compensated via simple bitshifting. To overcome this problem the CIC uses a Look-Up Table (LUT) with compensation values in a custom format. There are 11 bit of linear, fractional, fixed-point gain which is applied at the input of the CIC and 7 bit of bitshifting (i.e. power of two) gain, which is applied at the output. This compromise uses just a single DSP at the input, one RAMB18 block and some logic for the variable bitshifting at the output. The DC gain is essentially unity using this scheme.

An approach to build a multiplier-free interpolator would be to employ a “Sum Of Powers Of Two” (SOPOT) architecture similar to [CY02]. Here the hardware multipliers are replaced with clever bitshifting and addition to make for very efficient, though not very flexible, filter tap computation. The restricted choice of filter coefficients could lead to a minimal increase in filter size, as the zeros of the transfer function do not land exactly on the unit circle. Since the multiplications of the STFT pulse generator fit into the 240 DSPs of the XC7A100T FPGA, this design effort was not required. However, a fourth STFT branch would already require adaptations of the interpolators.

3.3.3 Upconverter

Since the DUC was not a custom development for the STFT pulse generator, this section only gives a brief overview of the implementation.

Performance

The DUC consists of a Numerically Controlled Oscillator (NCO), a Direct Digital Synthesis (DDS) and a complex multiplication. An NCO is just a phase accumulator with Frequency Tuning Word (FTW) and Phase Offset Word (POW) inputs, which generates a monotonically increasing phase which wraps around at 2π . The DDS generates a complex sinusoid signal from the NCO phase, which is then modulated with the baseband by the multiplier.

Synthesis of the carrier is not trivial since a complex, high resolution sinusoid either needs significant memory or computational resources if implemented in a naïve way. While computational methods such as the well-known CORDIC algorithm often require neither the complex multiplier nor other multiplications, their implementation can be very resource demanding, especially at high clock frequencies.

The DUC synthesizes the carrier using a hybrid LUT/computational scheme. It takes an 18 bit phase input and produces 16 bit real/imaginary outputs. One octant of a low resolution, 9 bit phase, 15 bit real/imaginary circle is stored in a LUT. Additionally, one octant of a 9 bit phase, 3 bit real/imaginary scaled derivative of the circle is stored. The 6 LSB of the phase input are then used to linearly interpolate between the coarse LUT samples using the derivative stored in the additional LUT. The interpolated sample is mapped into the correct octant using the remaining 3 Most Significant Bit (MSB) of the phase input.

To gain a higher frequency resolution, the NCO operates at a phase width of 32 bit. The phase is then truncated to 18 bit at the DDS input. This architecture can synthesize a $>83\text{dB}$ SNR, $>84\text{dB}$ SFDR carrier for every possible output frequency. Dithering of the truncated DDS input could smooth out spurs from phase truncation and correlated computation errors and therefore improve the worst

3 STFT Pulse Generator

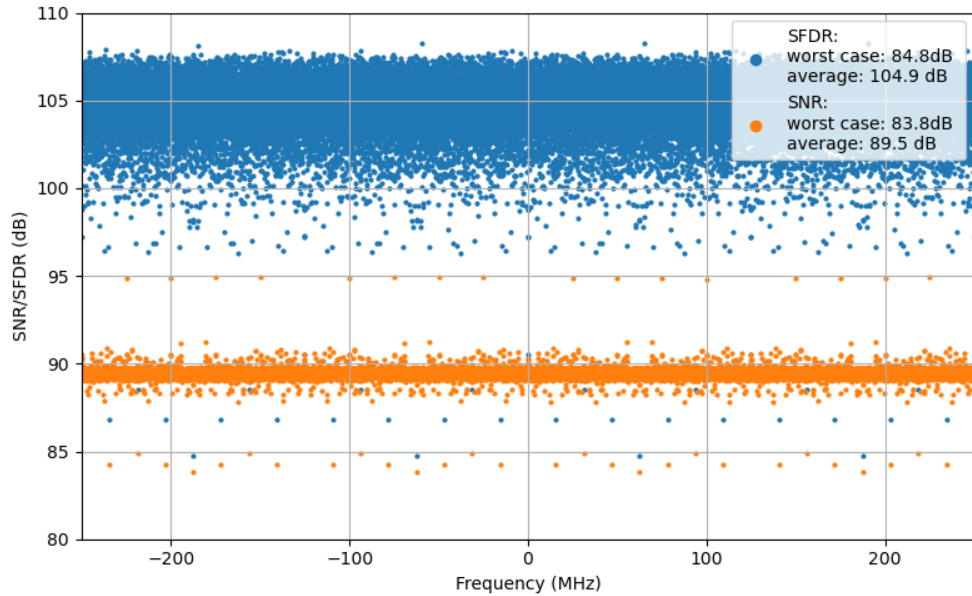


Figure 3.12: Scatter plot of 10^5 equidistant output frequencies of the DDS with a 500MSps sample rate. Evaluating the output SNR and SFDR at more points could lead to slightly lower values for their respective worst cases.

case SFDR at the cost of impaired average values for SNR and SFDR. However, the achieved performance was deemed acceptable without dithering. SNR and SFDR at various output frequencies are depicted in Figure 3.12.

Architecture

The LUTs combined are 36 bit wide and have a 9 bit address space. Hence, they fit exactly into one XILINX RAMB36 primitive. The 3x6 multiplication for the interpolation is implemented in fabric logic, just like the 16 bit adder to add it to the coarse LUT sample. The octant mapping reduces to multiplexing. The architecture is illustrated in Figure 3.13. Not shown are pipeline registers between all computations, memories and multiplexers.

To achieve the 2x supersampling necessary for the STFT pulse generator, two independent DDS make use of the same LUT RAM with two separate read ports.

The NCO generates two consecutive phase samples in one clock cycle. Similarly, two separate complex multipliers modulate the two baseband samples from the interpolator onto the two carrier samples. The multipliers are essentially the same as the butterfly core from 3.3.1, just without the last adder stage. They consist of three DSP blocks each and introduce little rounding error, as discussed in 3.3.1.

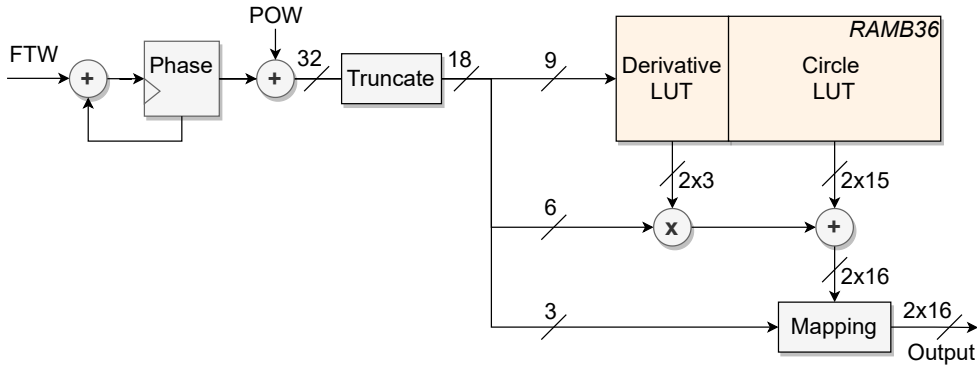


Figure 3.13: Simplified DDS Block-Diagram. Relevant signal widths are marked.

3.4 ARTIQ integration

As illustrated in Figure 3.14, various infrastructure modules are necessary on Kasli and Phaser to drive the STFT pulse generator.

Starting from the actual DSP blocks, there are a number of Control/Status Registers (CSRs), locally storing configuration parameters like branch interpolation rate, branch DUC frequency, master pulse settings, etc. The CSRs are connected using a simple CSR bus and accessed by a bus master. The bus master uses the CSR address and data transmitted in a header by the Fastlink frame-based interface. This interface connects the Phaser FPGA with the central Kasli FPGA and provides gigabit data rates in the Kasli-Phaser direction with simple CSR readback functionality in the Phaser-Kasli direction. The Serializer-Deserializer (SerDes)

3 STFT Pulse Generator

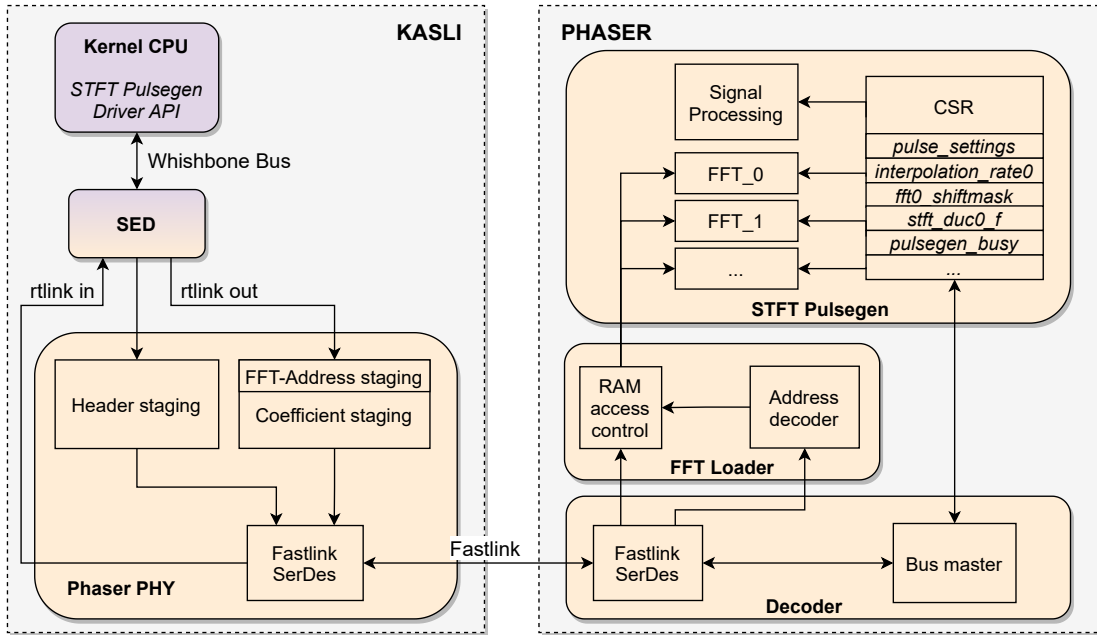


Figure 3.14: Integration Overview. Several gateway (orange) modules on Kasli relay the data from the Kernel CPU (purple) to the Fastlink interface. On the Phaser side, the data is decoded and the specified functions are executed by various gateway modules.

drives the physical Low Voltage Differential Signaling (LVDS) pairs and provides the deserialized data frames to the rest of the gateway.

A data frame consists of various fields and carries different data formats depending on the Phaser configuration. When Phaser is in “classic” mode, a frame transfers 16 complex, 14 bit, real/imaginary data samples in addition to the frame header. If specified by the header type field in the frame header, the frame is interpreted as an FFT frame and takes the data format shown in Figure 3.15. The other header fields contain CSR address and data, as well as a Write Enable (WE) bit. The bus master in the Fastlink decoder on Phaser then writes and reads the respective CSRs. The body of an FFT frame consists of one FFT identifier (ID) field, a base coefficient address and 13 FFT coefficients. The FFT loader uses the FFT ID to address the various FFT cores in the STFT pulse generator. FFT RAM access is managed using the FFT base-address. The 13 frame coefficients are written to

consecutive RAM locations starting at the base address. This allows for coefficients at specific locations to be altered without having to write the whole coefficient RAM while simultaneously maximizing throughput by utilizing the whole frame body.

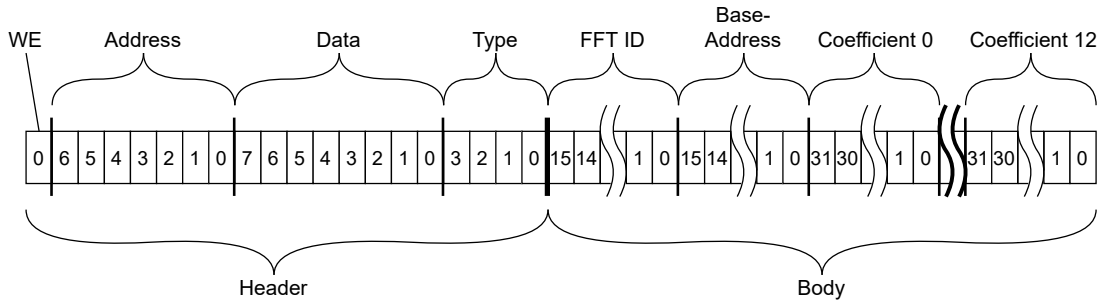


Figure 3.15: Fastlink Frame Dataformat

Inside the Phaser PHY on the Kasli side, a similar SerDes serializes the frames and receives the readback CSR data from Phaser. The frames are assembled in staging areas for the header and body. The staged header is always transmitted onto the next frame. Transmission of the staged FFT data is evoked by setting the FFT frame flag in the Phaser PHY. This causes the PHY to suspend the “classic” data frames and send the staged FFT frame instead, setting the header type field to tell Phaser the frame type.

As mentioned in 2.3, the CPU interacts with the timing-critical hardware via the RTIO system through the Scalable Event Dispatcher (SED). The SED has several rtlink (real-time link) connections to the PHY so it can write the various staging areas and control the “classic” gateway. Using the main SoC wishbone bus, the CPU can schedule the rtlink transactions. The transactions are abstracted into the driver API functions running on the kernel CPU. As described in 3.2.3, the API provides all of the functionality to stage coefficients, set interpolation rates, DUC frequencies, pulse settings, etc.

As also mentioned in 3.2.3, the pulse trigger flag must be set in accordance to the frame timing to achieve deterministic pulse emission. This is because the flag would be staged for a nondeterministic time in the header staging area. To synchronize

the frames with the pulse trigger, a CSR read from Phaser is performed first and the exact frame timing is recorded. The SED can then be configured to schedule the pulse trigger staging transaction an exact integer multiple of the frame-time later. This way, the time from the deterministic SED transaction to the actual pulse emission on Phaser will be deterministic as well. This is possible because the gateware on Kasli and Phaser is deterministic and the Fastlink interface is custom developed to feature deterministic latency.

3.5 Testing

Since all of the developed systems in this work reside in gate- and software, their performance is deterministic which makes a full characterization of the output redundant¹¹. Equivalence of the gateware simulation with the numerical models was continuously checked during development and realistic outputs at various stages of the real FPGA gateware verified.

Checking all of the infrastructure for ARTIQ integration is more challenging, since the interaction of the various subsystems is relatively complex. Deterministic timing was verified on an oscilloscope by emitting a signal se on another Sinara module, which is known to be deterministic, and checking for the STFT pulse to be emitted with a constant latency every time. A repeated mask test can be seen in Figure 3.16 Other functionality like coefficient loading, change of settings or repeated pulse emission was tested under various conditions. While some bugs might still be present in corner-cases which have not been explored during testing, the open development and direct user feedback via Github Issues can help to polish the system. The output of a testpulse with many tones and different interpolation rates is shown in Figure 3.17.

The system was originally intended to be tested with an actual ion trap similar to the one in Figure 1.2 at the PTB Braunschweig. Unfortunately the global Covid-19 pandemic prohibits the realization of an entangling experiment with the STFT pulse generator and the Phaser hardware at the current time.

¹¹This is of course not true for the data converters and analog components which are not part of this work.



Figure 3.16: STFT pulse generator timing test. The oscilloscope is triggered on the yellow TTL pulse and checks if the purple STFT pulse falls within the configured mask. The pulse is configured to be a single frequency, shaped by a Hann window. No timing errors are reported after 1400 tries.

3 STFT Pulse Generator

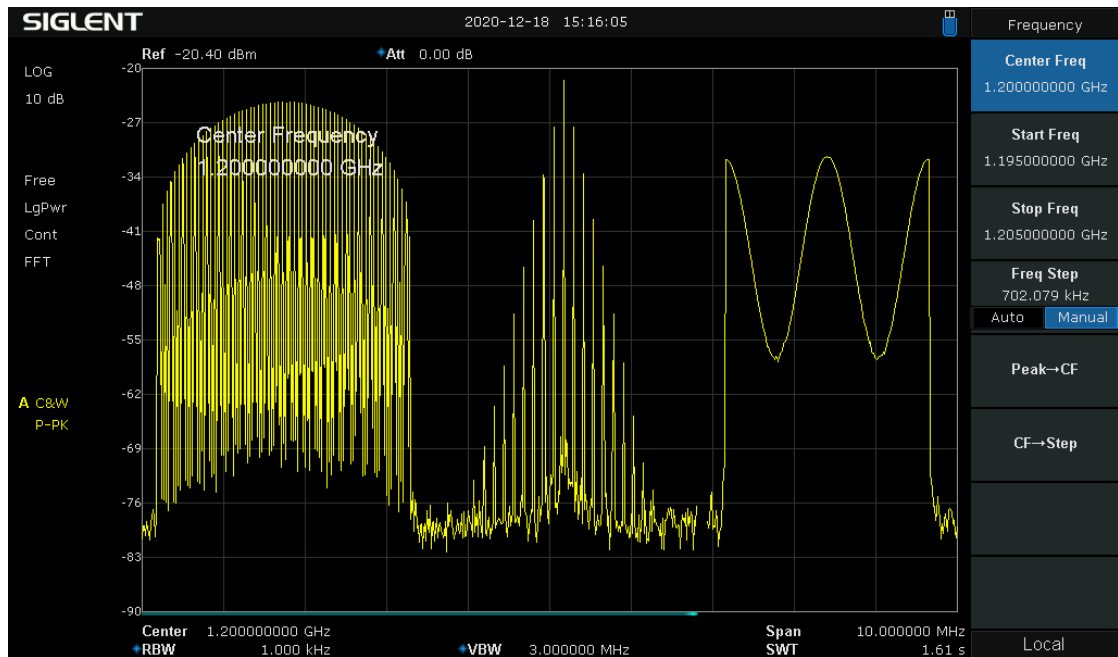


Figure 3.17: STFT pulse generator output on a spectrum analyzer. Three sets of many tones are individually offset from the 200MHz DAC DDS frequency and then mixed onto a 1GHz carrier. The different tone spacings and spectral structure reflect the interpolation rate and FFT coefficients provided via the ARTIQ interface.

4 Conclusion and Outlook

This thesis presented the development and integration of the STFT pulse generator. Several DSP gateway modules were developed and sufficient performance was achieved. Integration into the ARTIQ framework was accomplished and functionality verified. While there is currently no experimental validation in an actual trapped ion experiment, QUARTIQ intends to eventually upstream the system into the main Phaser and ARTIQ distributions, making it available to a growing number of quantum experimentalists using ARTIQ.

Even though the current configuration of the STFT pulse generator is capable of producing very complex pulses, future developments in trapped ion quantum gates might require even more tones or more individual bands. Other schemes like dynamical decoupling [Ber+12] might need continuous control over the qubit state, coherently transitioning into the gatepulse. Another future requirement might be to additionally chirp some of the frequency bands during the pulse. Since the gateway is developed in a very generic way using MiGen, many of these demands can be incorporated with relative ease.

Quantum information processing with trapped ions is still in a very early state. While extreme conditions and control are necessary to maintain quantum coherence, other technologies such as wireless communications have demonstrated exponential improvements with rising engineering efforts. Just like in many current technologies, signal processing will be an integral part of scaling up trapped ion quantum computers in the near future.

A API Documentation

`class artiq.coredevice.phaser.PhaserPulsegen(phaser)` [\[source\]](#)

Phaser Short-Time Fourier-Transform (STFT) Pulse Generator.

Synthesizer for complex, high resolution pulses.

3 sets of 1024 tones with customizable tone-spacing can be placed in a 400MHz band. For Phaser Upconverter, this window can itself be placed somewhere within [-400,400]MHz from the $[0.3,4.8]$ GHz carrier. The pulse can be windowed in time by a narrow-band window specified using 1024 fft parameters and an interpolation rate and emitted with deterministic latency.

Architecture:

Three branches, each containing:

One 1024 point radix2 fft with variable shift schedule allowing for maximized SNR in different conditions.

Two (real/imaginary signal part) variable rate interpolators:

Max rate-change = 1024

Possible rate-changes: 2, 4, 8, 12, 16, 20, ...

Image rejection > 89.5 dB in all conditions

Passband droop < 10%/0.9dB (can be compensated for in fft coefficients)

Cutoff at 80% input nyquist (meaning the highest 10% of positive and negative fft coefficients cannot be used)

One complex upconverter:

Same as in "classic" phaser mode

mHz resolution

SNR > 100dB

Window/shaper path:

Window specified using another 1024 point fft (however less tones can be used eg. only 3 for a hann window)

Another interpolator with the same specs but a maximum rate-change of 8192

Pulse emission can be triggered deterministically with single cycle (4ns) precision

API:

Each fft can be loaded/cleared/(re-)started individually

All interpolator rates can be changed individually

Pulse can be triggered either by sending a start frame from Kasli or as soon as an fft computation is done

Windower/shaper can be bypassed so there is a continuous stft output

Channel 1 can be switched from phaser "classic" to stft pulsegen on the fly (Ch.0 is always Ch.0 of phaser "classic")

Note

There are a number of pitfalls in configuring the pulsegen. Coefficients, shift schedules, interpolation rates and upconverter frequencies have to be such that no overflows or aliasing occur in the signal processing. High distortion in the output is usually a sign of overflow, while flipped/misplaced spectral components arise from aliasing.

check_fft_busy() → `numpy.int32` [\[source\]](#)

Check if in fft computation. UNTESTED

Returns: 1 if busy and 0 if not.

check_pulsegen_busy() → `numpy.int32` [\[source\]](#)

Check if pulsegen is currently emitting a pulse. UNTESTED

Returns: 1 if busy and 0 if not.

clear_full_coef(*id*) [\[source\]](#)

Sets all coefficients on phaser to zero.

Parameters: *id* – fft/branch identifier

clear_staging_area() [\[source\]](#)

Clear the frame staging area in the phy.

get_frame_timestamp() [\[source\]](#)

Performs a Phaser register read and records the exact frame timing in `self.frame_tstamp`. For deterministic timing a pulse has to be triggered an integer multiple of `self.tframe` later.

send_coef(*id, adr, real, imag*) [\[source\]](#)

Send a set (max. nr_coef_per_frame) of consecutive fft coefficients to phaser starting at adress adr.

Parameters: **id** – fft/branch identifier
adr – frame fft mem base address
real – real part coefficient list
imag – imaginary part list

send_frame() [\[source\]](#)

Send out an FFT frame with the staged data next.

send_full_coef(id, real, imag) [\[source\]](#)

Sends a full fft coefficient set to phaser.

Parameters: **id** – fft/branch identifier
real – real coefficient data array
imag – imag coefficient data array

set_duc_cfg(id, clr=0, clr_once=0, select=0) [\[source\]](#)

Set the digital upconverter (DUC) and interpolator configuration.

Parameters: **id** – fft/branch identifier
clr – Keep the phase accumulator cleared (persistent)
clr_once – Clear the phase accumulator for one cycle
select – Select the data to send to the DAC (0: DUC data, 1: test data, other values: reserved)

set_duc_frequency(id, frequency) [\[source\]](#)

Set the DUC frequency in SI units.

Parameters: **id** – fft/branch identifier
frequency – DUC frequency in Hz (passband from -200 MHz to 200 MHz, wrapping around at +- 250 MHz)

set_duc_frequency_mu(id, ftw) [\[source\]](#)

Set the DUC frequency.

Parameters: **id** – fft/branch identifier
ftw – DUC frequency tuning word (32 bit)

set_duc_phase(*id, phase*) [\[source\]](#)

Set the DUC phase in SI units.

Parameters: **id** – fft/branch identifier
phase – DUC phase in turns

set_duc_phase_mu(*id, pow*) [\[source\]](#)

Set the DUC phase offset.

Parameters: **id** – fft/branch identifier
pow – DUC phase offset word (16 bit)

set_interpolation_rate(*id, rate*) [\[source\]](#)

Set the interpolation rate.

Parameters: **id** – fft/branch identifier
rate – interpolation rate (16 bit)

set_nr_repeats(*rep*) [\[source\]](#)

Set the number of fft repeats.

Parameters: **rep** – nr repetitions (16 bit)

set_pulsesettings(*disable_window=1, gated_output=0*) [\[source\]](#)

Write to pulsesettings register.

Parameters: **disable_window** – disables the “shaper” branch
gated_output – enables the gated shaper

set_shiftmask(*id, mask*) [\[source\]](#)

Set the fft shiftmask register.

Parameters: **id** – fft/branch identifier
 mask – shiftmask (16 bit)

stage_coef_adr(*id, adr*) [\[source\]](#)

Write fft memory address of a frame into staging area.

Parameters: **id** – fft/branch identifier
 adr – frame fft mem base address

stage_coef_data(*pos, r, i*) [\[source\]](#)

Write i/q data in staging frame location.

Parameters: **pos** – frame loaction
 r – real data
 i – imag data

start_fft(*id*) [\[source\]](#)

Start the fft computation.

Parameters: **id** – fft/branch identifier

trigger() [\[source\]](#)

Sets the pulsegen trigger flag. A pulse will be emitted as soon as the fft computation is finished.

List of Figures

1.1	Exemplary quantum circuit[yao] that generates a highly entangled state between 4 qubits (before the measurement). After state preparation on the very left, single and two qubit gates are applied, followed by a measurement at the end which reveals the final quantum state. In this circuit the qubits will have a 50/50 chance to all be 1 or all be 0, an outcome not possible with classical logic.	8
1.2	Ion trap fabricated at the “Physikalisch-Technische Bundesanstalt” (PTB) Braunschweig[Bra]. The inset shows arrays of fluorescing beryllium ions.	10
1.3	Exemplary ion mode spectrum[Cal]. The symmetrical structure of the red (left) and blue (right) sideband is due to various vibrational modes in the ion crystal. The modes are named by their movement pattern like Center Of Mass (COM) where the ions all move back and fourth in unison.	12
1.4	PTB Hardware for Mølmer-Sørensen microwave gates.	14
2.1	An image of a quantum laboratory with several instrumentation devices annotated[M-Lb].	15
2.2	A selection of Sinara hardware modules[M-Ld]. In the standard configuration the modules are mounted in a Eurocard 3U chassis.	17
2.3	The ARTIQ architecture spans several devices and abstraction layers. Shown in blue are high level software functions running on the host PC. Purple are low level software functions on the Kasli FPGA. Gateware is shown in orange and hardware is depicted in green. The unannotated arrows represent various interfaces, communication or logical connections.	20
3.1	Phaser[Kas] PCB with the main ICs annotated.	24
3.2	STFT pulse generator Block-Diagram	26

List of Figures

3.3	9 FFT computation scenarios for the developed core. For each scenario an ideal 1024 point FFT of the time-domain output is depicted. The y-axis is the spectral magnitude in dB relative to a Full-Scale sinusoid (dBFS) and the x-axis are the FFT bins. For each scenario additional information is given. Signal level refers to the magnitude of the strongest signal bin. The SNR is calculated as follows: Signal power is defined as the ideal signal power output by an ideal FFT (scaled to be equivalent to the core output). The error signal is determined by subtracting the ideal signal from the FFT core output. The noise power is defined to be the power of this error signal, with the SNR being the ratio of the two terms. The SFDR is simply given as the difference of the highest signal and the highest noise bin. Since 3.3h is not computed by the actual FFT core, it is plotted in a different color.	35
3.4	FFT Block-Diagram	36
3.5	Butterfly Signal-Flow and Datapath. The dashed lines in the datapath diagram illustrate register stages in the dataflow. The blue boxes signify the underlying DSP block actualization.	38
3.6	Simplified 8 point FFT memory banking scheme. The memory banks are illustrated with different colors and the respective entry address is shown. RAM1 is shown in orange, RAM2a in purple and RAM2b in blue. Using this scheme, neither the butterfly inputs nor the outputs ever access the same memory bank.	40
3.7	Frequency magnitude (left) and impulse response (right) for the three interpolator filters. Relevant frequency bands are highlighted in each frequency response. The CIC response is plotted with an exemplary rate-change of $R_{CIC} = 6$	43
3.8	Composite frequency responses for various interpolator configurations. The worst case image for $R_{comp} = 8$ is highlighted. As R_{comp} increases, the images are suppressed further.	44
3.9	CIC passband droop for various rate-changes R_{CIC} . As the normalized angular frequency is different in every case, the attenuation is plotted over a frequency relative to R_{CIC} . The relative passband width is constant for all rates.	45
3.10	Signal flow of the multiplexed DSP architecture. Depending on the control signals different filters are realized.	47
3.11	Three iterations of the HBF topology.	49

3.12	Scatter plot of 10^5 equidistant output frequencies of the DDS with a 500MSps sample rate. Evaluating the output SNR and SFDR at more points could lead to slightly lower values for their respective worst cases.	52
3.13	Simplified DDS Block-Diagram. Relevant signal widths are marked.	53
3.14	Integration Overview. Several gateway (orange) modules on Kasli relay the data from the Kernel CPU (purple) to the Fastlink interface. On the Phaser side, the data is decoded and the specified functions are executed by various gateway modules.	54
3.15	Fastlink Frame Dataformat	55
3.16	STFT pulse generator timing test. The oscilloscope is triggered on the yellow TTL pulse and checks if the purple STFT pulse falls within the configured mask. The pulse is configured to be a single frequency, shaped by a Hann window. No timing errors are reported after 1400 tries.	57
3.17	STFT pulse generator output on a spectrum analyzer. Three sets of many tones are individually offset from the 200MHz DAC DDS frequency and then mixed onto a 1GHz carrier. The different tone spacings and spectral structure reflect the interpolation rate and FFT coefficients provided via the ARTIQ interface.	58

List of Tables

3.1	STFT Pusegenerator Parameters	27
3.2	Specifications	31

Bibliography

- [15] *DAC34H84 Quad-Channel, 16-Bit, 1.25 GSPS Digital-to-Analog Converter (DAC)*. SLAS751D. Texas Instruments, 2015. URL: https://www.ti.com/lit/ds/slas751d/slas751d.pdf?ts=1613054873313&ref_url=https%253A%252F%252Fwww.google.com%252F.
- [16] *TRF372017 Integrated IQ Modulator PLL/VCO*. SLWS224E. Texas Instruments, 2016. URL: https://www.ti.com/lit/ds/symlink/trf372017.pdf?ts=1613027442658&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTRF372017.
- [18] *7 Series DSP48E1 Slice User Guide*. UG479 (v1.10). Xilinx, 2018. URL: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.
- [19] *Koch group - Quantum dynamics & control*. 2019. URL: <https://www.physik.fu-berlin.de/en/einrichtungen/ag/ag-koch/index.html>.
- [20] *7 Series FPGAs Data Sheet: Overview*. v2.6.1. Xilinx, 2020. URL: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.
- [BB20] Charles H Bennett and Gilles Brassard. “Quantum cryptography: Public key distribution and coin tossing.” In: *arXiv preprint arXiv:2003.06557* (2020).
- [Ben+93] Charles H Bennett et al. “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels.” In: *Physical review letters* 70.13 (1993), p. 1895.
- [Ber+12] A Bermudez et al. “Robust trapped-ion quantum logic gates by continuous dynamical decoupling.” In: *Physical Review A* 85.4 (2012), p. 040302.
- [Bow+13] R Bowler et al. “Arbitrary waveform generator for quantum information processing with trapped ions.” In: *Review of Scientific Instruments* 84.3 (2013), p. 033108.

Bibliography

- [Bra] PTB Braunschweig. *Mit Mikrowellen zum Quantencomputer*. URL: www.uni-hannover.de/de/universitaet/aktuelles/presseinformationen/detail/news/mit-mikrowellen-zum-quantencomputer/.
- [Bun20] Bundesfinanzministerium. *Eckpunkte des Konjunkturpaketes*. 2020. URL: https://www.bundesfinanzministerium.de/Content/DE/Standardartikel/Themen/Schlaglichter/Konjunkturpaket/2020-06-03-eckpunktepapier.pdf?__blob=publicationFile&v=12.
- [BV97] Ethan Bernstein and Umesh Vazirani. “Quantum complexity theory.” In: *SIAM Journal on computing* 26.5 (1997), pp. 1411–1473.
- [Cal] Berkeley University of California. *Ion trap quantum processor*. URL: <http://research.physics.berkeley.edu/haeffner/teaching/exp-quant-info/Ion-QC2.pdf>.
- [CN08] Wei-Hsin Chang and Truong Q Nguyen. “On the fixed-point accuracy analysis of FFT algorithms.” In: *IEEE Transactions on Signal Processing* 56.10 (2008), pp. 4673–4682.
- [CY02] SC Chan and KS Yeung. “On the design and multiplier-less realization of digital IF for software radio receivers with prescribed output accuracy.” In: *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No. 02TH8628)*. Vol. 1. IEEE, 2002, pp. 277–280.
- [CZ95] Juan I Cirac and Peter Zoller. “Quantum computations with cold trapped ions.” In: *Physical review letters* 74.20 (1995), p. 4091.
- [Deu85] David Deutsch. “Quantum theory, the Church–Turing principle and the universal quantum computer.” In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117.
- [Fey82] Richard P Feynman. “Simulating physics with computers.” In: *Int. J. Theor. Phys* 21.6/7 (1982).
- [Fey86] Richard P Feynman. “Quantum mechanical computers.” In: *Foundations of physics* 16.6 (1986), pp. 507–531.
- [Gae+16] John P Gaebler et al. “High-fidelity universal gate set for be 9+ ion qubits.” In: *Physical review letters* 117.6 (2016), p. 060505.
- [Gmb21] VDI Technologiezentrum GmbH. *Roadmap Quantencomputing*. 2021. URL: <https://www.quantentechnologien.de/artikel/roadmap-quantencomputing-uebergeben.html>.

- [Göc11] Heinz G Gökler. “Most efficient digital filter structures: The potential of halfband filters in digital signal processing.” In: *Applications of Digital Signal Processing*. InTech, 2011, p. 96.
- [Gro96] Lov K Grover. “A fast quantum mechanical algorithm for database search.” In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.
- [Hah19] M Sc Henning Hahn. “Two-qubit microwave quantum logic gate with 9 Be ions in scalable surface-electrode ion traps.” In: (2019).
- [Kas] Greg Kasprowicz. *Phaser Sinara Hardware Repository*. URL: <https://github.com/sinara-hw/Phaser/wiki>.
- [Ker+19] Florent Kermarrec et al. “LiteX: an open-source SoC builder and library based on Migen Python DSL.” In: *OSDA 2019, colocated with DATE 2019 Design Automation and Test in Europe*. 2019.
- [Lia+17] Sheng-Kai Liao et al. “Satellite-to-ground quantum key distribution.” In: *Nature* 549.7670 (2017), pp. 43–47.
- [M-La] M-Labs. *ARTIQ Documentation*. ARTIQ-5. M-Labs. URL: <https://m-labs.hk/artiq/manual/>.
- [M-Lb] M-Labs. *Experiment Control*. URL: <https://m-labs.hk/experiment-control/artiq/>.
- [M-Lc] M-Labs. *MiGen Documentation*. M-Labs. URL: <https://m-labs.hk/migen/manual/introduction.html>.
- [M-Ld] M-Labs. *Sinara Core*. URL: <https://m-labs.hk/experiment-control/sinara-core/>.
- [Ma97] Yutai Ma. “An accurate error analysis model for fast Fourier transform.” In: *IEEE transactions on signal processing* 45.6 (1997), pp. 1641–1645.
- [Org07] Reinhold Orglmeister. *Signalverarbeitung*. Vorlesungsskript. 2007. URL: <http://www.emsp.tu-berlin.de>.
- [Pre18] John Preskill. “Simulating quantum field theory with a quantum computer.” In: *arXiv preprint arXiv:1811.10085* (2018).
- [Ric+15] Stephen Richardson et al. “Building conflict-free FFT schedules.” In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 62.4 (2015), pp. 1146–1155.
- [Sha+20] Yotam Shapira et al. “Theory of robust multiqubit nonadiabatic gates for trapped ions.” In: *Physical Review A* 101.3 (2020), p. 032330.

Bibliography

- [Sho99] Peter W Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.” In: *SIAM review* 41.2 (1999), pp. 303–332.
- [SM99] Anders Sørensen and Klaus Mølmer. “Quantum computation with ions in thermal motion.” In: *Physical review letters* 82.9 (1999), p. 1971.
- [Wan+17] Ye Wang et al. “Single-qubit quantum memory exceeding ten-minute coherence time.” In: *Nature Photonics* 11.10 (2017), pp. 646–650.
- [yao] yaoquantum. *Prepare Greenberger–Horne–Zeilinger state with Quantum Circuit*. URL: <https://docs.yaoquantum.org/v0.1/tutorial/GHZ/>.
- [Zar+19] G Zarantonello et al. “Robust and resource-efficient microwave near-field entangling be+ 9 gate.” In: *Physical review letters* 123.26 (2019), p. 260503.