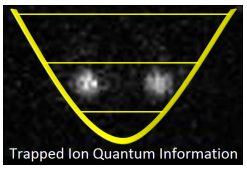
ETH zürich



MSc Quantum Engineering - Master Thesis

RFSoC based Real-Time Control for Trapped Ion Quantum Computing

Roger Serrat i Guevara

Supervised by

Alexander Ferk

Principal investigator

Prof. Jonathan Home

Trapped Ion Quantum Information Group
Swiss Federal Institute of Technology (ETH) Zürich

June 2024

[This page is intentionally left blank.]

Abstract

Real-time control over RF and digital signals plays a key role in trapped ion experiments as well as other quantum computing platforms. Many control solutions employ dedicated components requiring complex, high-speed communication and high integration effort. The AMD RF System-on-Chip (RFSoC) integrates an RF system together with digital processing capabilities on a single chip, making it an ideal candidate to control these experiments.

In this thesis we present a full implementation of a control system for trapped ions on a Gen3 RFSoC. The design is based on the current control system of the TIQI group, called QuENCH. In our design, QuENCH's multi-FPGA architecture has been adapted for a single chip fully integrating the RF generation. The resulting system features 16 RF DACs, capable of generating phase-synchronous RF signals with up to 4 tones each at 4 GSPS, together with 32 digital outputs and 4 digital inputs. The device has been tested in an ion-trap experiment, successfully trapping calcium ions and driving the optical qubit transition to observe Rabi oscillations.

Acknowledgements

I would like to start by thanking Prof. Dr. Jonathan Home for allowing me into the group not once, but twice, as I conducted my semester and master theses here. When I first joined the masters I carried a great curiosity about trapped ion technologies, and I a extremely grateful for the opportunities I have had to work closely to them.

I owe a great deal to my supervisor, Alexander Ferk, who has guided me throughout the entire project. Since day one you have been available to answer even the stupidest of my questions and point me in the right direction. At the same time, you have made me feel trusted and respected, which has kept me going even when my confidence was lowest. I want to thank you especially for the patience you showed during my time away; in such hard times your attitude was reassuring. Even though I don't always understand your humour, it has been a great pleasure working alongside you.

I want extend my gratitude to Martin Stadler, who first introduced me to this project. It is partly because of you that I understood the value that engineers provide in supporting physics research. Also to Bahadir Dönmez, with whom I had the pleasure to share an office, and who altruistically offered to help at any minor inconvenience. I also acknowledge the rest of the TIQI control team – Michael, Michi, Mose and Peter – for following along the process. It is not common for research groups to have a dedicated engineering team, and I couldn't have asked for a better one.

My recognition goes to Kadir Akin for the interest he has taken in this project. I sincerely appreciate our fruitful discussions in gateware development.

I cannot forget the people with whom I have spent my time outside of the studies. In particular, I want to dedicate a special mention to my colleagues Luis, Raquel and Pablo, as well as to my flatmates Victor and Pep. Thank you for making me feel at home in this foreign country.

Finally, I want to thank my parents, Xavi and Montse, without whose unconditional support I would not be writing these lines. Also my brother, Jordi, for always being there to get a laugh out of me. And of course, to my beloved girlfriend, Anna, thank you for staying next to me through all the highs and lows of these past few years. Even from the distance, I feel us closer than ever.

Contents

Chapter 1

Introduction

Trapped ions constitute one of the most promising platforms for quantum computation [1]. Charged particles – individual atoms or even small molecules – are confined in a potential well generated by a static electric field in combination with either a magnetic field or an oscillating electric field. The qubits are then often defined between two long-lived internal states of each ion. Common choices include using the electronic ground state and an excited metastable state, separated by an optical transition, or two Zeeman or hyperfine levels in the ground state, separated by a microwave transition. Regardless of the choice of qubit, many of the relevant operations rely on light-matter interaction, namely cooling, state preparation, coherent control and detection. As a consequence, lasers become an essential part of the system.

Over the last years numerous advancements have been made in an effort to scale up the number of (individually addressable) ions in the traps, as well as to improve the fidelity and reliability of the operations performed on them. A popular solution are segmented surface-electrode trap architectures [2], comprising an increasing amount of electrodes, and employing integrated optics to implement quantum control [3]. However, such improvements on the traps also come with the cost of adding complexity to the auxiliary engineering systems.

Specifically, a large number of electric signals are used to manage all electrical and optical devices [4]. The most relevant problematic comes from real-time components[1] which require precise synchronous control. These include acousto-optic modulators (AOMs) and acousto-optic deflectors (AODs)[2] for modulating laser pulses, radio frequency (RF) and DC trap voltages, digital signals for monitoring and triggering, and readout of peripherals such as cameras and photo-muliplier tubes (PMTs). Additionally, a few other devices must be managed asynchronously, at slower timescales, with less precise timing.

Related technologies such as neutral atoms [5] also employ a similar set of control signals. There, the main trapping mechanism consists of static optical tweezer arrays

[1] Real-time components are designed to perform tasks that must be executed within specified timing constraints.

[2] AOMs and AODs are intrinsically the same device, just operated differently. The distinction is often made to note that AOMs are primarily used to change the intensity and frequency of the laser, while AODs are optimized to affect the beam position.

created by an asynchronously controlled spatial light modulator (SLM) for example. In turn, AODs are used for atom transport in addition to performing operations on the qubits. While some operations in neutral atom systems are similar to trapped ions, the differences in electrical and optical requirements lead to a new range of requirements for the control system.

The currently existing control solution in the Trapped Ion Quantum Information (TIQI) group follows a modular approach [6]. The RF pulses are generated phase-synchronously by dedicated direct digital synthesis (DDS) cards, each with its own field-programmable gate array (FPGA), that are managed by a common main controller together with other inputs and outputs. The workload on the main controller is distributed between an FPGA and a central processing unit (CPU), in an effort to obtain a design that is fast, yet flexible.

Despite its proven practicality and ongoing maintenance, this system still has a few drawbacks. Namely, it only supports up to 32 single-tone channels with stringent timing constraints. A new system has been developed to mitigate these drawbacks, using high speed digital-to-analog converters[3] (DACs) as RF frontend. These DACs require a high-speed interface (JESD204B) to transfer data from an FPGA to the DAC for waveform generation, which introduces considerable complexity in terms of clocking and synchronization. Another option are the RFSoC chips from Xilinx/AMD[4] [7], highly programmable system-on-chip (SoC) devices purposely designed for cutting-edge applications in telecommunications. In particular, the Zynq UltraScale+ RFSoC ZCU216 Evaluation Kit, which includes a Zynq™ Ultrascale+™ RFSoC of 3rd generation, was already used in an earlier project in the group [8] to test synchronous RF generation.

The goal of this thesis is to adapt the core of the control system, namely the software for the main controller and the gateware for RF generation, to the ZCU216. The project is based on the DAC-based version of the control system, called *Quantum Experiment Next-generation Control Hub* (QuENCH) [9]. The project involves adapting the distributed architecture onto one device, (partially) removing the communication layer, as well as improving the software and gateware to be more device agnostic.

This document is structured as follows. Chapter 2 provides an overview of the current control system, focusing on the RF generation. Chapter 3 describes the most relevant features of the target device. Chapter 4 presents the hardware architecture implemented in the FPGA. Chapter 5 lists the changes applied to the software to make it compatible with the new device. Chapter 6 shows some of the tests carried out during development, including a trial in a real ion-trap experiment. Lastly, chapter 7 summarizes the achievements accomplished throughout the thesis and mentions potential further developments.

[3] AD9154

[4] AMD recently acquired Xilinx, the original provider of these chips and related devices.

Chapter 2

Control system

Over the last decade TIQI has implemented a custom solution to control its multiple experiments, originally developed by Vlad Negnevistky and others under the name *Modular Advanced Control of Trapped IONs* (M-ACTION) [6]. Since its creation this system has been continuously maintained and improved, and it has ultimately evolved into the *Quantum Experiment Next-generation Control Hub*, QuENCH for short, designed by Martin Stadler and others [9]. In this chapter we provide a general description of the control system, focusing on the latest iteration, as it constitutes the basis for our adaption to the RFSoC.

2.1 Requirements

A particularity of trapped-ion experiments with respect to other quantum technologies is the concurrent use of electric, magnetic and optical fields. The goal to perform computations with the ions involves having the ability to precisely control said fields, such that arbitrary pulses and waveforms can be generated.

Paul traps – the most widely used way of trapping ions for the purpose of quantum computing – achieve axial confinement by means of a static electric potential, while radial trapping is provided by an oscillating quadrupole electric field. In the simplest configuration this is managed with asynchronous devices, namely a few DACs to select the trap electrodes' voltage and fixed-frequency RF sources. However, ion transport and the excitation of motional modes via modulation of trap electrodes requires synchronous control over the voltages on the trap electrodes.

On the other hand, a variety of optical and microwave fields are used to address specific transitions in the ions. In general, lasers beams would only have to operate at a single frequency, targeted towards a particular transition or Doppler cooling, far Doppler cooling, etc. Still, some applications benefit from the flexibility to detune or pulse them. Especially gate operations require precise control, not just in frequency, but also in phase and amplitude. This control is provided by RF-driven AOMs.

The set of control signals is completed with several digital IOs, including digital outputs for monitoring and triggering purposes, and digital inputs for readout devices (cameras and PMTs). An example of all signals used for a specific experiment can be found in [4], whereas a rough estimation of the general requirements for the current experiments is given in table 2.1.

Table 2.1: Estimated requirements for the control system in TIQI experiments. Based on [6] but updated for the modern setups.

Type	Requirements	Timing	Quantity
RF output	100 kHz – 500 MHz freq, phase coherent	<20 ns resolution <1 ps jitter	10 – 30
Analog output	10 V, ∼1 MHz bandwidth	<100 ns jitter	10 – 40
Digital output	LVTTL (0 – 3.3 V)	<100 ns jitter	5 – 20
Digital input	LVTTL (0 – 3.3 V), count and timestamp	<20 ns resolution, <10 µs readout latency	1 – 4

2.2 Overview

The basic idea behind TIQI's control system is to employ a hybrid FPGA/CPU architecture, where the real-time sequencing is handled on multiple FPGAs, while an ARM CPU running a custom baremetal application coordinates the sequences that are executed on each node. A high-level diagram of the entire control system is depicted in fig 2.1.

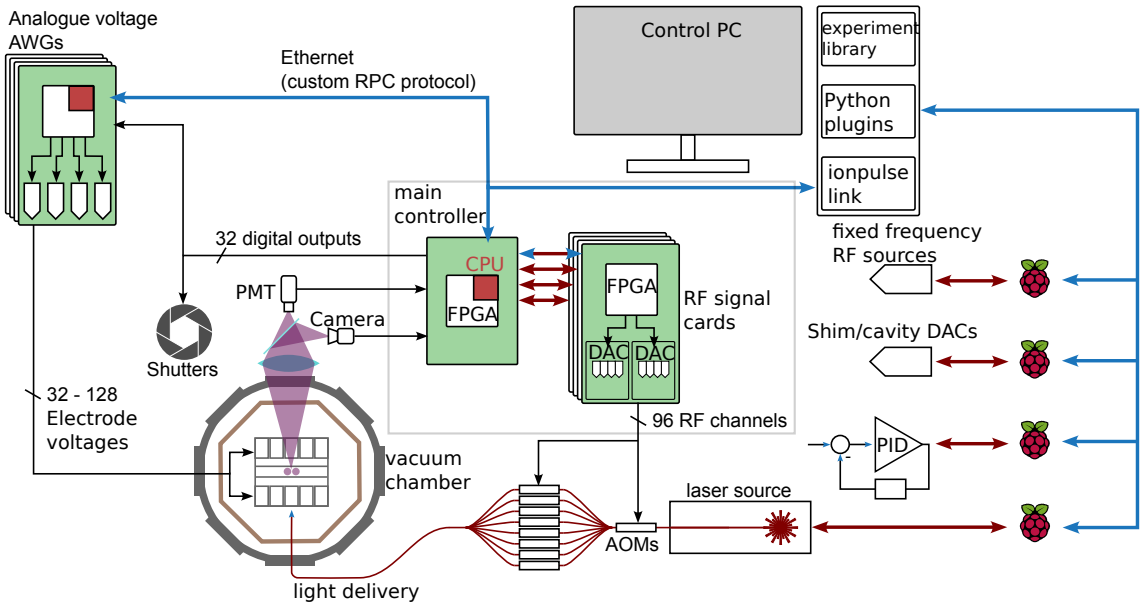


Figure 2.1: Overview of the entire control system. A description of the main components is provided in the main text. In the lower left corner, a cryostat with a surface-electrode trap is shown; the violet dots represent two ions. Image by Martin Stadler.

The central component is the main controller[1], a SoC that comprises both a CPU and an FPGA. The processor runs *Ionpulse*, a C++ baremetal application in which experiments can be defined and parameterized. These are then translated into a sequence of instructions describing the pulses to be generated.

The instructions are sent via a backplane to the dedicated RF cards, where an FPGA-based sequencer decodes and executes the instructions. A similar sequencer

[1] Avnet Zedboard

on the main controller takes care of photon counting and time-tagging (digital inputs), as well as controlling 32 digital outputs.

In the original design, RF generation was done by means of standalone DDS cards that employed a single DDS chip (AD9910) per RF channel. However, they could only generate a single sine wave tone using parameters provided by the custom sequencer. In QuENCH these cards have been replaced by the commercially available AFCK[2] with custom FMC DAC modules featuring a high speed DAC (AD9154). Waveform synthesis is then implemented in the FPGA, currently configured for up to 4 different frequency tones per physical output, in addition to improving some timing limitations on the pulses given by the serial interface of the DDS chip.

The main controller spawns a server that can be accessed from the control PC, connected via Ethernet. There, the *Ionizer* GUI allows users to start and stop experiments, customize their parameters and visualize the results.

Asynchronous devices are controlled independently with their own Raspberry Pis, connected to the control PC through Ethernet. These are not relevant for this project and hence will not be discussed further.

2.3 QuENCH

The core of the system, i.e. the main controller and RF generation, is designed in a modular manner, as depicted in fig. 2.2. The main building blocks for QuENCH are the following:

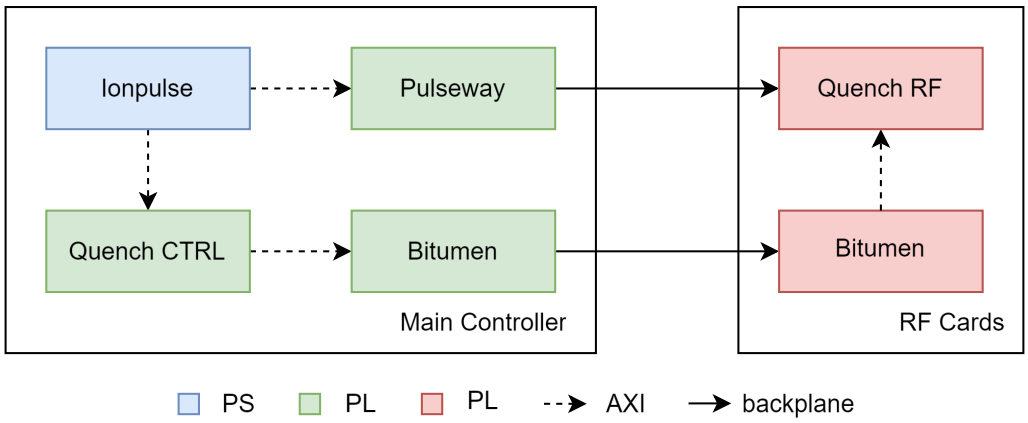


Figure 2.2: QuENCH functional block diagram. In the main controller, the software application provides the instructions for `quench_ctrl` and `pulseway` via AXI. From there, instructions are serialized by `bitumen` and sent to the RF cards over the backplane, together with a trigger signal, where they are deserialized and used to configure sequencers which in turn control the DDS cores.

Ionpulse is the top-level software application that constitutes the entry point to the system. It manages the other modules in order to run the experiments defined by the user, as well as to communicate with Ionizer on the control PC.

[2] AMC FMC Carrier Kintex

Pulseway encloses the functionalities related to digital IO, including a pulse sequencer for the digital outputs, pulse counters for the PMTs and a trigger generator for the RF cards.

Quench CTRL manages communication between the CPU and the RF cards. Besides direct AXI writes it can also use direct memory access (DMA) to retrieve the instructions and distribute them.

Bitumen implements a serial communication protocol with error correction for backplane instructions.

Quench RF encapsulates the RF signal generation. Instructions received from the backplane are interpreted by the `bp_codec`[3] and used to write to the parameter and instruction block RAMs (BRAMs) or perform control actions.

2.3.1 Quench CTRL

The control firmware implemented in `quench_ctrl` uses DMA to fetch the instruction sequence from the processing system's (PS) random-access memory (RAM) via the master AXI interface[4]. The instructions are sent over the backplane, with asynchronous FIFOs being used to handle the clock domain crossing between the management and the backplane clock. This configuration allows streaming data to multiple cards simultaneously. One `bitumen` core per card serializes the data stream for transmission.

2.3.2 Pulseway

In `pulseway`, instructions for output generation are stored by the PS in a BRAM, accessed via a slave AXI interface. From there, the instructions are interpreted by a sequencer and used to configure a timer which manages the status of the digital outputs, including the TTL lines and the trigger for `quench_rf`. FIFO updates, i.e. parameter changes for loops, may be sent via DMA.

Each PMT channel is monitored by two pulse counters: one keeps track of the number of pulses it receives, while the other implements a time-tagging functionality that stores the timestamp when each event is received. The counters push data to a FIFO which is polled from the PS. Additionally, the module offers support for an external trigger input for line triggering as well as a camera.

2.3.3 RF generation

The RF cards implementing `quench_rf`, as depicted in fig. 2.3, have access to 8 DAC channels, each individually controlled by a `quench_rf_channel`. In turn, each of these channels has 4 independent sine synthesizers implementing the DDS cores. Access to 3 arbitrary waveform generator (AWG) channels, streaming from a RAM at 1600 MT/s with a width of 32 bits, is also available. Note that these are shared between all 8 physical outputs; each DAC channel can select whether to use the sine synthesizers or the AWG.

[3] The instructions are encoded for the `bp_codec` directly in the software driver.

[4] A slave AXI interface is used to configure the master AXI. It may also be used as an alternative, slower method for the PS to send instructions to the FPGA or over the backplane.

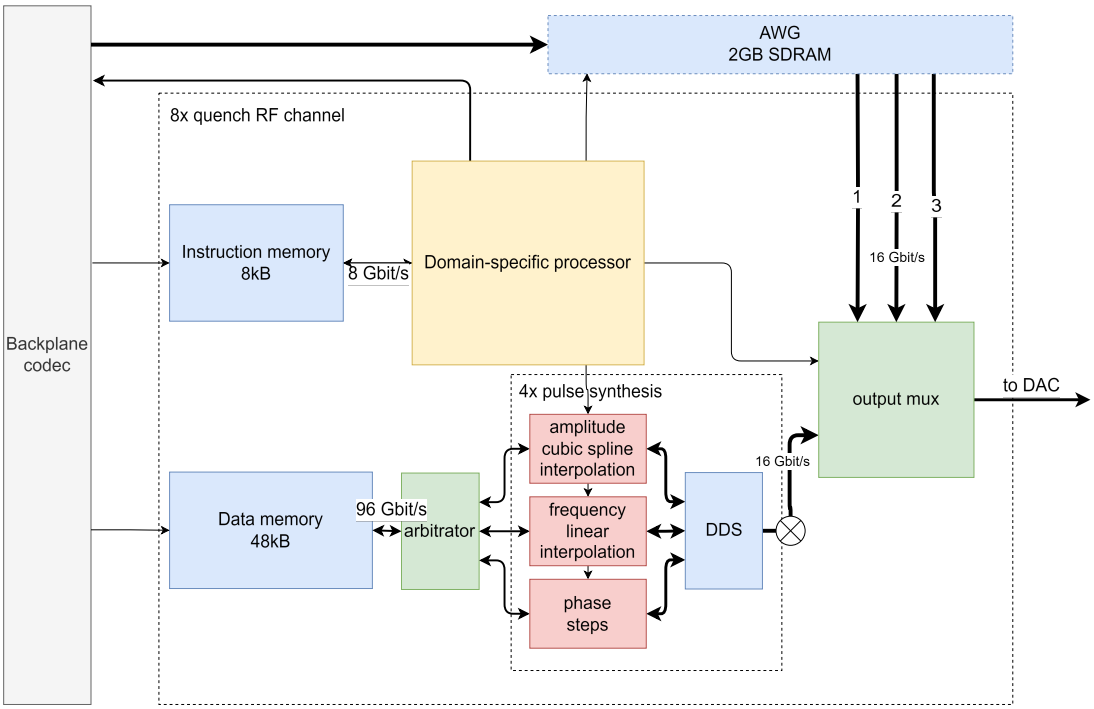


Figure 2.3: Schematic of the `quench_rf` module. It comprises 8 channels implementing 4-tone DDS, which also have access to 3 AWG channels. A backplane codec inteprets the instructions from the main controller. The module's main components are described in the main text.

Inside a `quench_rf_channel`, three dedicated interpolation engines provide the parameters for each sine synthesizer. These implement linear frequency interpolation, phase stepping and cubic spline amplitude interpolation, allowing the generation of chirped pulses and smooth amplitude shapes.

The basic operation of the DDS cores is as follows. Based on the fact that the phase is the integral of the frequency, a 32-bit binary counter – the so-called *phase accumulator* – is kept, representing the instantaneous phase of the signal. The *phase parameter* is included as an offset on this counter. At every clock cycle, the counter is incremented by a 32-bit frequency control word F, given as a fraction of the DDS clock frequency f_{clk}, i.e.

$$F = 2^{32} \times \frac{f_{out}}{f_{clk}} \tag{2.1}$$

where f_{out} is the target *frequency parameter*. Note that F is rounded to the nearest integer, resulting in a frequency resolution of $f_{clk}/(2^{32})$. This rounding is already done in the software, before the parameter is sent to the FPGA.

The instantaneous phase is converted from an angle into an (16-bit) amplitude according to a sinusoidal pattern. This step is implemented by means of a 9-bit look-up table (LUT), addressed with the most significant bits of the phase accumulator and extended to 12 bits based on the symmetry of sine and cosine, and a linear interpolator. The sampling rate is increased with respect to the DDS clock by generating multiple samples per cycle with the proper phase offset. Finally, the different tones are combined together and scaled according to the *amplitude parameter* before being sent to the output DACs.

2.4 Digital IO

The 32 digital outputs coming out from the Zedboard are routed to two DSUB-25 connectors, 16 signals to each. These can be used to easily connect to a custom TTL breakout box that shifts the voltage level up to 5 V and provides convenient BNC connectors, as well as power isolation and a LED indicator showing the level of each individual channel.

The PMT signals, together with an external input trigger, are fed into a small breakout board with SMA connectors. At the time of writing, two different boards can be found in the lab for this purpose. The legacy solution simply level-shifts the signals from 5 V to 3.3 V, with single-ended outputs. The new board uses LVDS buffers, which can handle pulses shorter than 10 ns, a separate circuit for the line trigger and an option to use one of the channels for time tagging.

Chapter 3

Target device

In this chapter we provide an overview of the most relevant characteristics and components of the AMD Zynq Ultrascale+ RFSoC ZCU216 Evaluation Kit used in this project, as well as the auxiliary parts that are specifically used with it. The information presented here is largely taken from the ZCU216 user guide [10] and the Zynq Ultrascale+ RFSoC data sheet [11]; further details can be found there.

3.1 Evaluation board: RFSoC ZCU216

The ZCU216 is an evaluation board designed for radio-telecommunication applications with frequencies below a few GHz, featuring the ZU49DR AMD Zynq™ Ultrascale+™ RFSoC Gen3 device[1]. The chip is split up into three main parts: the processing system (PS), the programmable logic (PL) and the RF Data Converter (RFDC).

3.1.1 Processing system

The Zynq UltraScale+ processor consists of two separate processing units: a 64-bit quad-core ARM Cortex-A53 application processing unit (APU), based on the ARMv8-A architecture, and a 32-bit dual-core ARM Cortex-R5F real-time processing unit (RPU), based on ARMv7-R.

It provides dedicated modules for a number of peripherals related to external memory interfacing, high-speed interfacing and general connectivity. The latter includes a pair of USB2.0 controllers, an I2C controller, a UART, four triple-speed Ethernet MACs and 128 bits of general purpose IO. Peripherals can be assigned to a predefined group of pins among the 78 dedicated multiplexed IO (MIO) pins. Since these may not be enough to allocate all the required peripherals, 98 extended multiplexed IO (EMIO) pins are available, allowing unmapped PS peripherals to access the PL IO.

The PS-PL interface also contains several AMBA AXI4[2] interfaces for primary data communications, most notably including 4 high-performance (HP) slave AXI interfaces to the PS DDR memory, 2 high-performance coherent (HPC) ports to the

[1] Throughout this thesis, the term RFSoC will be used to refer to this particular chip.

[2] Arm Advanced Microcontroller Bus Architecture, Advanced eXtensible Interface v4 [12, 13]

cache coherent interconnect (CCI), which allow access to the same data as the CPU without having to flush caches, and 2 HP master AXI interfaces from PS to PL. Additionally, there are 4 PS clock outputs and 4 PS reset outputs to the PL.

3.1.2 Programmable logic

Programmable logic is available by means of an UltraScale+ FPGA. It comprises ∼50k configurable logic interconnects (CLBs), each containing 8 6-input LUTs and 16 flip-flops, ∼4k digital signal processing (DSP) slices with 27×18 multipliers and 48-bit accumulators, ∼1k 36 Kb BRAMs with built-in FIFO support, and 80 4k $\times$ 72-bit UltraRAM blocks. All these elements are connected with a network of high-performance, low-latency interconnects. In addition to logical functions, the CLB provides a shift register, multiplexer and carry logic functionality as well as the ability to configure the LUTs as distributed memory to complement the configurable BRAMs. The DSP slice may also perform additional independent functions making use of its 96-bit-wide XOR functionality and 27-bit pre-adder.

The PL provides ∼400 configurable IO pins supporting multiple standards. They are divided between high-performance (HP) and high-density (HD) banks, with somewhat different capabilities [14]. Additionally to single-ended mode, most pin pairs can be configured as differential input or output pairs, and optionally terminated with a 100 Ω internal resistor. Double data rate (DDR) is supported by all inputs and outputs.

Clock generation and distribution components are located adjacent to the columns that contain the memory interface and input and output circuitry, providing low-latency clocking to the IO. Within every clock management tile (CMT) resides one mixed-mode clock manager (MMCM), two phase-locked loops (PLLs), clock distribution buffers and routing, and dedicated circuitry for implementing external memory interfaces.

The MMCM can serve as a frequency synthesizer for a wide range of frequencies and as a jitter filter for incoming clocks. At its center is a voltage-controlled oscillator (VCO), whose speed depends on the input voltage it receives from the phase frequency detector (PFD). Three sets of programmable dividers can be configured to provide the desired output frequency while keeping the VCO within the specified range. The output phase is also configurable under certain limitations. In turn, the PLLs offer fewer features than the MMCM, and are primarily present to provide the necessary clocks to the dedicated memory interface circuitry. Aside from the ones in the CMTs, the RFSoC is equipped with five additional PLLs in the PS for independently configuring the four primary clock domains within the PS: the APU, the RPU, the DDR controller and the IO peripherals.

Clocks are distributed throughout the PL via buffers that drive a number of vertical and horizontal tracks. There are 24 horizontal and 24 vertical clock routes per clock region, with 24 additional vertical clock routes adjacent to the MMCM and the PLL. Within a clock region, clock signals are routed to the device logic (CLBs, etc.) via 16 gateable leaf clocks. Clocks can also be transferred from the PS to the PL using dedicated buffers.

3.1.3 RF Data Converter

The RF Data Converter Gen3 [15] subsystem integrates 16 14-bit 9.85 GSPS DACs[3] and 16 14-bit 2.5 GSPS ADCs. It implements efficient digital up-conversion (DUC) and digital down-conversion (DDC), comprising programmable interpolation and decimation, a numerically controlled oscillator (NCO) and a complex mixer.

The analog circuitry for the RF-DACs supports the ability to adjust the output power, which is combined with digital control to implement the RF-DAC Variable Ouput Power (VOP) feature. The output current can be adjusted arbitrarily up to 40.5 mA (or 6.5 dBm at 50 Ω). Furthermore, the RF-DACs offer a mix-mode feature to optimize their output response to the second Nyquist zone by mixing the RF-DAC data with the sampling clock. As for normal operation in the first Nyquist zone, the output suffers a roll-off sinc response. An inverse sinc filter is made available to counteract this effect in applications that require a flat-output response over a wide bandwidth.

The RF-DACs and RF-ADCs are arranged in tiles[4] of 4 channels each. Each tile includes a clocking system with its own PLL. An external reference clock is required as either the sampling clock or as a reference clock to the internal PLL; one such differential clock is to be provided per tile[5], with an internal 100 Ω termination and a clock buffer already implemented in the tile architecture. Alternatively, tiles can forward their sampling clock to adjacent tiles, under certain constraints, through an on-chip distribution network. An optional user-configurable clock may be output from each tile to the PL.

Communication with the PL occurs by means of a single AXI4-Lite configuration interface and multiple AXI4-Stream data interfaces, one per channel. The AXI-Lite clock is to be driven by the system CPU and is common to all tiles, whereas each tile has its own AXI-Stream clock, which must be frequency-locked to the RF-DAC/RF-ADC sampling clock.

The aforementioned clocking and data interface offers great flexibility, allowing each tile to be driven with individual sample rates and PL data-word widths. The converters within a single tile share the same infrastructure, so the sample rates and latency are the same. However, for applications that require more than one tile, or even more than one device, matching latency across tiles is critical. The multi-tile synchronization (MTS) feature can be used to achieve relative and deterministic multi-tile and multi-device alignment. Its implementation is based on a simplified version of the standard JESD204B scheme, which uses a SYSREF clock to measure and correct the non-deterministic latency in the FIFOs that transfer data between the PL clock domain and the converter sample clock domain. For that purpose, the reference has to be provided to both the tiles and the PL, and is conveniently referred to as analog SYSREF and PL SYSREF, respectively.

[3] These are sometimes referred to as RF-sampling digital-to-analog converter (RF-DAC); idem for RF-ADCs.

[4] Tiles are numbered 0 – 3 or, equivalently, 224 – 227 for ADCs and 228 – 231 forDACs.

[5] While this is true for the RFSoC, the ZCU216 only provides reference clock connections for ADC tiles 225 and 226, and DAC tiles 229 and 230.

3.1.4 Booting and configuration

Zynq Ultrascale+ RFSoCs use a multi-stage boot process governed by the PS. Upon reset, the device mode pins are read to determine the primary boot device to be used. One of the CPUs executes code out of an on-chip ROM and copies the first stage boot loader (FSBL) from the boot device to the on-chip memory (OCM). Afterwards, the FSBL is executed, which initiates the boot of the PS and can load and configure the PL. This step may also be deferred to any later stage. The FSBL will typically load either a user application or, optionally, a second stage bootloader (SSBL) such as U-Boot. The SSBL will then continue the boot process by loading code from any of the primary boot devices. The entire boot flow is discussed in detail in [16].

In the ZCU216, available booting methods are using the 4 Gb dual Quad-SPI flash memory, a Micro-SD card, the USB-to-JTAG bridge and a JTAG pod flat cable header. Note that JTAG only supports non-safe booting and is recommended only for debugging, though it is the most convenient and thus most commonly used mode of operation during active development. The configuration mode is set through a dedicated 4-position DIP switch.

3.1.5 Connectivity

The ZCU216 complements the RFSoC by offering external connectors for a number of peripherals, including, but not limited to:

- Ethernet
- USB3.0
- JTAG
- UART
- 5x user push-buttons
- 8x PL user DIP switches
- 8x PL user RGB LEDs
- Power and status LEDs
- PS DDR4 4 GB, 2x PL DDR4 4 GB
- 2x PL PMOD (2x6 receptacles, 3V3 level-shifted)
- PL FMC+ HSPC

A USB-to-Quad-UART bridge collects the available UART peripherals, as well as the JTAG chain, into an easily accessible micro-USB connector. Its port assignment is given in table 3.1.

The FMC+ interface, based on the VITA 57.4 specification, offers a subset implementation of the high serial pin count connector (HSPC). In particular, it provides connectivity for 68 single-ended or 34 differential user-defined signals (LA[00:33]), 8 transceiver differential pairs. 2 transceiver differential clocks, 2 differential clocks, and 239 ground and 16 power connections.

Table 3.1: Port assignment between the USB-to-UART bridge and the RFSoC.

USB-to-UART bridge	Zynq Ultrascale+ RFSoC
Port A - JTAG	ZCU216 JTAG Chain
Port B - UART0	PS UART0
Port C - UART2	PL UART2
Port D - UART3	System Controller UART

I2C connectivity is distributed along two separate buses. One connects specific banks in the PS and the PL, and the system controller, to a GPIO 16-bit port expander, that enables controlling some resets and power system pins, as well as accepting various input alarms without requiring the PL-side to be configured. It also reaches power controllers and power monitors through an I2C switch. The second bus connects the same sources to another two I2C switches that enable communication with various I2C capable target devices such as the configurable clocks, the FMC connector or the PS DDR.

Several clocking options are offered. Those integrated in the board itself include a 100 MHz and a 125 MHz fixed-frequency clocks, a 33.33 MHz PS reference clock and three user-configurable SI570 oscillators. Furthermore, a dedicated connector provides additional clocks for the PL and the RFDC from an external clocking board.

The RF signals are taken out through two special connectors, labelled as RFMC 2.0, together with 32 digital IOs, to be accessed with an external break-out board.

3.2 Clocking board: CLK104

The CLK104 RF clock add-on card [17] provides an ultra low-noise, wideband RF clock source for the ADCs and DACs, as well as the PL. Its complete block diagram of is given in fig. 3.1.

The main component is the LMK04828B clock distribution chip which spawns all the necessary differential clock signals, including the PL clock, the analog and PL SYSREF, and four DAC/ADC reference clocks. Regarding the latter, two of them are taken directly to tiles 226 and 230. The other two, that correspond to tiles 225 and 229, each have their own dedicated LMX2594 PLL RF synthesizer, offering a higher output frequency, and reach the ZCU216 by means of external Carlisle SSMP connectors.

A 10 MHz reference clock is to be provided to the LMK; its source can be chosen between an external single-ended SMA connector, an internal TCXO crystal, or a clock from the RFSoC PL. Optionally, an external synchronization signal can also be used for multi-board designs.

All three chips can be configured through an I2C bus. Note that there are separate connections to write to the registers of each component, but reading from them is done through a single, shared bus. The chip to be read must then be selected through a multiplexer, where the select signal is set via serial peripheral interface (SPI) from a dedicated GPIO pin in the PS.

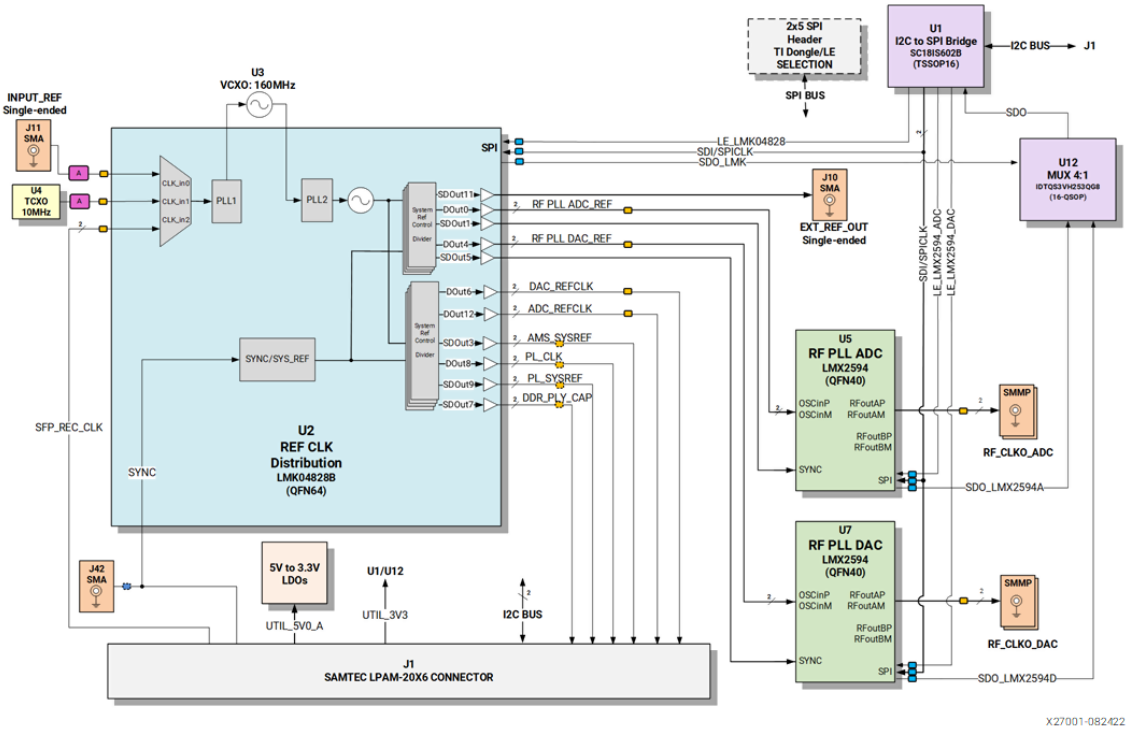


Figure 3.1: CLK104 RF clock add-on card block diagram. Several clocks for the PL and the RFDC are generated in the LMK chip (blue). In turn, two LMX PLLs (green) allow to obtain higher clock frequencies for the DAC and ADC reference clocks. Configuration of the board is carried out via an I2C bridge (purple). Figure taken from [17].

From a user perspective, two methods are available to upload the configuration: manually with the Board User Interface (BUI)[6] or via software using the RFCLK driver. Only a small set of example configuration values is provided by default. Custom configurations can be obtained using the TICS Pro software. A small guide is provided in the appendix A.

3.3 RF breakout board

As mentioned above, the ZCU216 incorporates two connectors specifically for the RF components which comprise connectivity for the analog signals from the RF-DACs and to the RF-ADCs, together with 32 digital IO signals, an I2C bus and several power lines. These are originally expected to be used with the XM650 or the XM655 boards, depending on the desired application. The latter was used during the early development stage for this thesis. However, as described below, its particular choice of baluns is sub-optimal for our needs, resulting in the design of a custom PCB.

3.3.1 XM655

The XM655 balun add-on card is a full break-out board of 16 DAC channels and 16 ADC channels to SMA connectivity using Carlisle-CoreHC2 assembly connections.

[6] These tool is found in the ZCU216 Board Interface Test (BIT), available from the board resource files.

Table 3.2: Specifications for the baluns in the XM655 and custom break-out boards. All the listed baluns present a balanced to unbalanced configuration with a 50 Ω termination.

Label	Part	Frequency range
XM655		
Low-freq. (10 MHz – 1 GHz)	Minicircuits TCM2-33WX+	10 MHz – 3 GHz
Mid-freq. (1 – 4 GHz)	Anaren BD1631J50100AHF	1.6 – 3.1 GHz
High-freq. (4 – 5 GHz)	Anaren BD3150N50100AHF	3.1 – 5 GHz
High-freq. (5 – 6 GHz)	Anaren BD4859N50100AHF	4.8 – 5.9 GHz
Custom		
A	Minicircuits TCM2-33WX+	10 MHz – 3 GHz
B (low-freq.)	Minicircuits ADT2-1T	4 – 450 MHz

It comprises four different sets of four baluns, each targeting a different frequency range, as reported in table 3.2.

Four pin headers break out up to 40 digital IOs. Note that 8 of these pins on the headers are not connected. Additionally, these headers also expose the I2C bus and various power and ground lines.

3.3.2 Custom breakout board

The XM655 baluns were chosen to span as much of the frequency range of operation as possible, in order to provide maximum flexibility. However, the required signals for our trapped ion experiments rarely go above a few hundreds of MHz.

The new board comprises dedicated baluns for the 16 DAC channels: (A) 12 of the original low-frequency ones, and (B) another 4 operating at a lower nominal frequency (see table 3.2). In addition, all 16 ADC channels were connected via an instrumentation amplifier[7] offering a theoretical input range from DC to 22 MHz. Consequently, we now have access to all 32 RF channels, instead of only 16 as with the XM655.

The pin-header strips on XM655 were replaced by two DSUB-25 that expose the digital IOs, offering direct compatibility with the digital output isolator currently used in the control system. Since these connections were planned to be used solely as outputs, unidirectional buffers were placed. In this case, no connections were made available for the I2C bus and the power lines.

We highlight that this is only the first version of the board and it is likely subject to considerable changes in the future. Some upgrades are already planned for a revision, such as changing some of the digital outputs into inputs for the PMTs, or routing the RF output through a PCB edge connector for simplified front panel connectivity with lower losses.

[7] INA851RGTR

Chapter 4

Hardware architecture

In the process of porting the system from a multi-FPGA to a single-FPGA configuration – sketched within the full control system in fig. 4.1 – the gateware has to undergo significant changes. Most significantly, backplane communication is to be removed. Furthermore, the use of a processor and FPGA from a new family entails a new set of available components and new constraints that need to be considered carefully. The adaptation of the hardware design constitutes the bulk of the tasks carried out for this thesis, the result of which is presented in detail throughout this chapter, together with the most relevant issues encountered during its development.

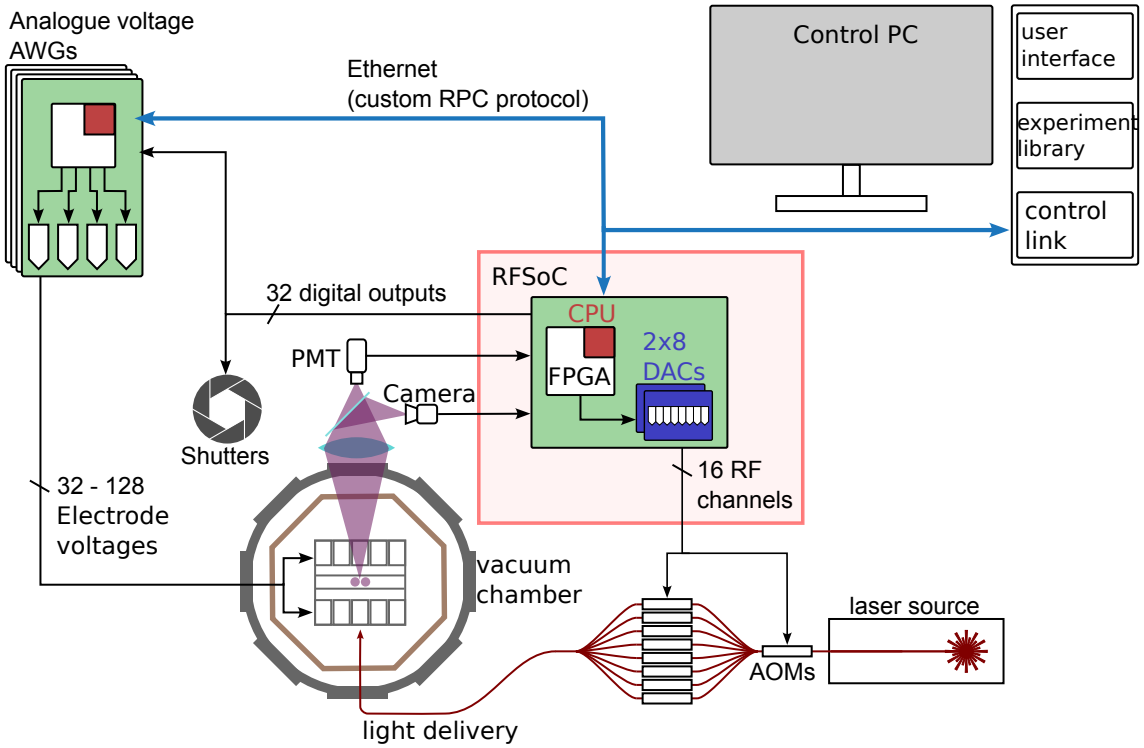


Figure 4.1: Schematic of the complete control system using the RFSoC, highlighted in red, to replace the main controller and RF cards. Here, 16 DACs are integrated in the main controller, divided into two separate virtual slots. The surrounding elements remain the same as in fig. 2.1.

4.1 Gateware overview

At a functional level, the module structure is still fairly similar, as depicted in fig. 4.2. The main difference resides in the removal of the serial communication layer, i.e. `bitumen` and the entire backplane. This functional description, however, overlooks important details.

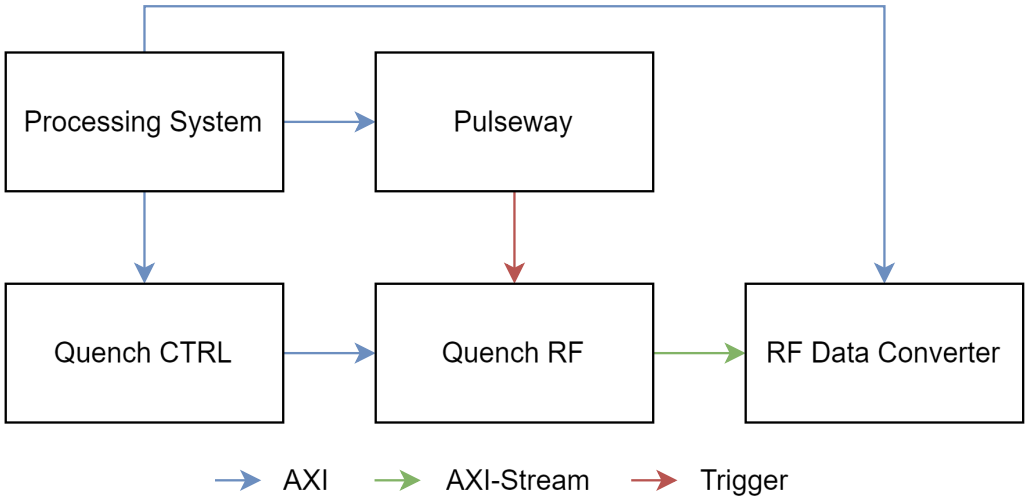


Figure 4.2: Overview of the gateware implementing QuENCH in the RFSoC. As in the original `quench` system, the PS prepares the instructions for `pulseway` and `quench_ctrl`. From there, however, the instructions and the trigger signal are sent directly to `quench_rf`, skipping serialization and the exernal backplane. The DACs are handled by the RFDC IP, where data is to be provided by an AXI-Stream interface.

At the centre of the design resides the new processor, an ARM Cortex A53. On the one side, two master HP AXI interfaces are used to transfer data from the PS to `quench_ctrl` and to `pulseway` respectively, and to dynamically set the configuration of the RF Data Converter. On the other side, two slave HP AXI interfaces enable receiving feedback and output data from `quench_ctrl` and `pulseway`. Several AXI SmartConnects are placed to automatically translate between interfaces that use different versions of the AXI protocol. The Address Editor in Vivado can be used to automatically assign the AXI address range for each component based on the address width of each interface, such that address overlaps are avoided.

Since the RF generation is now done on the main controller, a wrapper for `quench_rf` is instantiated. Data is sent directly from `quench_ctrl` without the need for serialization. Therefore the `bitumen` module can be removed. For compatibility with the gateware and software on the rack-based system, the `bp_codec` instruction decoder is kept. Alternatively, one could remove the decoder and directly expose the instruction and data BRAMs to the software driver. A single trigger signal runs from `pulseway` to `quench_rf` to mark the start of the sequence. Furthermore, the PL UART is set up for the UART debug functionality; its usage is presented in section 6.1.

A dedicated IP is available for the RFDC. For now, only the 16 DAC channels are instantiated; ADCs can be added in a future extension of the project. Samples

Table 4.1: Configuration for the user LEDs.

LED index	Signal
LED0	MMCM *locked*
LED1	Down-converted clock
LED2	Constant LOW
LED3	PMT3
LED4	PMT2
LED5	PMT1
LED6	PMT0
LED7	TTL0

are sent from the quench_rf wrapper through individual AXI-Stream buses to the RFDC.

The design is completed with 8 green LEDs, mainly for debugging purposes, allocated according to table 4.1. The clock down-conversion is done by a simple clock counter, that toggles the output signal every fixed number of positive edges – in our case matching the clock frequency for a resulting period of 1 s.

Early on in development the decision was made not to include the two GPIO output signals required for the CLK104 multiplexer select SPI, as it normally requires an AXI-to-GPIO interface [18] that unnecessarily clutters the design if one is not going to read the clocking configuration. The only reason for reading back the configuration would be to ensure that communication is working or validate what was previously written. The implications of this decision on the software driver are discussed in section 5.2.2.

4.2 Clocking structure

The entire PL clocking structure, depicted in figure 4.3, is generated from a single MMCM in order to guarantee phase alignment between all clocks. The input clock `PL_CLK` is provided at a rate of 125 MHz, the maximum frequency allowed by the input clock buffers. All other clocks are derived from the input clock. The safe clock startup option is enabled, as recommended by Vivado, and the locked signal is made available to check from the PS when the clock is stable during the initial setup.

The management clock `MGMT_CLK` at 125 MHz is common for all AXI interfaces. Even though a `MGMT_CLK` of 250 MHz would work for most of the system, there is currently a design flaw related to the FIFOs in `quench_ctrl` that causes some unexpected behaviour when processing looped parameters if the clock is too fast. Lowering the AXI clock only represents a temporary solution until this issue is properly addressed.

Most clocks are related to the RFDC, and hence their rate is determined by its desired configuration. For the settings given in section 4.3.5, the IP prompts the required AXI-Stream clock at 250 MHz. Thus, the `DAC_CLK` is provided to tile 230 at that frequency and distributed from there to the other three tiles, without it being modified in any of the tile PLLs. The analog DAC reference clock `DAC_REF_CLK` is conveniently set to the same frequency.

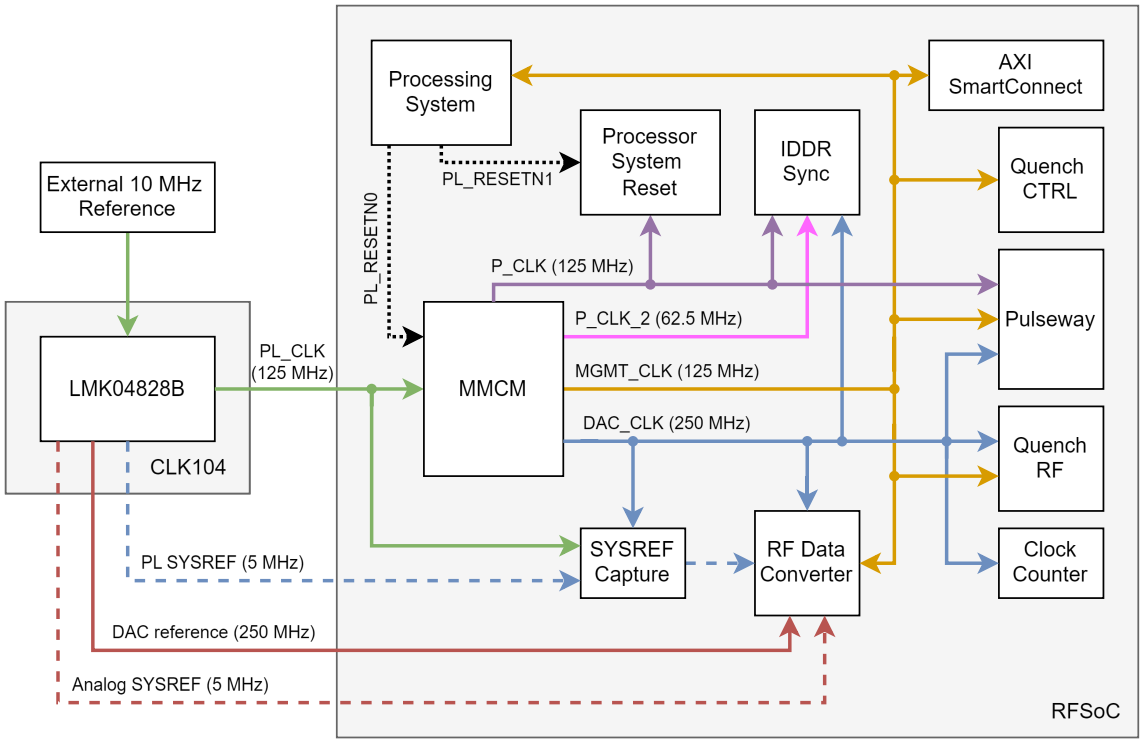


Figure 4.3: Schematic of the clock distribution structure. The LMK04828B in the CLK104 is used to generate the main clock for the FPGA, as well as the DAC reference clock and the SYSREF. A single MMCM derives the internal clocks for the FPGA, keeping them synchronous. Two PL resets are generated from the PS. Further details on the clocks involved are given in the main text.

Since MTS is enabled, the RFDC also imposes strict requirements on the SYSREF, as discussed in [15]:

1. SYSREF must be a high-quality, free-running, low-jitter signal.
2. SYSREF must be an integer submultiple of all PL clocks that sample it, and be less than 10 MHz.
3. SYSREF must be safely captured by the PL, before passing to the core. In particular, setup/hold of the PL SYSREF to PL clock must be handled as part of the user design
4. Analog SYSREF and PL SYSREF must be the same frequency and have a constant phase relationship.
5. For MTS synchronization, the analog SYSREF and PL SYSREF must be a continuous clock for the duration of the MTS procedure.

Both clocks are generated from the CLK104, which already covers some of these points, at a suitable frequency of 5 MHz. To capture the PL SYSREF, it is sampled by two consecutive flip-flops, clocked with the PL clock and the DAC clock, respectively.

Last, the MMCM generates two additional clocks for pulseway: `P_CLK`, at half the frequency of `DAC_CLK` (125 MHz), and `P_CLK_2`, at a half of that (62.5 MHz). These are used for input double data rate (IDDR) synchronization (see section 4.3.3). `P_CLK` and `DAC_CLK` are also sent to `pulseway`, where the latter is considered the double-rate clock.

Two active-low resets are generated from the PS. First, `pl_resetn0` enables resetting the MMCM alone, which is recommended after updating the input clock during the CLK104 configuration. After the clock is locked, a second reset (`pl_resetn1`) is sent to all PL components. A Processor System Reset module synchronizes the reset signal to the slowest synchronous clock, in this case `P_CLK` (since all clocks in the design are synchronous and `P_CLK_2` doesn't actually sample any component with a reset, only clocking an IDDR register). From there, dedicated reset signals are distributed to the interconnects, and to peripherals with an active-high or active-low reset.

4.3 Hardware components

4.3.1 Processing System

Configuring the PS with its numerous peripherals and tight coupling to the board constraints is a complex task. Fortunately, Vivado already provides a board preset [19] for the ZCU216. After the PS is instantiated in the board design, a Run Block Automation option is offered to automatically apply the preset configuration. Doing so ensures that the UART, I2C and other peripherals are properly configured, but the user can still customize the configuration freely.

For this project, only a few modifications are required. First, we need to enable the two PL reset signals. Up to four such signals are supported which, if enabled, are assigned to the last 4 EMIO pins. In our case, this corresponds to EMIO 95 and 94,

located in bank 5, for plreset0 and plreset1, respectively. Second, we configure two HP slave and two HP master AXI interfaces, all in the full power domain (FPD), where the DMA unit is managed by the APU[1]. Finally, a single GPIO input pin is used to read out the locked signal from the MMCM (EMIO 0).

4.3.2 Quench CTRL

The current implementation of `quench_ctrl` already provides an option to exclude `bitumen` and expose the parallel interface instead. To do so, the `EXCLUDE_BITUMEN` flag must be set, which we define in a Verilog header file that is globally included. No further changes are required on this module.

4.3.3 Pulseway

Aside from the management clock, two main clocks are used in `pulseway`: a slow clock `P_CLK`, used for the FIFOs, and a fast clock `P_CLK_X2` at double the slow clock's speed, used for the scheduler and counters. The instruction decoder can work with either of them.

In our design, `P_CLK` is sourced by the homonymous clock at 125 MHz, with `P_CLK_X2` being associated to `DAC_CLK`. The double-rate clock is selected for decoder. Furthermore, we disable the AC trigger and the camera, and configure 4 PMT inputs.

A major difference with the design for the Zedboard (Zynq 7 Series FPGA) is that IDDR registers use a different component, i.e. `IDDR` for the Zynq 7 [21, 22] and `IDDRE1` for the UltraScale [23, 24]. While Vivado is, in principle, able to implement the appropriate module even if the other is used in the HDL description, some constraints apply to their arguments. Therefore, it is more convenient to support a separate description depending on the hardware in use.

IDDR is used to synchronize the PMT and sync (external line trigger) inputs. This is completely detached from the rest of the `pulseway` functionality, and thus the decision was made to take it out of the module. This change was applied not only to this project, but also to the Zedboard. However, the mode of operation is slightly different.

In the Zedboard, the `IDDR` is clocked at the desired input frequency (`P_CLK` for sync and `P_CLK_X2` for PMTs), and the input is sampled only in the rising edges. That is, the double-rate feature is not used, but only the inherent synchronization it provides. We cannot do the same due to the fact that the input buffers cannot be clocked at our higher clock speeds (>125 MHz in HD banks [14]). The solution is to actually take advantage of the double data rate, sampling at both edges of a clock at half the target speed, and then sample them alternately at the faster clock rate. Note that in order to take the samples in the correct order, regardless of the clock phase, the `IDDRE1` must be used in `OPPOSITE_EDGE` mode.

[1] The FPD can be shut down while still operating the low power domain (LPD) to reduce power consumption drastically [20].

4.3.4 Quench RF

The `quench_rf` wrapper in the current control system is very specific to the AFCK, providing, for instance, specific controllers for the JESD204 interface to the DACs, an internal configuration manager (ICM), and an interface between the AWG and an external DDR (MIG). However, the DDS cores are encapsulated in the `qu_rf_main` module, which we can use without having to modify it.

A `qu_rf_main` instance is composed of the 8 `qu_rf_channel`s, with `bp_codec` providing the input interface. The output is given in a single array, with the number of samples per clock cycle being configurable. In our new `quench_rf` wrapper, we generate two `qu_rf_main` instances for 16 RF channels total. In order to maintain compatibility with the software, we consider them as virtually two separate slots. Any remaining signals related to the ICM or MIG are ignored or set to a constant zero.

The UART debug module and the trigger capture from the original wrapper are also included. However, special care needs to be taken to distribute the UART instructions to the two virtual cards; the slot mask is used for this purpose (for backplane instructions, this information is already provided from `quench_ctrl`). Output data is also multiplexed according to the slot mask, only if exactly one slot is selected, to avoid conflicts.

Since we remove the JESD interfaces, we must adapt the parallel output bus to the RFDC AXI-Stream interface. The samples are already provided as an array of 16-bit signed integers, ordered as expected by the RFDC, so we can partition the bus into individual channel arrays and feed them to the corresponding AXI-Stream `tdata`. It is assumed that the DDS cores provide valid data at all times. Hence, the only restriction on the AXI-Stream `tvalid` signal is that it has to be asserted at least one cycle after a reset [25]. A flip-flop with a synchronous reset may be used for that, momentarily deasserting `tvalid` whenever a reset is received. Since both sides are using the same clock, no clock domain crossing is needed.

Parameters such as the number of samples per channel per clock cycle are derived from the target DDS sampling rate and clock frequency. Therefore only a couple parameters have to be considered to adjust `quench_rf` in conjunction with the RFDC settings.

4.3.5 RF data converter

The RF Data Converter is configured to instantiate all the DAC channels, with MTS enabled and the DAC reference clock being received through tile 230 and forwarded from there to the rest of the tiles. The rest of the configuration is shared by all 16 DACs, as they are to be used indistinguishably.

The settings were chosen at first to suit a target sampling rate of 4 GSPS[2]. Considering the default 16 samples per AXI-Stream cycle and no DUC, this results in the 250 MHz reference clock requirement discussed in the clocking section. We use real output data (no digital mixing), no interpolation, and the sinc filter enabled.

[2] While DAC speeds of around 10 GSPS – near the limit for our device – are necessary for superconducting experiments, that is not the case for our trapped ion experiments. Currently QuENCH uses a sampling rate of 1 GSPS, so increasing that to 2 – 4 GSPS is already an improvement.

Throughout most of the development, the system was used with a sampling rate of 1 GSPS to reduce logic resource consumption on the FPGA, for decreased implementation time. In an effort to keep the required changes on the clock structure to a minimum the reference clock has not been modified, but rather we enabled the DUC datapath mode to use 4 samples per AXI-Stream cycle, instead of the compulsory 16 samples per cycle when DUC is disabled.

4.4 Constraints

Design constraints define the requirements that must be met by the compilation flow in order for the design to be functional on the board [26]. These can be divided into physical constraints, used during the implementation steps to place the components into the given location, and timing constraints, used to verify that the timing requirements are fulfilled. For the purpose of this section, physical constraints regarding clocking components (e.g. clock buffers) are described under timing constraints.

4.4.1 Physical constraints

Physical constraints are provided in the ZCU216 master XDC file. Most of the signals are already fixed to specific peripherals as described there. This includes the differential PL and PL SYSREF clocks, the PL UART, the DIP switches and the LEDs. Note that the analog RF and clock connections are hardwired and thus not defined in the constraints. The digital IO signals offer some more flexibility. For the first version of the custom breakout board, the 32 TTLs are constrained to the `DACIO` and `ADCIO` pins, following the mapping specified in appendix B.1. The `pulseway` trigger line is received from pin 1 in the PMOD1 header.

The PMT inputs were originally planned to go on the PMOD0 header. However, those pins are not directly connected to the FPGA, but rather go through a level shifter[3] to reduce the voltage from 3.3 V to 1.8 V. The provided component is not compatible with LVDS, which would prevent us from using differential signals. While this is not critical – currently the Zedboards work with the single-ended buffer board (see section 2.4) – it would be desirable. Furthermore, this particular level shifter is too slow for our purposes. According to the datasheet [27], it has a rise time of around 15 ns for an internal voltage of 1.8 V, whereas the PMTs in our test setup[4] sends 10 ns pulses for every detected photon [28]. A possible alternative would be to replace some of the outputs in the breakout board for inputs, and move those TTLs to the PMOD. However, this is more of a mid-term solution, as it would require a redesign of the board since the buffers are unidirectional, which would exceed the time-frame for this thesis.

Instead, we opted to receive the PMTs signal through the FMC+ connector present in the ZCU216. For that, we use an FMC XM105 Debug Card [29], which provides a large number of general-purpose pins. Hence, it offers a convenient interface to the differential buffer/trigger board. Further details, including the pin mapping and notes on the FMC connector, are given in appendix B.2. An important fact

[3] Texas Instruments TXS0108E

[4] PMT H10682-210 from Hamamatsu. Refer to section 6.4 for details on the test setup.

to consider here is that the FMC is connected to HP pins on the RFSoC. Unlike HD pins, these allow using the internal 100 Ω termination, so external terminations do not need to be soldered onto the buffer board. They are enabled by setting the `DIFF_TERM_ADV` constraint to `TERM_100`.

4.4.2 Timing constraints

External clocks need to be created in the constraints and their period specified; this applies to the PL and DAC clock. Generated clocks that are derived from them do not need to be included, as the MMCM automatically takes care of that.

Another relevant set of constraints is related to the clock resources location and routing. In our design the PL clock is routed to the MMCM through a differential input buffer (IBUFDS) and a global clock buffer (BUFG), as it is input through HD global clock (HDGC) pins [30]. If no such constraints are provided, Vivado raises an error during implementation. In the error message it is suggested to set the `CLOCK_DEDICATED_ROUTE` to `ANY_CMT_COLUMNS` on the output of the IBUFDS, which will turn the error into a warning but is not advised as it is sub-optimal. In the reference it is said to set this constraint to `FALSE` instead, but this is in general discouraged [31]. The warnings were finally resolved by constraining the `CLOCK_DEDICATED_ROUTE` to `ANY_CMT_COLUMN` for the output of the BUFG.

Constraining a GCIO automatically places its destination BUFG in the same clock region, but placing a BUFG does not result in a predictable placement of its destination MMCM. Therefore, Xilinx recommends specifying its location by means of a `LOC` constraint. To find a suitable tile, we first implemented the design without any such constraint, and used the location chosen by Vivado as its fixed value.

Chapter 5

Software

In this chapter we focus on *Ionpulse*, the software application that runs on the processing system and interfaces with the hardware through AXI. Even though a large part of it is agnostic to the underlying hardware, a few device-specific modifications had to be made for the RFSoC. We start off by giving an overview of the application´s structure and main ideas. This is by no means meant to be an exhaustive description – for that the reader can refer to Vlad Negnevitsky's [6] or Martin Stadler's [9] thesis – but rather to provide some context before presenting the extensions that the project has required to include compatibility with the RFSoC. Last, we discuss a couple additional considerations that do not strictly concern the application, but are somewhat related to its development and execution.

5.1 Ionpulse

The C/C++ code is structured hierarchically, following the block diagram in fig. 5.1, with several layers of abstraction. The lowest level is comprised of the drivers for the various communications components (Ethernet, serial) and FPGA modules (`hiway`[1], `pulseway` and `quench`).

The second API layer, referred to as `ionpulse_sdk_core`, encompasses the implementation of the main features. These include the creation of RF and digital pulse sequences, PMT read-out, communication with the PC and DEATH[2,3], and creating experiments and remote parameters. Some auxiliary mathematical libraries and algorithms are also provided. One of the main components of interest here is the `experiment` class, containing the API for the creation of experiments. These are shown as graphical pages in the Ionizer GUI, whose remote parameters can be modified and scanned (i.e. run repeatedly while changing a parameter). Another relevant component is the `bp_dds` API, which interfaces with the drivers to create synchronized digital and RF pulses. An additional class, `bp_dds_cache`, has been introduced to optimize the storage of parameters.

The third level, `ionpulse_sw`, is unique to each experimental setup, where the researchers configure the code for their hardware and implement their own experiments

[1] `hiway` is a legacy module from the M-ACTION system, not relevant for QuENCH.
[2] *Direct Ethernet-adjustable Transport Hardware*
[3] AWGs used to control trap dc electrode voltages

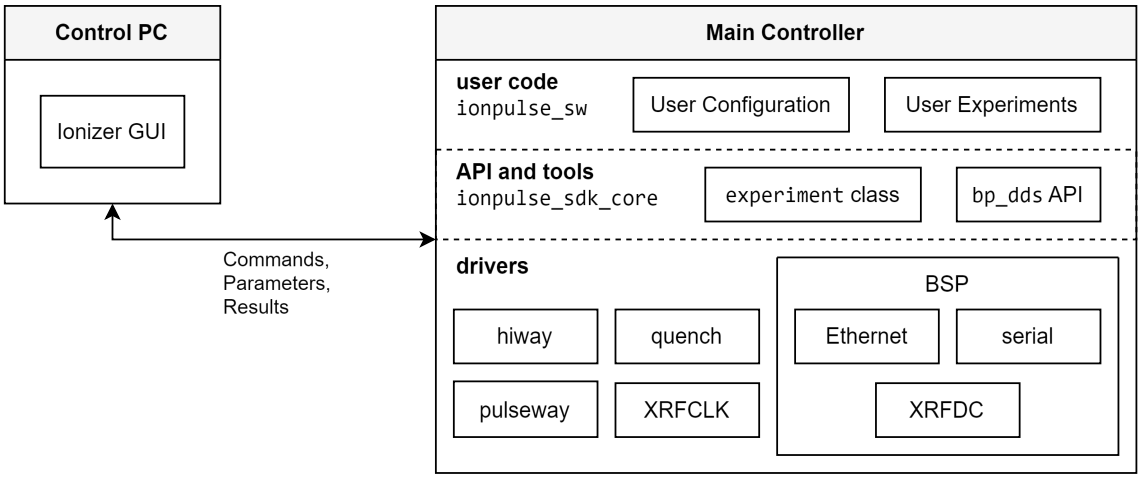


Figure 5.1: Ionpulse software hierarchy. The application for the main controller is based on a layered structure, with several levels of abstraction. At the top level a server handles requests from the PC to receive instructions from the GUI and exchange data.

and custom parameters. Global functions can also be added, to be used by multiple experiments. This is convenient for sequences that are run frequently, such as cooling, state preparation or detection. A development repository, `ionpulse_sw_test` is used by the control team to implement and test new features before they are made available to the rest of the group.

Finally, the top layer consists of the Ethernet server that handles requests from the control PC, where the Ionizer GUI is run. It is used for sending commands to control the execution of experiments, updating parameter values and returning experiment results.

5.2 Extension to RFSoC

5.2.1 Board Support Package

The board support package (BSP) provides the software interface between the PS and the hardware. For the RFSoC, it can be obtained using the AMD Vitis IDE. The BSP sources are automatically generated when a platform component is created using the `.xsa` file exported from Vivado. In the process of building the platform, the BSP is compiled; the resulting files and headers can be included from the Ionpulse CMake. Currently two versions of the BSP are available, corresponding to two different IDEs: Vitis Unified uses the new system device tree (SDT), while Vitis Classic offers support for the legacy BSP. The differences between them are explored in section 5.3.1.

In the BSP settings, we need to enable the lwIP library, which handles network communication. Libmetal is already enabled in the default configuration to access devices such as the RFDC. Besides, the proper timer must be selected for the sleep functionality. By default Vitis uses TTC0, one of the 32-bit triple-timer counters (TTCs) [20]. However, this register overflow roughly every 42 seconds when running at the standard rate of 100 MHz. Instead, we opt for the `Default` timer[4], which

[4]Admittedly, the naming is confusing, with `Default` not actually being selected by default.

uses the 64-bit System Clock. A clock overflow could cause PMT reads to time out, consequently crashing the ongoing experiment.

Two additional files are required: the spec file (`Xilinx.spec`) and the linker script (`lscript.ld`). Both are generated by Vitis when an application component is created, for instance using the Empty Application template. A newly created linker script allocates only a small memory range for the stack and heap, which is not enough for our application and can lead to unexpected behaviour. Reasonable values for these settings were obtained from the Zedboard's linker script (`STACK_SIZE 0x1000000` and `HEAP_SIZE 0x1F000000`).

5.2.2 Initialization sequence

When Ionpulse is first executed, several initialization tasks need to be carried out. These include configuring the host platform and setting up the network, both of which are highly hardware-dependent. The project is already structured in such a way that different implementations can be provided for the corresponding functions, using CMake to include and compile only the relevant files. For the RFSoC, this procedure concerns three main components: network, clocking and multi-tile synchronization.

It is common to find examples (usually targeting other device families) that use this stage to configure the PS UART0 and set it as the standard output stream (stdout) peripheral. In our case this is already done, as can be verified from the corresponding definitions in `xparameters.h`. We may thus use the standard `printf` function to output debug information and receive it on the control computer by monitoring the appropriate serial port (see table 3.1).

Network

Network configuration on the Zynq UltraScale+ is done very similarly to the Zedboard's Zynq7000, primarily due to the use of lwIP to handle the low-level implementation. The library's initialization procedure follows from its multiple examples[5].

The first step is to enable interrupts, which are used by the Ethernet MAC driver. The main difference between both devices relies on the fact that the interrupt structure in ARMv8 underwent some changes from its predecessor [32]. In particular, the various interrupt types were grouped and renamed into just two kinds of exceptions: synchronous and asynchronous. Optionally, interrupt initialization could be skipped altogether when paired with the BSP from Vitis Unified, as the library uses a newly-introduced interrupt driver wrapper.

The remainder of the network procedure applies to both devices, given that the peripheral name is identical. As per the library's instructions, one must initialize a slow and a fast TCP timer, as well as an Ethernet reset timer. Restrictions apply on their timeout value that are not discussed here. The status of these timers is periodically polled, instead of using interrupts.

[5]Examples on the usage of lwIP are provided by Vitis.

Clocking

Configuration of the CLK104 can be done via software using the XRFCLK driver [18]. This driver is not provided in the BSP, but it is available on GitHub[6]. Unfortunately, at the time of writing this thesis it has not been updated to support compatibility with the Vitis Unified IDE. As a result, we had to apply a small fix to stop the I2C initialization from failing.

The base driver requires the device ID (or peripheral address) corresponding to the SPI bus, which is used to select the input chip for the readout multiplexer. Since we do not plan on reading the clock configuration, we prepare a custom initialization function that skips this step. This removes the requirement for the two additional GPIO pins that we deliberately did not include in our hardware design (section 4.1).

As we also mentioned before, the driver only provides a very limited set of possible configurations, none of which matches our required settings exactly. Therefore, we have extended the list of available configurations for the LMK with an additional set of values that we exported from TICS Pro (appendix A). This way we set the proper values for the SYSREF, PL CLK and DAC reference clock, as well as select the external 10 MHz reference. Additionally, for debugging purposes, we forward SYSREF to the reference output pin in the CLK104.

After the clock configuration sequence is completed, a reset pulse is sent to the MMCM. Since the PL resets are mapped to the last bits of EMIO, in GPIO bank 5, their output value can be set by writing the GPIO `DATA_5` register [33]. For instance, in the case of the MMCM, one would clear the most significant bit in that register, wait for an arbitrary amount of time, and set it again, thus creating the reset pulse. Next, we can monitor the MMCM status by checking the GPIO `DATA_3_RO` register [34], where the EMIO0 input (GPIO bank 3, bit 0) is stored. As a reminder, that is where we connected the MMCM's *locked* signal. Once the clock is locked, we send a second reset pulse, now to the rest of the PL.

Multi-tile synchronization

The full MTS sequence is described in the RFDC product guide [15][7]. After configuring the SYSREF clocks and setting up the XRFDC driver, we initialize the MTS functionality for all tiles, using tile 0 as the master reference. We then call the `XRFdc_MultiConverter_Sync` method, which executes the entire procedure for measuring and configuring the internal latencies. Since, for now, we are only considering single-device synchronization, we do not need to specify a target latency; the algorithm will choose an appropriate value. Nonetheless, we do set the logging level for the metal library to `DEBUG`, and report the results at the end of the initialization sequence.

To implement multi-device synchronization, the latency in the FIFOs needs to be measured, which can be obtained from the debug logs, and used together with some margin as the target latency. Furthermore, synchronization of digital features (mixer

[6] https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/board_common/src/rfclk

[7] For a more practical description, we refer the readers to the official blog post written on the topic [35]

Table 5.1: CMake cache variables to be configured in the presets when compiling for the RFSoC.

Name	Value	Meaning
`RFSOC`	ON	Compile for RFSoC instead of Zedboard. Selects proper compiler, platform sources and BSP.
`SDT`	ON/OFF	Consider the SDT version of the BSP. Must be ON only if the BSP was compiled using the Vitis Unified IDE.
`HWY_DUMMY_MODE`	ON	Enable `hiway`'s dummy mode[9].
`quench_slot_mask`	0b00000011	Select which backplane slots contain QuENCH cards. For the RFSoC, we have two virtual slots.
`no_of_pmts`	4	Number of PMT inputs configured in Pulseway.

settings, NCO phase reset, coarse delay, etc.) requires a significantly more complex procedure, especially in the multi-device case. Such features are not employed in our application.

5.2.3 CMake

The incorporation of new or alternative files for the RFSoC, as well as the use of the new processor, have implications on the compilation process. As a result, various files along the CMake structure had to be modified or replaced.

First, the new 64-bit PS architecture entails that the 32-bit compiler[8] is not valid anymore. Instead, to target the Cortex-A53 we need to build our baremetal application using the `aarch64-none-elf` compiler. For that purpose we prepared a separate toolchain file, which also selects the proper system processor and spec file.

Second, some RFSoC-specific files need to be included in place of their Zedboard counterparts. This involves the sources for platform initialization, the BSP and the linker script. An `RFSOC` flag indicates the target device, in order to conditionally select them. Furthermore, an `SDT` flag is used to support any BSP version, regardless of whether it was compiled with Vitis Unified or Classic, accordingly adjusting which libraries and directories are linked.

Last, the `ionpulse_sw_test` CMake presets have been extended to include release and debug configurations for the RFSoC. These are set up to point to the appropriate toolchain file and define all required cache variables. A full list of said variables is given in table 5.1.

5.3 Additional considerations

5.3.1 Vitis Classic vs Unified IDE

Ionpulse was originally developed using the Xilinx SDK (later rebranded as Vitis) and by the start of this thesis it was fully compatible with Vitis 2022.2. The first

[8] `arm-none-eabi`
[9] Refer to apendix C.

test project for the RFSoC was built using Vitis 2023.1. Soon after, Xilinx released Vitis 2023.2, which represented a major update as the Vitis Unified IDE, launched with version 2023.1, would become the default. In turn, support for the old IDE – renamed as Vitis Classic – would be limited and eventually dropped. In light of this event, an interest arose to explore and test the new environment.

The first important feature is the transition from Makefile to CMake as the base engine for compilation. However, the CMake workflow already showed some problems early on. For instance, source files inside subfolders are not compiled unless the top-level CMakeLists is modified, requiring a custom CMake structure even for simple projects. Furthermore, the file explorer tree in the GUI does not reflect the underlying directory structure of the project, in particular not showing CMake files in subdirectories.

Another promising upgrade is that the internal API is not based on Tcl anymore, but on Python. As the language is more accessible, it is now easier to automate tasks such as creating and building projects dynamically. However, this feature is in an early stage of development, therefore it is functionally still limited and unexpected issues are common to find.

The most significant issue we encountered is caused by the server that Vitis Unified uses to internally run tasks. When a workspace is created from a Python script, a file called `repo.yaml` is created in the root folder. Seemingly, the server binds this file to the client. If the same workspace is later opened in the Unified IDE (i.e. another client), many actions attempt to open this fail, always unsuccessfully. For instance, this prevents the IDE from accessing the BSP configuration, displaying an error message instead (*internal 13: error opening bsp settings because .repo.yaml is being used by another process*). A possible workaround is to delete the file, as it will be automatically generated again. However, doing so from the Python script itself requires waiting for the server to be completely closed, which the API does not directly report.

The last relevant change is related to the BSP. Instead of using a device ID, now a system device tree is created, and peripherals are identified by their base address. This entails major changes in the drivers; the ones provided in the BSP were already adjusted, but their examples or external drivers might not be. In particular, the XRFCLK driver was obsolete, and we had to manually adapt it to use the base address when initializing the I2C driver.

Additionally, the SDT adds a compulsory requirement for the Xiltimer library, which cannot be disabled. A direct implication is that some files are actually replaced, as is the case for the `sleep.h` header. This file's version from the standalone BSP provides functions specific to each processor, whereas the one in Xiltimer is designed to be hardware-independent and thus only the generic functions are available. Moreover, the timer functions are now found in `xiltimer.h` instead of `xtime_l.h`; this is taken into account in the code by means of a conditional import, based on which file exists in the include path[10].

Ultimately, a known issue[11] exists in Vitis Unified 2023.2 by which the RFSoC cannot be detected properly. This has critical implications, such as the FSBL not

[10] This is implemented using the `__has_include` preprocessor operator.

[11] Even though this is a known issue, it is not mentioned in the official list of known issues.

being generated[12] or not being able to program and run the application on the PS.

For this reason, we had to abandon using the Unified IDE to run the test project on the board. We also discarded the idea of linking Vitis' CMake to our own CMake structure for Ionpulse, in favor of using an entirely standalone CMake-based workflow. However, Vitis Unified was used to build the test project via Python and, most importantly, to compile the BSP.

5.3.2 Ionpulse programmer

The group decided some time ago to progressively stop offering support for Vitis in favor of maintaining their own CMake structure for compiling the project. However, doing so requires a custom solution for programming the device. With that in mind, Bahadir Dönmez built the Ionpulse Programmer, a tool that connects to the Zedboard and uploads the bitstream and gateware. Following a similar reasoning, and motivated by the issues with Vitis, we decided to also make use of it to program Ionpulse onto the RFSoC.

The tool allows selecting an available JTAG target, and a serial port to receive the UART output. The user simply needs to provide a file (`.xsa`) with the exported hardware description and the binary file (`.elf`) of the compiled application to be run on the PS. It is worth noting that it uses some files from the Vivado installation which are shared with Vitis; therefore, it is important that the IDE is closed or unexpected issues may occur.

The procedure to program the UltraScale+ is different from the Zynq7000, and so it required a separate implementation. The programming sequence for the RFSoC, executed via XSDB, is the following:

1. Connect to host and list available targets
2. Select target: `APU*`[13]
3. Generate system reset
4. Load bitstream (obtained from the XSA)
5. Run PS initialization script (`psu_init.tcl`, obtained from the XSA)
6. Remove PS-PL power isolation
7. Generate PS-PL reset
8. Configure PSU[14] protection settings
9. Select target: `Cortex-A53* #0`
10. Reset active processor and clear registers
11. Download ELF
12. Optionally resume execution (done on run mode, not on debug)

[12] If we were to flash the RFSoC, we would need to either generate it with Vitis Classic or build it manually from the corresponding example application.

[13] The APU may enter a cache reset state, which is noted by an extension to the name. The star allows accounting for such different possible names.

[14] The ARM APU and RPU are grouped under PSU.

Chapter 6

Testing

The goal for the developed system is to be used in the lab to control experiments, and therefore reliability is of importance. To ensure functionality before a test with an experiment, a number of tests and measurements have been performed which are summarized in this chapter. Additionally, a measurement of the phase noise is reported in appendix D.

6.1 Quench RF and the UART debugger

The `quench_rf` module features a UART debugger that enables standalone testing the DDS system, offering access to `quench_rf` in isolation from the rest of the gateware. It uses the FPGA's PL serial port to establish communication with the PC and can receive a set of instructions and distribute them to the DDS cores, same as if they would have been received from the backplane and deserialized.

Note, however, that the implementation of `quench_rf` alone already requires most clocks to be set up, especially if the RFDC is also used. For this purpose, the CPU was included in the design early in development to program the CLK104 board. A minimal project was prepared in Vitis, based on [8], which only implemented the clock initialization sequence described in section 5.2.2. This project was later used to test MTS, play with the Vitis Unified IDE and to generate the BSP for Ionpulse.

On the PC side, a C++ tool – called the `bp_cmd_generator` – uses part of the Ionpulse code to generate the sequence of instructions for a set of user-defined parameters. One may choose the frequency, phase and amplitude of the signal per channel and per tone. The generated sequence is stored in a binary file `.hex`. An auxiliary Python script is used to connect to the serial port, and send the instructions from the given file to `quench_rf`. A custom protocol is used, with a command being sent first to indicate which action the system should take, e.g. setting an internal signal or expecting a new instruction. To acknowledge reception of the commands, a copy is sent back from the FPGA and compared against the original value.

The first tests for `quench_rf` on the RFSoC are to set up the clocking structure, followed by a set of instructions generated with different parameters. The output of the DAC channels is observed with an oscilloscope to test that all channels could be addressed, and that their frequency, amplitude and phase were correct.

6.2 Breakout board baluns

As mentioned in section 3.3, only one of the four sets of baluns available in the XM655 has a nominal frequency span overlapping with our range of operation. Even the low frequency baluns do not reach the lower end of our spectrum. A quick survey of the market showed that the original "low-frequency" baluns offered the best compromise of bandwidth and insertion loss. Available components with a wider nominal bandwidth were designed for telecommunications applications with much less strict performance requirements, and therefore suffered from higher insertion losses and phase imbalance. On the other hand, baluns targeting frequencies below 1 MHz offered a rather limited range, not suitable for our higher frequencies. Nonetheless, one set of the latter was included in the new design for testing purposes.

We measured the RFSoC output power, including losses through the baluns. The measurement consisted in generating single tone signals, through UART debugger first and from Ionizer later on, and obtaining their power using the spectrum analyzer integrated in an oscilloscope[1]. The DAC was run at half the maximum power (VOP current at 20 mA) – or, equivalently, signal generated at 50 % amplitude – to avoid potential damage to the RFSoC due to reflections of the signal in the baluns, especially for frequencies outside their range of operation.

The results are shown in fig. 6.1. The low frequency baluns (ADT2-1T) performed slightly better over the measured range, although they show bigger losses for higher frequencies. On the custom breakout board the original baluns (TCM2-33WX+) show higher losses, even though the overall behaviour is the same. In principle, measurements above 500 MHz are possible with the system, but we limited measurements to the first Nyquist zone sampling at 1 GSPS. The decay in power as the signal approaches the Nyquist frequency is partially caused by the behaviour of the DACs themselves, even though the sinc filter is enabled [15].

Most amplifiers used for AOMs offer a power of 5 W with a gain +30 dB to +35 dB, meaning we could have a full range of 37 dBm if we are able to provide 7 dBm at the input. With 4 dBm we can use most or all of the range of the amplifier.

6.3 Ionizer test experiments

As a preliminary step before taking the RFSoC to the lab once the software and hardware adaptations were complete, we tested some specific parts or components of the system separately. For that, we used some of the features provided in Ionizer and prepared simple experiments based on the already existing ones.

6.3.1 Constant Frequency Output experiment

One of the test experiments present in the development version of Ionpulse is the "Constant Frequency Output" experiment. Essentially, it allows enabling individual channels and setting frequency, amplitude and phase (FPA) values for them. We implemented a modified version of it that uses solely `quench` channels – two slots (16

[1] Tektronix MDO4054C

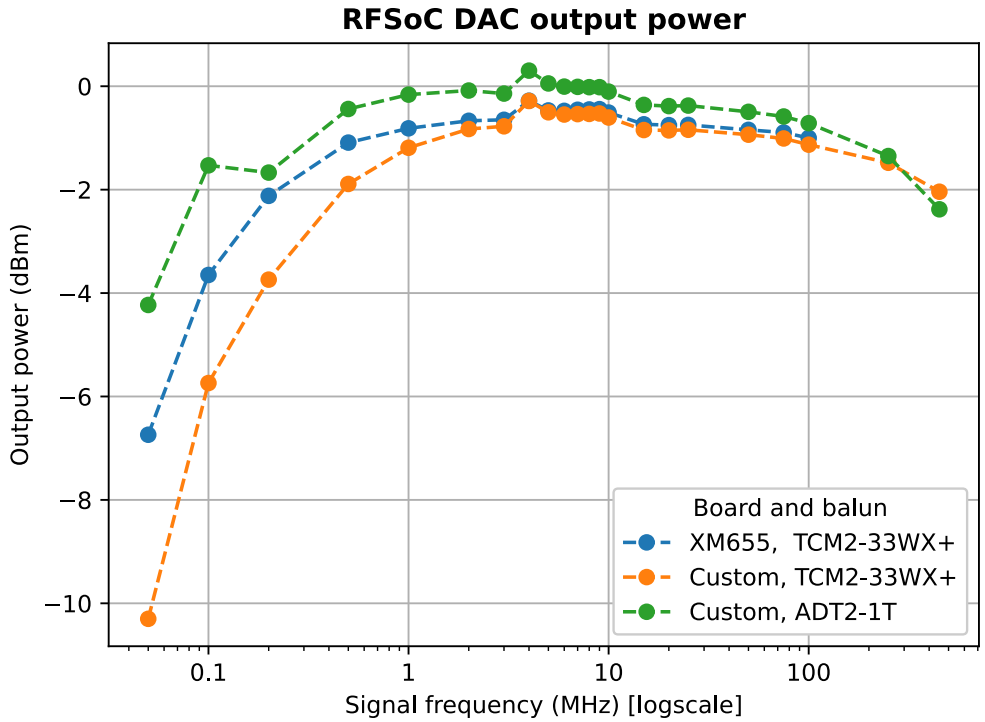


Figure 6.1: Power measured at the output of the baluns, for both breakout boards. Each point corresponds to a single-tone signal generated with at half power (DAC VOP current 40 mA, signal amplitude 50 %). The DAC sampling rate is 1 GSPS, with the sinc filter enabled. Measurement was done on a Tektronix MDO4054C oscilloscope, configured with resolution bandwidth 100 Hz and span 2.5 kHz.

channels) – and outputs an `edge`[2] with a single tone for each of them. This offers an interface to generate simple pulses, which in turn serves as a basis for further tests (e.g. balun power measurements). In other words, it provides a similar functionality to the UART debugger but with a more convenient usage, at the expense of going through a larger portion of the system.

6.3.2 Digital IOs

There are two ways to test the digital outputs[3]. On the one hand, Ionizer already provides a functionality to enable or disable individual channels. On the other hand, when pulses are created with the `bp_dds` interface one can define a target channel mask. The digital outputs selected by this mask are then gated following the pulse shape, with their value being set according to a given pattern.

We extended the Constant Frequency Output experiment to test the second method. Essentially, it creates a pulse in a selected quench channel, using the FPA parameters set for such channel. At the same time, it generates a `wait` on the `bp_dds` driver with one digital output enabled. Thus, this is the simplest possible example using both elements in sync. The index of the `quench` channel and the digital channel can be defined from the GUI, while the pulse time is fixed to 5 µs.

This small experiment was used to measure the delay between the digital output and the RF channels. We generate a digital pulse and a `quench` pulse at the same time, and observe the difference in the scope, as shown in fig. 6.2. The measurement yielded a delay of approximately -0.27 µs, deterministic between consecutive executions. Selecting different `quench`/digital channels remains within less than 1 ns away from that result. Note that ionpulse already applies a correction between both pulses, for which we included a new RFSoC-specific default. This can only be set with a resolution of 4 ns, so a deterministic delay of 3 ns still remains, but that is acceptable for our applications.

Readout from the PMTs is enabled following a similar idea to the digital outputs, where a counter for PMT pulses is gated with a digital signal. Incoming pulses (10 ns long) from the PMT are counted and time-tagged, whereas pulses received outside the temporal window are ignored. We modified an existing experiment to alternatively enable and disable one of the digital outputs, which we connected to the PMT input. This experiment was then executed continuously to verify the reported number of counts. It is relevant to note that the counter in `pulseway` is implemented to detect rising edges, not the level of the input, so simply keeping the digital output at HIGH during the time that the PMT gate is active would always result in zero counts.

6.3.3 Quench Loop Testcase

Most of the advanced features, like forks, are used only in more complex experiments. However, even the simplest experiments that are actually useful in the

[2] In Ionpulse, an `edge` simply toggles the amplitude of the output once. In contrast, a `cap` (or simply a pulse) is comprised of a rising and a falling edge that confers it sort of a baseball cap shape (hence its name).

[3] In the group, digital outputs are colloquially referred to as TTLs. While this terminology is not completely accurate, it is commonly found in the code and in the GUI.

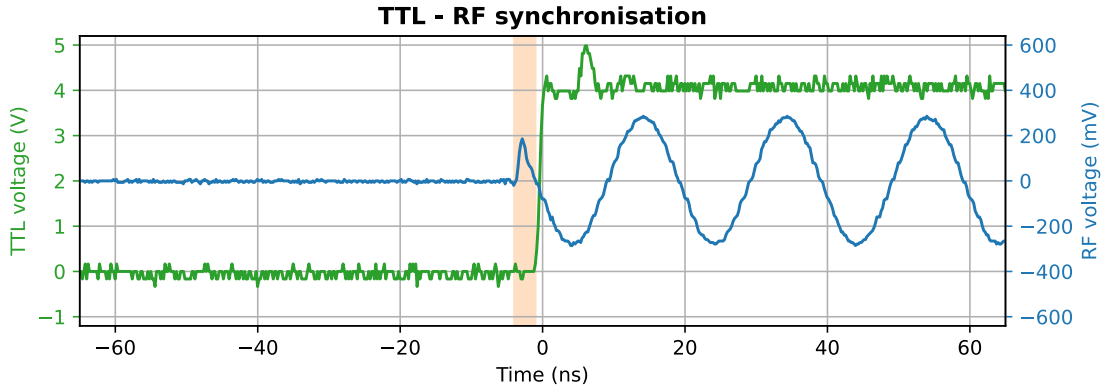


Figure 6.2: Delay between a digital output (TTL, green) and an RF (blue) channel generated synchronously. A compensation of $-0.27\,\mu s$ is already applied by the sequencer. The orange strip highlights the remaining delay, which is $<5\,ns$. A $50\,\Omega$ termination was placed on the digital connection to reduce the overshoot and ringing caused by reflections in the cable, reducing the output voltage to approx. $4\,V$.

lab require sideband cooling, which makes use of loops. Parameterized loops are used for running a given sequence multiple times, but changing a parameter in each execution.

To test this, we created an experiment consisting of a simple loop using only `quench` and digital IOs. In the experiment we prepare loops over an array of amplitude values. In each iteration, a single pulse is generated, after which a `wait` is added before skipping to the next value. The wait time is set to a few seconds, such that the evolution of the pulses could be clearly seen on the scope.

The initial tests with this experiment allowed us to find and fix a bug in the creation of looped tones. It also uncovered an issue in the `quench_ctrl` state machine, limiting us from using a faster AXI clock. The exact cause and resolution of the issue falls outside the scope of this thesis.

6.4 Full RF control in a real ion-trap experiment

To demonstrate the capability of the RFSoC to control a real experiment, we performed an end-to-end test in the *Cryo* setup [3, 36]. This setup, sketched in fig. 6.3, is centered around a cryogenic, segmented surface electrode trap with integrated photonics, designed to operate with calcium ions. The trap features three zones of operations where the ions can be trapped and controlled. For the purpose of our test, we consider only zones 1 and 2, respectively referred to as *working* zone and *loading* zone.

The preceding step before any experiment is trapping an ion. The Ca oven is heated up to emit neutral calcium atoms. In the trap, a two-stage Doppler cooling scheme is continuously applied, where the laser beam is red-detuned from the calcium's 397 nm transition to slow down the atoms. After a certain period of time – of the order of a few hundreads of µs – the detuning is reduced to cool the atoms even further. Two auxiliary beams at 854 nm and 866 nm are used for repumping. Simultaneously, the electrode voltages are configured such that a confining potential is formed on

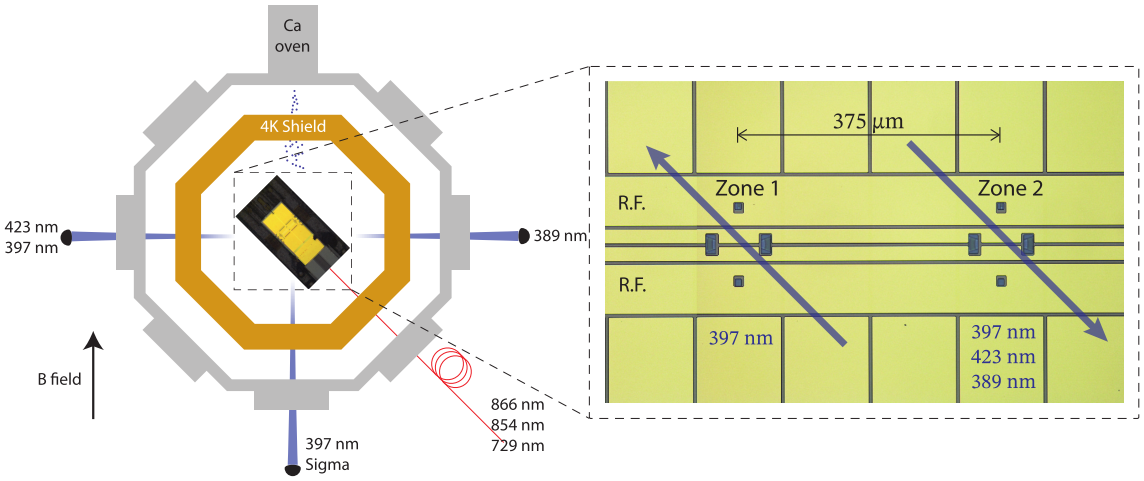


Figure 6.3: Schematic of the *Cryo* setup. The trap is placed inside a cryostat, allowing access for multiple free-space (blue) and fibre (red) lasers. A magnification of the trap highlights the free-space beams available in each of the zones considered in this project. Fibre light is provided to both zones by means of the in-couplers. Figure adapted from [3] and [36]

the loading zone, where two free space photoionization (PI) beams[4] at 423 nm and 389 nm are used to ionize neutral Ca from the oven.

During our experiment a single PMT was used for detection, placed in the working zone. A transport procedure has to be executed to displace the potential well from the loading zone to the working zone, effectively moving any captured ion. In the working zone, qubit readout is done via state-dependent fluorescence using the 397 nm and 866 nm laser light. If an ion is indeed confined, the number of counts received in the PMT is significantly increased, as shown in fig. 6.4. The oven and ionizing beams have to be shut down as soon as the presence of the ion is confirmed.

A common benchmark for new quantum system is to perform a Rabi experiment. Essentially, it consists of preparing the qubit in the ground (or excited) state and driving it resonantly, effectively rotating its state around the X axis in the Bloch sphere representation. The qubit is then expected to exhibit a well-defined periodic behaviour over time, oscillating between the ground and excited state.

In our setup, several AOMs, listed in table 6.1, are used to modulate the laser beams in order to implement the complete sequence depicted in 6.5. The sequence starts by cooling the ions close to their Doppler limit, following the procedure described above. A short pulse of circularly-polarized light at 397 nm is applied to prepare the qubit in the ground state. Next, a pulse of varying duration addresses the qubit transition (729 nm) resonantly. Finally, the PMT is gated and the 397 nm readout beam is switched on. The final state of the ion is determined by applying a threshold to the resulting number of counts, such that we minimize the effect of dark counts, i.e. detections caused by environmental photons or other lasers. Note that, by the end of the sequence, the cooling procedure is activated again, otherwise heating of the ions will lead to higher ion loss rates.

The complete Rabi experiment involves scanning over the qubit (729 nm) pulse du-

[4]The PI beams are controlled by associated digital outputs, toggled manually from Ionizer.

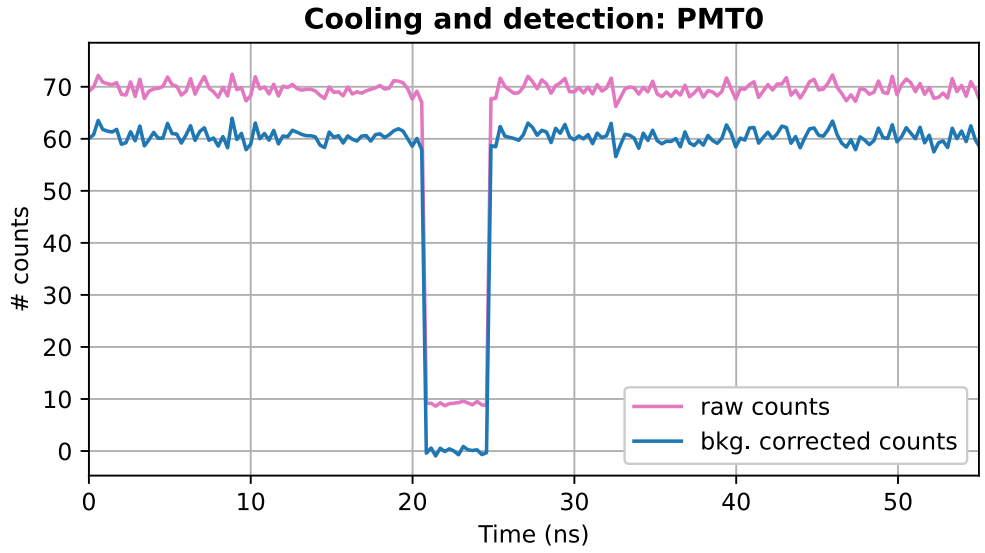


Figure 6.4: Trace of the PMT counts over time, when the detection sequence is run continuously and an ion is trapped. The change in fluorescence corresponds to the transport of the ion from the working zone (where the PMT is placed) to the loading zone.

Table 6.1: Settings for the AOMs used in the Cryo experiment.

Label	Type	Central frequency	Order
397-working	double-pass	80 MHz	1
397-loading	double-pass	80 MHz	1
397-sigma	single-pass	200 MHz	1
854	single-pass	350 MHz	1
866	single-pass	300 MHz	1
729-1	single-pass	320 MHz	-1
729-2	single-pass	150 MHz	1

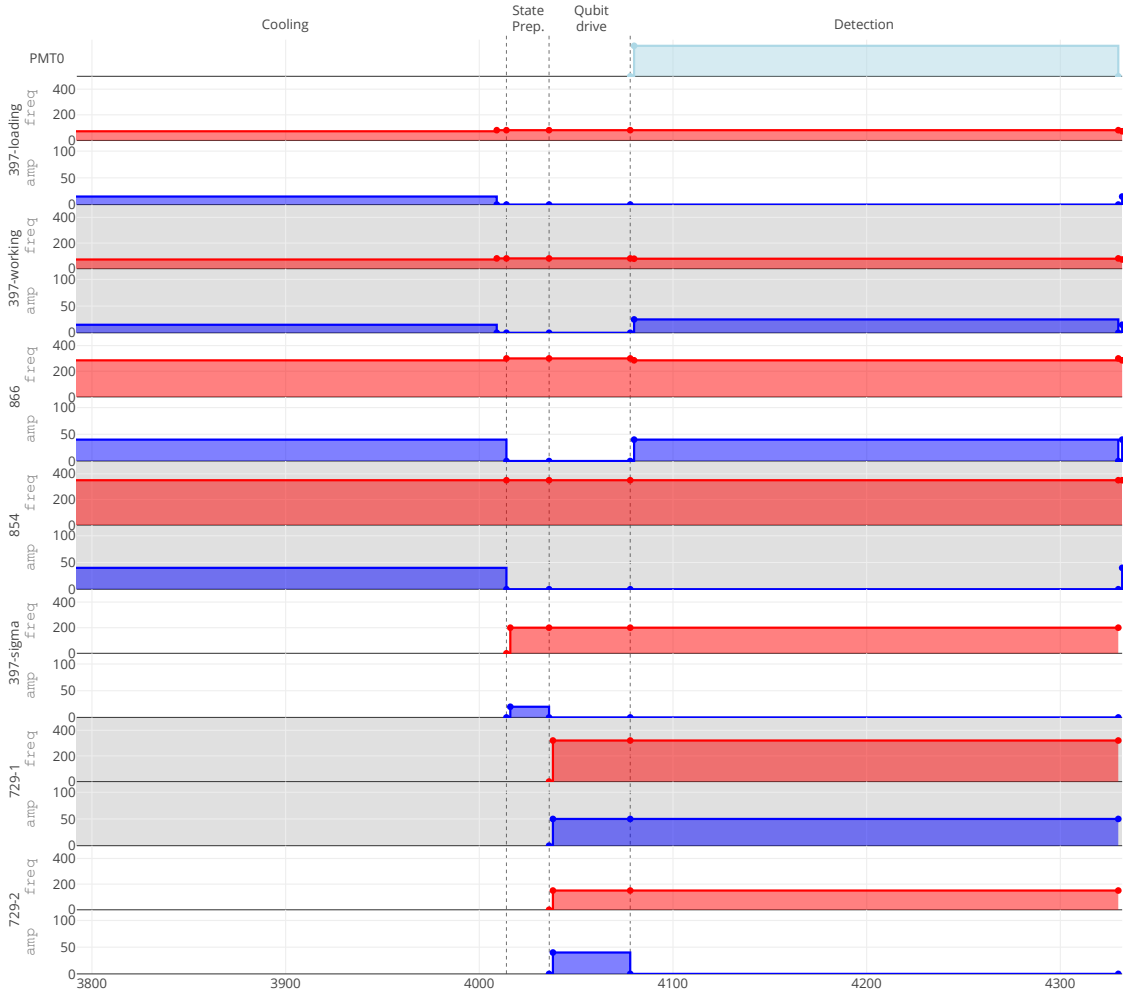


Figure 6.5: RF pulse sequence for the Rabi experiment. Each AOM modulates the corresponding laser, with the exception of the 729 nm laser, where the two AOMs are placed sequentially along the beam. The sequence may be divided in four stages: cooling, state preparation, qubit drive and detection. The first part of the cooling pulse has been skipped here. Note that, by the end of the sequence, the cooling lasers are switched on, such that the cooling procedure is kept active until the next experiment execution.

ration. For each possible value, the sequence is executed multiple times to obtain statistics and estimate the probability of the ion being in either state (note that it is actually in a superposition). The results for a full scan are plotted in fig. 6.6, showcasing the expected periodic behaviour. We highlight that the poor contrast is mainly due to the bad optimization of the experimental parameters. Nonetheless, the observed pattern is an indication that the RFSoC successfully managed to generate all the RF pulses to control the experiment.

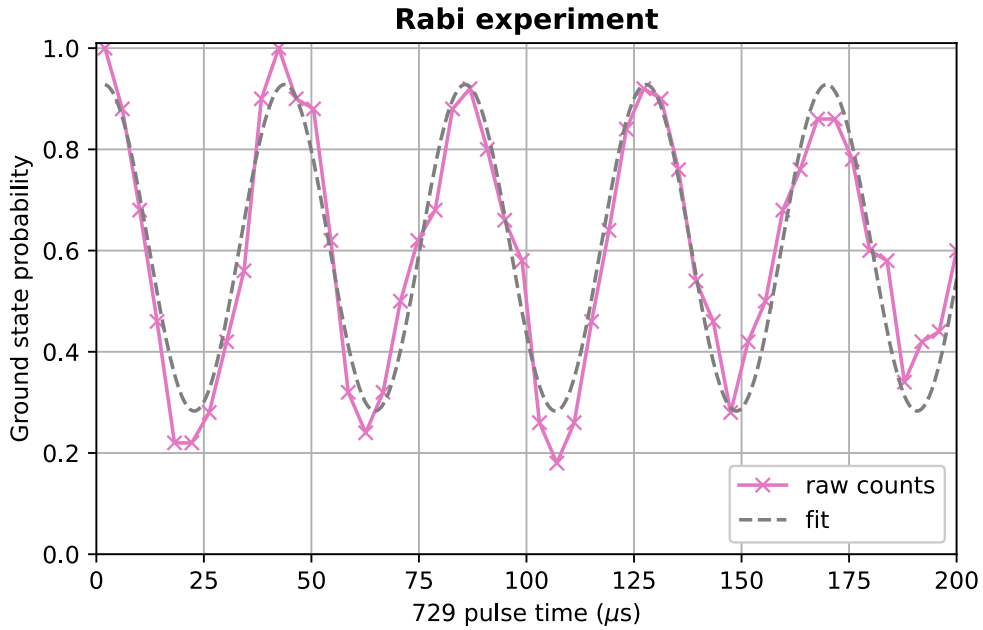


Figure 6.6: Rabi experiment, where the qubit transition (729 nm) is driven resonantly for a variable amount of time. Each data point is the average of 50 shots. The result is fitted to a sinusoidal. The characteristic Rabi oscillations are observed, albeit with limited contrast due to the poor optimization of the experimental parameters.

Chapter 7

Conclusion and outlook

In this work we have successfully carried out the task of adapting the QuENCH control system to the ZCU216 Evaluation Kit, centered around a Zynq UltraScale+ RFSoC Gen3. The original design was built with a modular architecture in mind, were the main controller and the RF generation system were split into two separate FPGAs that communicated through a backplane. We reused some of the submodules to implement the entire design using a single FPGA, thus removing the backplane. We have made the necessary adjustments to make the system compatible with the new hardware, considering the new PL architecture, a new processing unit and an entirely new RF data converter subsystem.

Furthermore, we have adapted the Ionpulse software application to allow it to run on the new processor, featuring a 64-bit ARM architecture. This has meant employing a different compiler and a unique BSP, adjusting the hardware settings and implementing a completely new initialization sequence. The latter has most notably been necessary to set up the external clocking board, which generates all the clock signals for the main board, and to run the multi-tile synchronization procedure that aligns the output from the DACs.

During the development process, the device has undergone a preliminary evaluation, with simple tests focused on key aspects of the design. Among them, we have isolated `quench_rf` to assess DDS generation by means of the UART debugger interface, prepared small pulses from the GUI going through `quench_ctrl`, verified the response of PMT inputs and TTL outputs managed by `pulseway`, and ran non-trivial experiments with looped sequences. Finally, we have been able to use the RFSoC in a real lab setting, accomplishing the significant goal of trapping an ion.

This thesis represents the first stage for a project, mainly establishing a proof of concept that the RFSoC is actually suitable to run experiments. The current implementation offers the same RF capabilities as the current QuENCH system, with the exclusion of the AWG. Future work could include, for instance, the integration of the ADCs (e.g. for feedback loops) or even AWGs. For the latter, one may use the two PL DDR memories offering a bandwidth of 2666 MT/s each, supporting up to 10 channels total with a target sampling rate of 1 GSPS.

In terms of development, the RFSoC is still lacking a simulator for the gateware. One could be implemented with the Verilator tool, which is already used to emulate the FPGA from the Zedboard. That would allow the engineers to debug the gateware

without having to go through the inconvenience of using Vivado's Integrated Logic Analyzer (ILA) on the physical device.

Another important feature of QuENCH is its ability to scale, offering up to 96 RF channels. While a single RFSoC is limited to 16 output channels, it allows for multi-device synchronization following an extended implementation of the MTS protocol. Specifically, an external reference should still be used to align the tiles, but a global deterministic latency would have to be negotiated between all synchronized devices. This would also require the synchronization of the internal clocks, especially the SYSREF, for which the CLK104 offers a SYNC input. The clock-sharing infrastructure will need to be carefully engineered[1].

Going a step further, the system is also not up to its full potential in terms of performance. For most of the development some signals have been deliberately kept at a lower rate, either to reduce implementation times (thus offering a more agile development when working on the gateware) or due to issues in the HDL code that were deemed to complex to tackle in the available time frame. The AXI management clock or the DAC sampling rate are both relevant figures that could be improved. Pushing these to higher clock frequencies would greatly boost performance and potentially open a new range of possible applications for the RFSoC and QuENCH. In particular, our target device could be a potential solution to make TIQI's control system available to other research groups, offering a more plug-and-play and robust solution with respect to the current rack based solution.

To conclude, we have demonstrated that the RFSoC is a suitable candidate for controlling trapped-ion experiments, and that it has the capability to compete with (or, in some aspects, even improve on) the current hardware. We anticipate its usage as experiment control system in our and other labs, and most definitely encourage its future development.

[1]For a large-scale example of a clock synchronization infrastructure, we refer the reader to the White Rabbit Project hosted by CERN [37]

Appendix A

LMK configuration with TICS Pro

According to the CLK104 User Guide [17], there are two methods to configure the clock chips on the board: manually using the BUI tool, or dynamically from the APU. However, neither case offers a user-friendly interface to update specific parameters, but instead rely on an entire configuration set being exported from the Texas Instruments Clocks and Synthesizers (TICS) Pro software [38]. Here, we provide a small guide on how to use this software to obtain the configuration values for the LMK chip, and how to apply them using the APU.

Select device

Upon launching TICS Pro, we must first select the device we want to configure. In our case, go to Select Device > Clock Generator/Jitter Cleaner (Dual Loop) > LMK0482x > LMK04828B.

A menu on the left of the window will show the components that the user can configure: user controls, CLKin and PLLs, SYNC/SYSREF, Clock Outputs and burst mode. It also offers a tool to set predefined modes, a current calculator and an interface to read and write the raw registers.

Initial configuration

With so many configurable options, accidental errors are rather likely to occur, especially for the inexperienced user. It is therefore much safer to start from a predefined configuration that is close to our target, which can be imported from a text file.

In our case we found a suitable configuration in the clock driver (XRFCLK) source code. It sets the DAC reference clock to 250 MHz, the PL clock to 125 MHz and the SYSREF (both analog and digital) to 10 MHz. It also sets the ADC reference clock and the LMX input clock, which we do not care about. To import it, we need to convert the C++ array to the format accepted by TICS Pro:

TICS Pro			C++	
R0 (INIT)	0x000090		{ 0x000090,	
R0	0x000010	⟷	0x000010,	
R2	0x000200		0x000200,	
R3	0x000306		0x000306,	
...			...	}

Registers are labeled according to their address, which corresponds to the first two bytes of their value. The register data is given in the 8 least significant bits.

Update parameters

Once the configuration is imported, we only have to check that the parameters are set to suit our application, and otherwise update them. As an example, the changes we manually applied are listed below, with the full settings being shown in the corresponding figure. For the input and output clocks, one can compare the signal names from TICS Pro to those given in the CLK104 schematics (see fig. 3.1).

- CLKin and PLLs (fig. A.1)
  - Select `CLKin0` (external reference) as clock input for PLL1
  - Disable `CLKin1` (internal oscillator reference) and turn off its multiplexer output.
  - Set `CLKin0` to 10 MHz. While doing so doesn't actually modify any registers, it affects the calculation of the derived frequencies, which is important to make sure all components are operated in the permitted range.
- SYNC/SYSREF (fig. A.2)
  - Set SYSREF frequency to 5 MHz
- Clock Outputs (fig. A.3)
  - Power down `CLKout0_1` (ADC LMX)
  - Power down `CLKout4_5` (DAC LMX)
  - Power down `CLKout10_11` (external reference output)
  - Power down `CLKout12_13` (ADC reference clock)

Apply configuration with XRFCLK

The configuration is now ready to be exported into a text file. This can be used directly in the BUI if desired. For the driver, it needs to be formatted into a C++ array a described above. Registers between R386 and R395 (readback registers) must be removed from the exported values to match the array length expected by the driver.

To place the new array in the driver we opted to extend the `LMK_CKin` array in the `xrfclk_LMK_conf.h` header file, where all the available configurations are listed [18]. Doing so requires the `LMK_FREQ_NUM` constants to be updated accordingly in the top-level header (`xrfclk.h`).

The configuration is uploaded to the LMK chip with

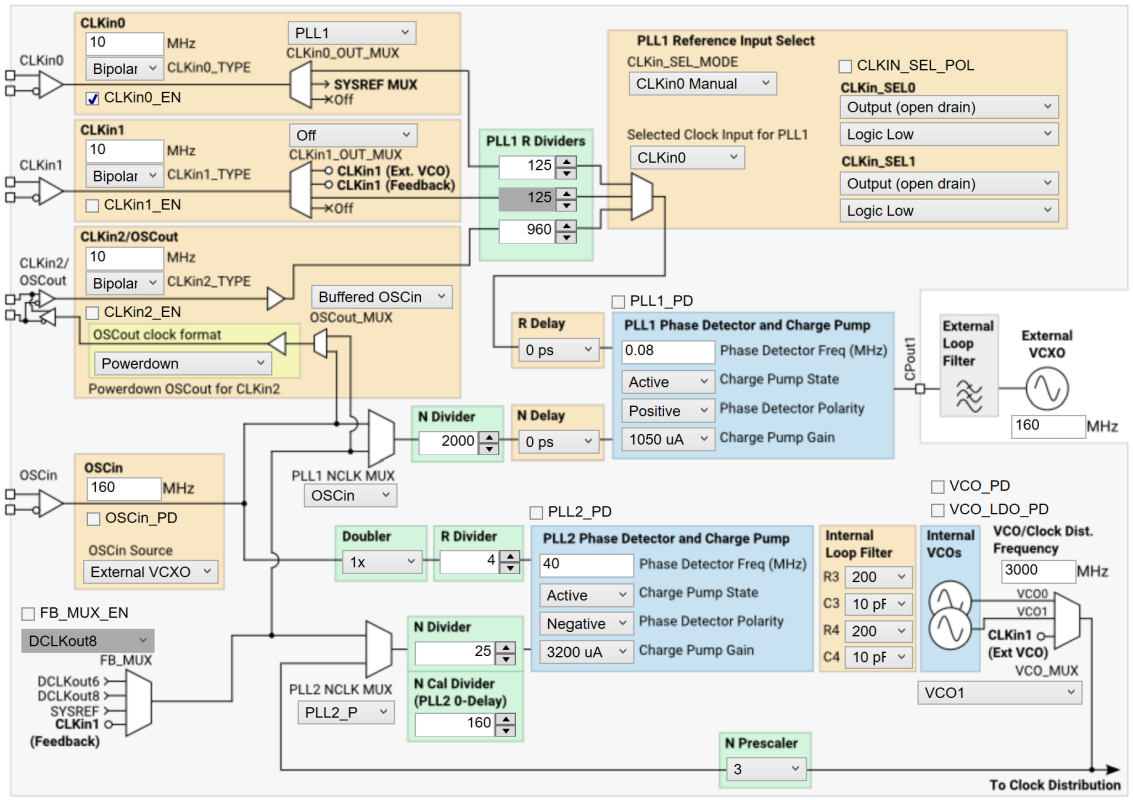


Figure A.1: LMK configuration in TICS Pro: CLKin and PLLs.

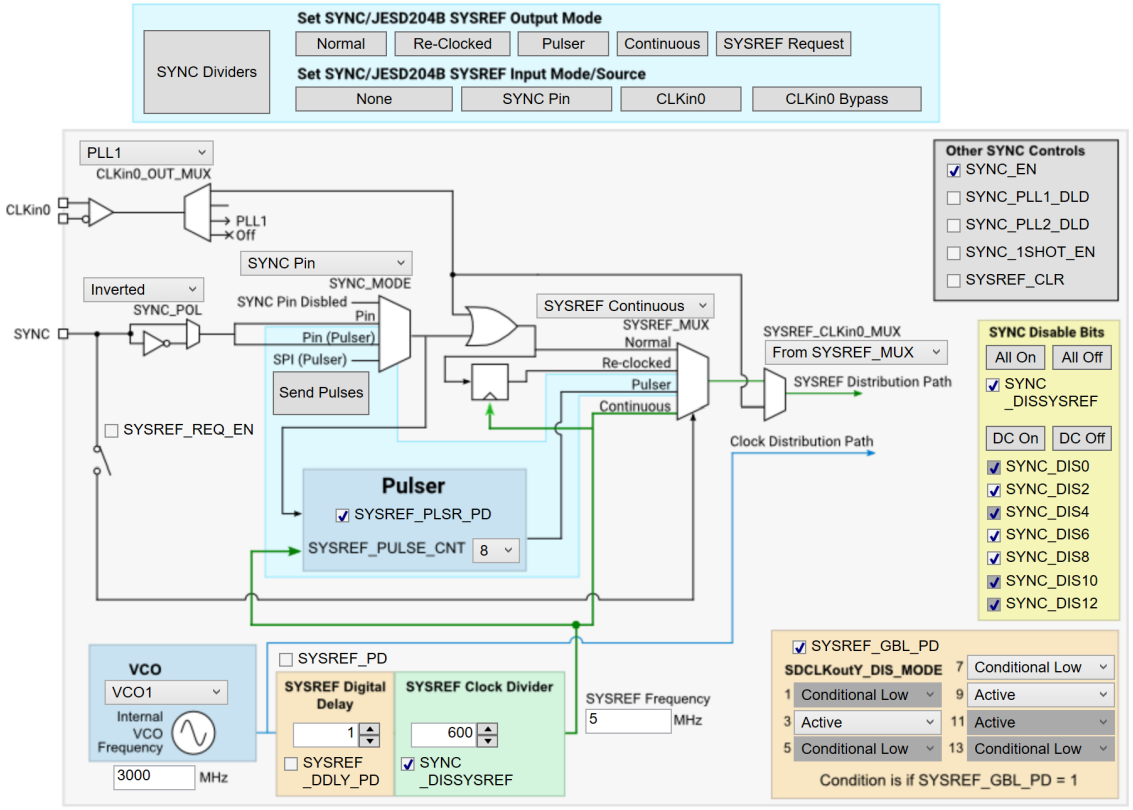


Figure A.2: LMK configuration in TICS Pro: SYNC/SYSREF.

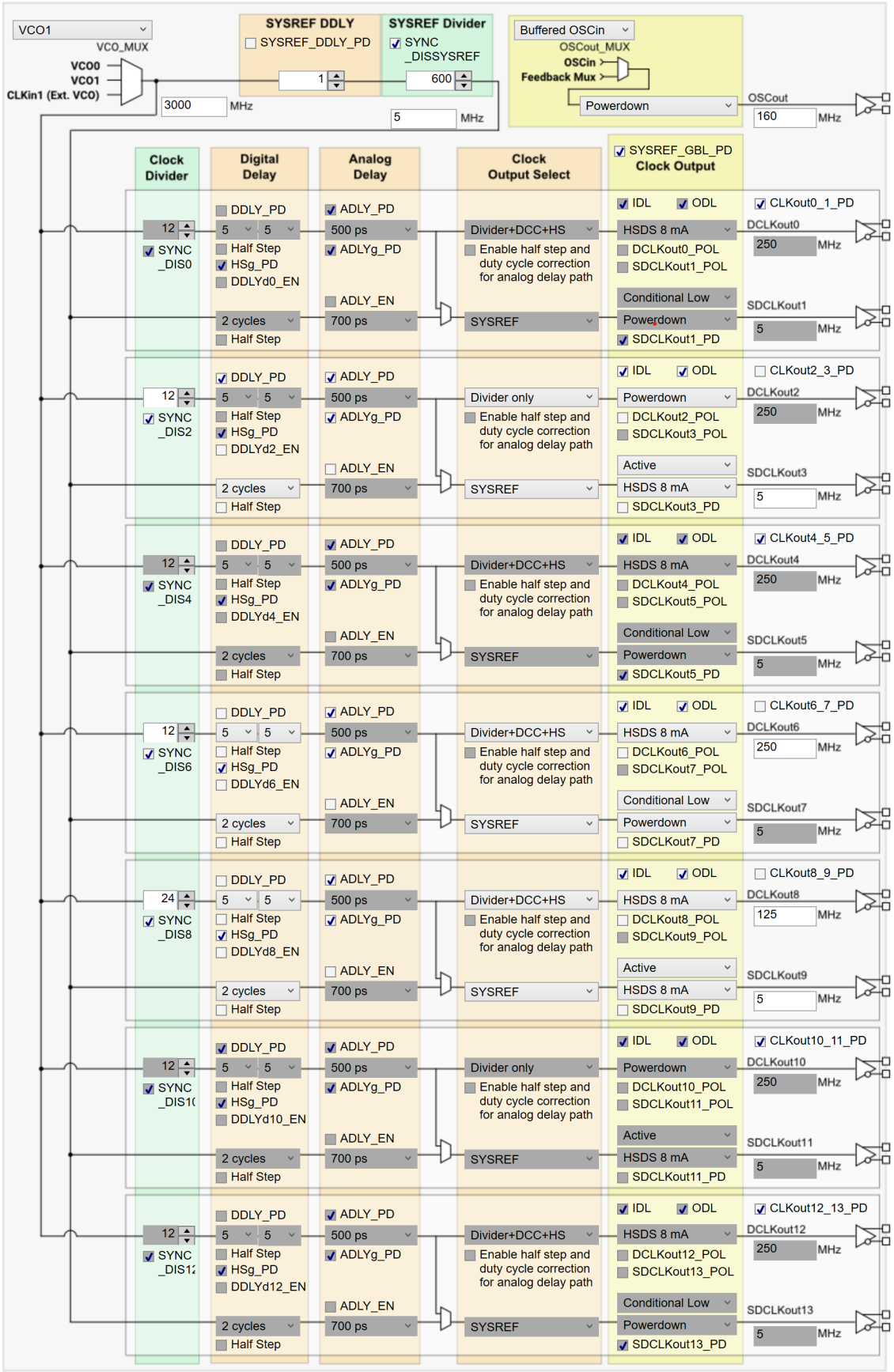


Figure A.3: LMK configuration in TICS Pro: Clock Outputs.

XRFClk_SetConfigOnOneChipFromConfigId(RFCLK_LMK, 2)

where the first arguments determines the chip where the configuration will be uploaded (i.e. LMK or LMX), and the second the index of the target configuration in the aforementioned array.

Optional: update single registers from XRFCLK

While debugging the system it may be convenient to apply small temporary changes to the configuration, for which it could become tedious to follow the same process repeatedly. An alternative solution is to write single registers directly.

To exemplify this, consider the case where we want to test that SYSREF is set up correctly. A dedicated external reference output connector is provided in the CLK104. However, we previously disabled the corresponding LMK output (`CLKout_11`). From fig. A.3 we see that we only need to disable the corresponding clock branch power down, which translates to toggling one bit in register R302. From the APU, we can do that like so:

XRFClk_WriteReg(RFCLK_LMK, 0x012EF0)

Once again, the first parameter identifies the target chip, whereas the second is the new value for the register. Note that the address is already contained within it.

A small window in TICS Pro offers information on the registers, in particular about the meaning of their bits. It is shown either when selecting an entry in the register map or when hovering the mouse over a setting in the diagrams. What is more, whenever a setting is modified (i.e. a button is pressed, checkbox toggled or numerical parameter entered) the application logs which register has been written and what is its new value.

Appendix B

Additional information on auxiliary boards

B.1 RF breakout board: TTL pin mapping

The 32 digital outputs are currently all routed from the RFSoC to the RF breakout board, from which they are distributed between two DSUB-25 connectors. Each DSUB is then planned to take 16 of the digital signals to one of TIQI's TTL boxes. The complete mapping is provided in table B.1. Note that the pinout for the DSUBs was chosen to match the TTL box schematics, but the pin order is not preserved in order to ease the signals routability. Any missing pins not present in the table are grounded, except for pin 15 in each DSUB, which is left unconnected.

B.2 Buffer/trigger board: PMT output via FMC

With the digital IOs in the RF breakout board being occupied by the output channels, and the PMODs being routed through slow a slow level-shifter, the only remaining option to the input the PMT signals is via the FMC. The FMC XM105 Debug Card is then used to interface with the differential buffer/trigger board. Table B.2 contains the pin mapping that we have used.

As described in section 3.1.5, the FMC+ connector in the ZCU216 gives access to 34 differential signals, corresponding to the 34 LA pairs. In principle, we are free to choose (almost) any pins in the LA bus. However, some minor features are offered in different headers in the XM105. For instance, J16 provides two ground and two configurable power pins, which are rather convenient to power the buffer/trigger board. The output voltage is configurable by placing two jumpers on J6. To select 3.3 V, we need one between J6.1 and J6.3, and one between J6.2 and J6.4. On a separate note, we placed two `debug_on` signals — outputs set to constant HIGH for debugging purposes — in J15, where each pin is connected to its own LED.

In the RFSoC, the FMC IO pins are located in bank 66 and 67. These have the particularity that they are powered at a voltage VADJ, which is determined by an internal system controller [10]. If no card is present on the FMC connector, VADJ is set to 1.8 V. When an FMC card is attached, its IIC EEPROM is read to find a voltage supported by both the ZCU216 and the FMC module, within the available

Table B.1: Pin mapping for the digital outputs (TTLs), from their number in Ionpulse to the TTL box, through the custom breakout board. The outputs for the TTL box are numbered left to right, starting from 1.

TTL index	**RFSoC pin**	**Breakout DSUB pin**	**TTL box output**
TTL_00	ADCIO_00	J1.17	5
TTL_01	ADCIO_01	J1.13	4
TTL_02	ADCIO_02	J1.1	8
TTL_03	ADCIO_03	J1.12	3
TTL_04	ADCIO_04	J1.14	7
TTL_05	ADCIO_05	J1.11	1
TTL_06	ADCIO_06	J1.2	16
TTL_07	ADCIO_07	J1.10	2
TTL_08	ADCIO_08	J1.3	15
TTL_09	ADCIO_09	J1.9	9
TTL_10	ADCIO_00	J1.4	14
TTL_11	ADCIO_11	J1.8	10
TTL_12	ADCIO_12	J1.6	12
TTL_13	ADCIO_13	J1.7	11
TTL_14	ADCIO_14	J1.16	6
TTL_15	ADCIO_15	J1.5	13
TTL_16	DACIO_16	J2.17	5
TTL_17	DACIO_17	J2.5	4
TTL_18	DACIO_18	J2.16	8
TTL_19	DACIO_19	J2.4	3
TTL_20	DACIO_20	J2.3	7
TTL_21	DACIO_21	J2.2	1
TTL_22	DACIO_22	J2.14	16
TTL_23	DACIO_23	J2.13	2
TTL_24	DACIO_24	J2.1	15
TTL_25	DACIO_25	J2.9	9
TTL_26	DACIO_26	J2.6	14
TTL_27	DACIO_27	J2.11	10
TTL_28	DACIO_28	J2.7	12
TTL_29	DACIO_29	J2.8	11
TTL_30	DACIO_30	J2.10	6
TTL_31	DACIO_31	J2.12	13

Table B.2: Pin mapping for the PMTs. Beware that the pin pairs for PMT0 and PMT3 in the buffer/trigger board are physically inverted with respect to the XM105 pinout.

Signal name	FMC pin	XM105 pin	Buffer/trigger PMOD pin	Buffer/trigger SMA input
pmt_0_n	LA_29_N	J16.11	3	J1
pmt_0_p	LA_29_P	J16.9	1	
pmt_1_n	LA_31_N	J16.12	2	J2
pmt_1_p	LA_31_P	J16.10	4	
pmt_2_n	LA_30_N	J16.8	6	J3
pmt_2_p	LA_30_P	J16.6	8	
pmt_3_n	LA_28_N	J16.7	7	J4/J8
pmt_3_p	LA_28_P	J16.5	5	
GND		J16.3/J16.4	9/10	
3V3		J16.1/J16.2	11/12	
debug_on[0]	LA_33_N	J15.6 (LED DS1)		
debug_on[1]	LA_33_P	J15.5 (LED DS2)		

choices of 0.0 V, 1.2 V, 1.5 V and 1.8 V. Unfortunately, the XM105 only accepts 2.5 V, so this process fails and VADJ is automatically set to 0.0 V.

The above behaviour can be overridden via the system controller user interface, for which the BUI contains a dedicated page. This tool allows not only to read the current value of VADJ, but also to manually set it to one of the available options. Even further, it lets the user select an on-boot value. Since we do not wish to use any complex component (i.e. LVDS buffer or clock) in the debug card, we simply configure VADJ to be 1.8 V.

An important note when using the FMC is that it sometimes takes control of the JTAG. When that happens the board starts behaving unexpectedly. For instance, Ionpulse is not be able to detect it anymore. It also causes the EFUSE test in the BUI to fail, even if it seems to pass other tests. An effective workaround is to add a jumper between the TDI and TDO pins in the JTAG header of the XM105.

Appendix C

Hiway dummy mode

In the M-ACTION control system, the `hiway` module is responsible for backplane communication. It buffers single-channel data retrieved from the CPU and broadcasts it to the DDS cards via individual `bitumen` cores. For a detailed description, we refer the reader to Vlad Negnevitsky's thesis [6].

In QuENCH, this module is replaced by `quench_ctrl`, which offers a revised functionality. However, there are still some remnants of it in the software. In particular, `bp_dds` was planned to be only an interface with the `hiway` and `pulseway` drivers, but over time an increasing number of features were implemented relying in it. As a result, it became very intertwined with the rest of the code. In an ideal case, we would have preferred to remove this dependency altogether when compiling for the RFSoC, but that would have represented an unfeasible amount of work.

Instead, as a temporary solution, we decided to add a *dummy* mode to the `hiway` driver, which can be selected by setting the `HWY_DUMMY_MODE` in the CMake. Its main purpose is to skip all gateware-related actions, specifically read and write operations to hardware registers. In that sense, it is similar to the existing *x86 emulator* mode[1]. The difference between them is that the emulator mode provides logging capabilities, which the dummy does not.

From the perspective of the `ionpulse_sdk_core`, this change implies that the driver can be initialized without having to specify an actual peripheral address. The dummy mode allows all functions to be called without crashing, but prevents any uses of the undefined peripheral. Hence, no changes in `bp_dds` or in its usage are required.

[1] In fact, it has been implemented so that enabling the emulator mode will automatically activate dummy mode.

Appendix D

Phase noise measurement

Electronic signals are subject to noise, both in amplitude and in phase. In practice, phase noise dominates in terms of power, with amplitude noise contributing significantly less. Phase noise is evaluated by means of the power spectral density of the signal's phase, which can be measured with a signal source analyzer (SSA). It is given in units of dBc/Hz, denoting the noise power in a 1 Hz bandwidth relative to the carrier. A detailed discussion on phase noise, and its effect in trapped-ion systems, can be found in [39].

An in-depth analysis of the phase noise is outside the scope of this thesis. However, a measurement was taken to gain some insight on how the RFSoC compares with other devices employed in the group. The phase noise measurement for the RFSoC and a QuENCH DAC card are given in fig. D.1. As a reference clock, we used the Synchrona14 system clocking device from Analog Devices. Additionally, we include the results from a previous measurement on a Texas Instruments AFE8000EVM card, reported in [39], which used a HMC7044 as the reference clock (the same card included in the Synchrona14).

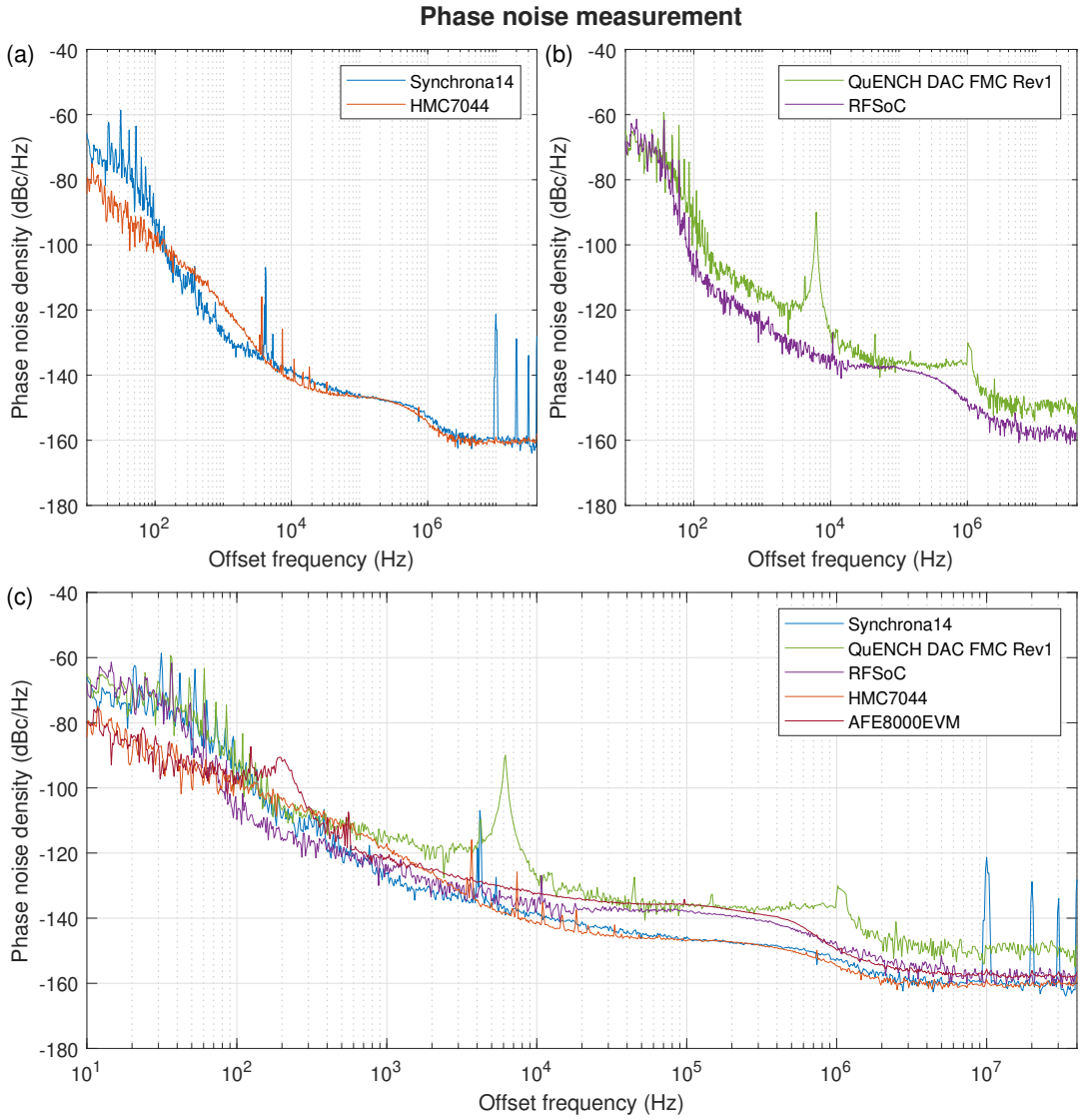


Figure D.1: Comparison of the phase noise for (a) the reference clock devices used during the measurements; (b) a QuENCH DAC and the RFSoC; and (c) these four components as well as the AFE8000EVM DAC card. All measurements were made with a Keysight A5052B SSA, at a carrier frequency of 150 MHz. Here, the reference for the RFSoC and the QuENCH DAC is taken from the Synchrona14, while the AFE8000 uses the HMC7044.

Acronyms

AC alternating current

ADC analog-to-digital converter

AFCK AMC FMC Carrier Kintex

AMBA Arm Advanced Microcontroller Bus Architecture

AOD acousto-optic deflector

AOM acousto-optic modulator

API application programming interface

APU application processing unit

AWG arbitrary waveform generator

AXI Advanced eXtensible Interface

BIT Board Interface Test

BNC Bayonet Neil-Concelman

BRAM block RAM

BSP board support package

BUFG global clock buffer

BUI Board User Interface

CCI cache coherent interconnect

CLB configurable logic interconnect

CMT clock management tile

CPU central processing unit

DAC digital-to-analog converter

DC direct current

DDC digital down-conversion

DDR double data rate

DDS direct digital synthesis

DEATH *Direct Ethernet-adjustable Transport Hardware*

DIP dual in-line package

DMA direct memory access

DSP digital signal processing

DSUB D-subminiature

DUC digital up-conversion

ELF Executable and Linkable Format

EMIO extended multiplexed IO

FIFO First In, First Out

FMC FPGA Mezzanine Card

FMC+ FPGA Mezzanine Card plus

FPA frequency, amplitude and phase

FPD full power domain

FPGA field-programmable gate array

FSBL first stage boot loader

GCIO global clock-capable IO

GPIO general-purpose IO

GSPS giga samples per second

GUI graphical user interface

HD high-density

HDGC HD global clock

HDL hardware description language

HP high-performance

HPC high-performance coherent

HSPC High Serial Pin Connector

I2C Inter-Integrated Circuit

IBUFDS differential input buffer

ICM internal configuration manager

IDDR input double data rate

IDE integrated development environment

ILA Integrated Logic Analyzer

IO input-output

IP intellectual property

LED light-emitting diode

LPD low power domain

LUT look-up table

LVDS low-voltage differential signaling

LVTTL low voltage TTL

lwIP lightweight IP

M-ACTION *Modular Advanced Control of Trapped IONs*

MIG Memory Interface Generator

MIO multiplexed IO

MMCM mixed-mode clock manager

MTS multi-tile synchronization

NCO numerically controlled oscillator

OCM on-chip memory

PC personal computer

PCB printed circuit board

PFD phase frequency detector

PI photoionization

PL programmable logic

PLL phase-locked loop

PMOD peripheral module

PMT photo-muliplier tube

PS processing system

QSPI Quad-SPI

QuENCH *Quantum Experiment Next-generation Control Hub*

RAM random-access memory

RF radio frequency

RF-ADC RF-sampling analog-to-digital converter

RF-DAC RF-sampling digital-to-analog converter

RFDC RF Data Converter

RGB red, green and blue

ROM read only memory

RPU real-time processing unit

SDK software development kit

SDT system device tree

SLM spatial light modulator

SMA subminiature version A

SoC system-on-chip

SPI serial peripheral interface

SSA signal source analyzer

SSBL second stage bootloader

SSMP subminiature push-in

stdout standard output stream

Tcl Tool Command Language

TCP Transmission Control Protocol

TCXO temperature-compensated crystal oscillator

TICS Texas Instruments Clocks and Synthesizers

TIQI Trapped Ion Quantum Information

TTC triple-timer counter

TTL transistor-transistor logic

UART universal asynchronours receiver/transmitter

USB universal serial bus

VCO voltage-controlled oscillator

VOP Variable Ouput Power

XDC Xilinx Design Constraints

XOR exclusive or

XSA Xilinx Support Archive

XSDB Xilinx System Debugger

References

[1] H. Häffner, C.F. Roos, and R. Blatt. "Quantum computing with trapped ions". In: *Physics Reports* 469.4 (2008), pp. 155–203. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2008.09.003.

[2] Steven A Moses et al. "A race-track trapped-ion quantum processor". In: *Physical Review X* 13.4 (2023), p. 041052. DOI: 10.1103/PhysRevX.13.041052.

[3] Alfredo Ricci Vasquez et al. "Control of an atomic quadrupole transition in a phase-stable standing wave". In: *Physical Review Letters* 130.13 (2023), p. 133201. DOI: 10.1103/PhysRevLett.130.133201.

[4] Maciej Malinowski. "Unitary and dissipative trapped-ion entanglement using integrated optics". PhD thesis. ETH Zurich, 2021. DOI: 10.3929/ethz-b-000516613.

[5] Dolev Bluvstein et al. "Logical quantum processor based on reconfigurable atom arrays". In: *Nature* 626.7997 (2024), pp. 58–65. DOI: 10.1038/s41586-023-06927-3.

[6] Vlad Negnevitsky. "Feedback-stabilised quantum states in a mixed-species ion system". PhD thesis. ETH Zurich, 2018. DOI: 10.3929/ethz-b-000295923.

[7] AMD. *AMD Zynq UltraScale+ RFSoCs*. URL: https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-rfsoc.html (visited on 05/31/2024).

[8] Ali Doruk Bekatli and Sevket Baturay. "A 16 channel DDS implementation for the ZCU216 RFSoC evaluation board". Bachelor's thesis. ETH Zurich, 2022.

[9] Martin Stadler. "Classical control for trapped-ion quantum physics". PhD thesis. ETH Zurich. To be published.

[10] *ZCU216 Evaluation Board User Guide*. UG1390. v1.2. AMD. Dec. 2023.

[11] *Zynq UltraScale+ RFSoC Data Sheet: Overview*. DS889. v1.14. AMD. June 2023.

[12] *Learn the architecture - An introduction to AMBA AXI*. 102202_0300_03_en. Issue 03. Arm. Oct. 2022.

[13] *Vivado Design Suite: AXI Reference Guide*. UG1037. v4.0. Xilinx. July 2017.

[14] *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics*. DS926. v1.12. AMD. May 2023.

[15] *Zynq UltraScale+ RFSoC RF Data Converter v2.6 Gen 1/2/3/DFE LogiCore IP Product Guide*. PG269. v2.6. AMD. May 2024.

[16] Nekija Dzemaili. "A reliable booting system for Zynq Ultrascale+ MPSoC devices". Bachelor's thesis. HU University of Applied Sciences Utrecht, 2021.

[17] *CLK104 RF Clock Add-on Card User Guide*. UG1437. v1.1. Xilinx. Aug. 2022.

[18] Xilinx. *Zynq UltraScale+ RFSoC Gen3: Programming the CLK104 module from the RFSoC APU*. URL: https://support.xilinx.com/s/article/1192842 (visited on 05/21/2024).
[19] *Zynq UltraScale+ MPSoC Processing System v3.5 LogiCore IP Product Guide*. PG201. v3.5. AMD. June 2023.
[20] *Zynq UltraScale+ Device Technical Reference Manual*. UG1085. v2.4. AMD. Dec. 2023.
[21] *7 Series FPGAs SelectIO Resources User Guide*. UG471. v1.10. Xilinx. May 2018.
[22] *Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide*. UG953. v2024.1. AMD. May 2024.
[23] *UltraScale Architecture SelectIO Resources User Guide*. UG571. v1.15. AMD. Mar. 2023.
[24] *UltraScale Architecture Libraries Guide*. UG974. v2024.1. AMD. May 2024.
[25] *AMBA AXI Protocol Specification*. ARM IHI 0022. Issue K. Arm. Sept. 2023.
[26] *Vivado Design Suite User Guide Using Constraints*. UG903. v2022.1. AMD. June 2022.
[27] *TXS0108E 8-Bit Bi-Directional, Level-Shifting, Voltage Translator for Open-Drain and Push-Pull Applications*. SCES642K. Rev. K. Texas Instruments. Apr. 2024.
[28] *Photon counting heads H10682 series*. TPMO1075E03. Hamamatsu Photonics. Mar. 2024.
[29] *FMX XM105 Debug Card User Guide*. UG537. v1.3. Xilinx. June 2011.
[30] *UltraScale Architecture Clocking Resources User Guide*. UG572. v1.10.2. AMD. Feb. 2023.
[31] *UltraFast Design Methodology Guide for FPGAs and SoCs*. UG949. v2023.2. AMD. Nov. 2023.
[32] *ARM cortex-A Series Programmer's Guide for ARMv8-A*. ARM DEN0024A. Issue A. arm. Mar. 2015.
[33] Xilinx. *68962 - How can I get a Zynq MPSoC PS pl_resetnx port's control address?* URL: https://support.xilinx.com/s/article/68962 (visited on 05/28/2024).
[34] *Zynq UltraScale+ Devices Register Reference*. UG1087. v1.10. AMD. Mar. 2024.
[35] Xilinx. *Getting in Synch with RF Data Converters*. URL: https://support.xilinx.com/s/article/1071111 (visited on 05/31/2024).
[36] Carmelo Mordini et al. "Multi-zone trapped-ion qubit control in an integrated photonics QCCD device". In: *arXiv preprint arXiv:2401.18056* (2024). DOI: 10.48550/arXiv.2401.18056.
[37] CERN. *The White Rabbit Project*. URL: https://white-rabbit.web.cern.ch (visited on 05/31/2024).
[38] Texas Instruments. *TICS Pro Software*. URL: https://www.ti.com/tool/TICSPRO-SW (visited on 05/30/2024).
[39] Simon Dörrer. "Impact of RF noise on trapped-ion quantum gate fidelity". MA thesis. ETH Zurich, 2023.