©Copyright 2025

Haochen Yu

Programmable System for Laser Control of Trapped-Ion Experiments

Haochen Yu

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2025

Committee:

Scott Hauck

Sara Mouradian

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

Programmable System for Laser Control of Trapped-Ion Experiments

Haochen Yu

Chair of the Supervisory Committee:
Scott Hauck
Department of Electrical and Computer Engineering

Modern quantum computing experiments demand exceptional precision, especially when controlling laser pulses for trapped-ion systems. Recent optical breakthroughs now pack dozens of channels onto a single chip, enabling efficient multichannel laser control. However, the supporting electronics often lack the necessary timing accuracy and flexibility. This shortfall can hinder progress and frustrate researchers.

This thesis presents an FPGA-based laser control system on a Xilinx Zynq platform. A key module generates precise pulse waveforms and integrates into a system that delivers 32 pulsed and 32 static voltage signals. By leveraging dual-port block memories, state machines, and high-speed interfaces, the system achieves sub-microsecond timing with low resource usage. Simulation and board testing confirm accurate pulse generation while highlighting opportunities for enhanced precision and improved error handling. Overall, this work offers a reliable, scalable, and cost-effective solution for advanced FPGA applications in quantum control.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I extend my deepest gratitude to everyone who has supported me on this journey. I am especially thankful to my advisor, Professor Scott Hauck, whose guidance and unwavering support have been indispensable. Working in his laboratory provided me with incredible opportunities and invaluable resources that enriched my personal, academic, and professional growth.

I also appreciate Professor Sara Mouradian, a valued member of my thesis committee, has been a driving force behind this project. Her strategic vision and deep understanding of its foundations have consistently propelled innovation and advanced its potential.

A sincere thank the ACME lab team for their invaluable help during the process. Special thanks to Geoffrey Jones for initiating the design. His insights and technical effort have shaped every corner of the system.

I truly appreciate the support of my friends and family. I am thank to Buke Hu from QT3 Laboratory for providing essential publications on the project's application, and to ACME alumni Jeffery Xu for his technical guidance during the design phase. Lastly, I owe my deepest appreciation to my mother, Dr. Li Yang, whose constant encouragement has been a cornerstone of my journey.

Chapter 1. Introduction

Trapped-ion quantum computing stands at the forefront of modern quantum technology, promising unparalleled computational capabilities through the manipulation of individual atomic ions. A cornerstone of these systems is the laser control unit, which must deliver precisely tailored optical pulses with extreme timing accuracy [8]. Motivated by recent advances in both quantum experiments and control hardware, this thesis presents a novel laser control system specifically designed for trapped-ion quantum computing [6].

This hardware design, implemented on a Xilinx Zynq FPGA platform, harnesses the combined power of high-speed and low-speed digital-to-analog converters (DACs) to achieve synchronous control across 32 discrete laser channels. With a frequency of 100 MHz, the system meets the rigorous temporal precision required for high-fidelity qubit manipulation. Such demanding performance is paramount, as even slight deviations in timing can significantly impact the accuracy and repeatability of quantum operations [8].

The design approach is both well-structured and modular, ensuring a robust foundation for development. Initially, a single pulse channel is developed and rigorously evaluated as a proof-of-concept. This initial validation demonstrates the feasibility and effectiveness of the design. Once the single channel is successfully validated, it is scaled into a comprehensive system by integrating custom programmable logic blocks. These blocks are designed to efficiently deliver stable DC voltage control and generate dynamic, high-frequency pulsed waveforms. This capability ensures versatile and precise signal manipulation. The scalability

and modularity of the design not only meet current experimental demands but also provide a framework that can adapt to future challenges in quantum computing [9]. This adaptability is particularly important as it allows the system to evolve with advancements in the field, ensuring long-term relevance and utility.

Central to the system's performance is its effective exploitation of the FPGA's built-in features. By incorporating custom memory IP, high-speed internal communication interfaces, and an embedded ARM CPU core, the design achieves remarkable throughput and performance. This careful coordination of resources ensures precise hardware timing and optimal utilization. This strategic use of FPGA capabilities underscores the system's ability to deliver high performance and reliability.

Chapter 2. Background

2.1 Laser Trapped-Ion Experiment

Trapped-ion quantum systems have emerged as a leading candidate for quantum computing platforms due to their outstanding coherence times, high-fidelity quantum gates, and promising scalability. In trapped-ion systems, charged atoms confined by electric fields are precisely controlled and manipulated using laser beams. These lasers serve instrumental roles, including initializing quantum states, implementing quantum gates, and performing state measurements [3].

However, traditional approaches for optical hardware tend to be bulky, expensive, and power-intensive, posing severe limitations on scalability. As the number of qubits in a trapped-ion quantum computing system grows, these challenges become more pronounced [5]. Compact, efficient, and rapid modulation capabilities are thus needed to scale quantum computing platforms beyond small-scale demonstrations.

Recent advancements in photonic integrated circuits (PICs) have substantially addressed these challenges [4]. These PICs are miniaturized, enabling direct integration with ion-trap chips, thereby dramatically reducing optical losses, system complexity, and the overall footprint. Crucially, integrated modulators facilitate modulation at very high speeds—on the order of nanoseconds—which matches the stringent timing requirements of quantum operations [3].

To fully utilize the capabilities of these advanced PICs, corresponding electronic control

systems capable of generating precisely timed, flexible modulation signals at around 100 MHz resolution are necessary. The 100 MHz resolution provided by our system ensures that the modulation signals can be precisely timed and shaped, directly corresponding to the stringent timing requirements of quantum gates that often operate on microsecond timescales. Having 32 synchronous dynamic voltage channels is necessary because it allows the simultaneous manipulation of multiple qubits with coherent precision. The precise synchronization of these channels ensures minimal timing jitter and improved gate fidelity, which is indispensable for scaling quantum computing systems [8].

2.2 Hardware Description

To meet the demands of advanced control electronics, this project utilizes a Field Programmable Gate Array (FPGA). FPGAs excel at high-speed, low-latency parallel processing, enabling the simultaneous control of multiple channels and the implementation of flexible, efficient digital signal processing algorithms. These capabilities make FPGAs particularly suited for managing complex experimental setups that demand precise timing and real-time adaptability. This project employs the Xilinx Zynq UltraScale+ ZCU102 Evaluation Board, a platform chosen for its remarkable computational resources and flexible high-speed interfacing capabilities.

The ZCU102 board (Figure 2.1) is integral for generating precise modulation signals required by quantum lasers. It features a range of peripherals, including PMOD interfaces that support low-speed digital-to-analog converters (DACs) via SPI protocols, and two FMC

HPC connectors for high-speed DACs. The core of this FPGA platform is the Xilinx Zynq architecture, which combines a Processing System (PS) with Programmable Logic (PL) [10].

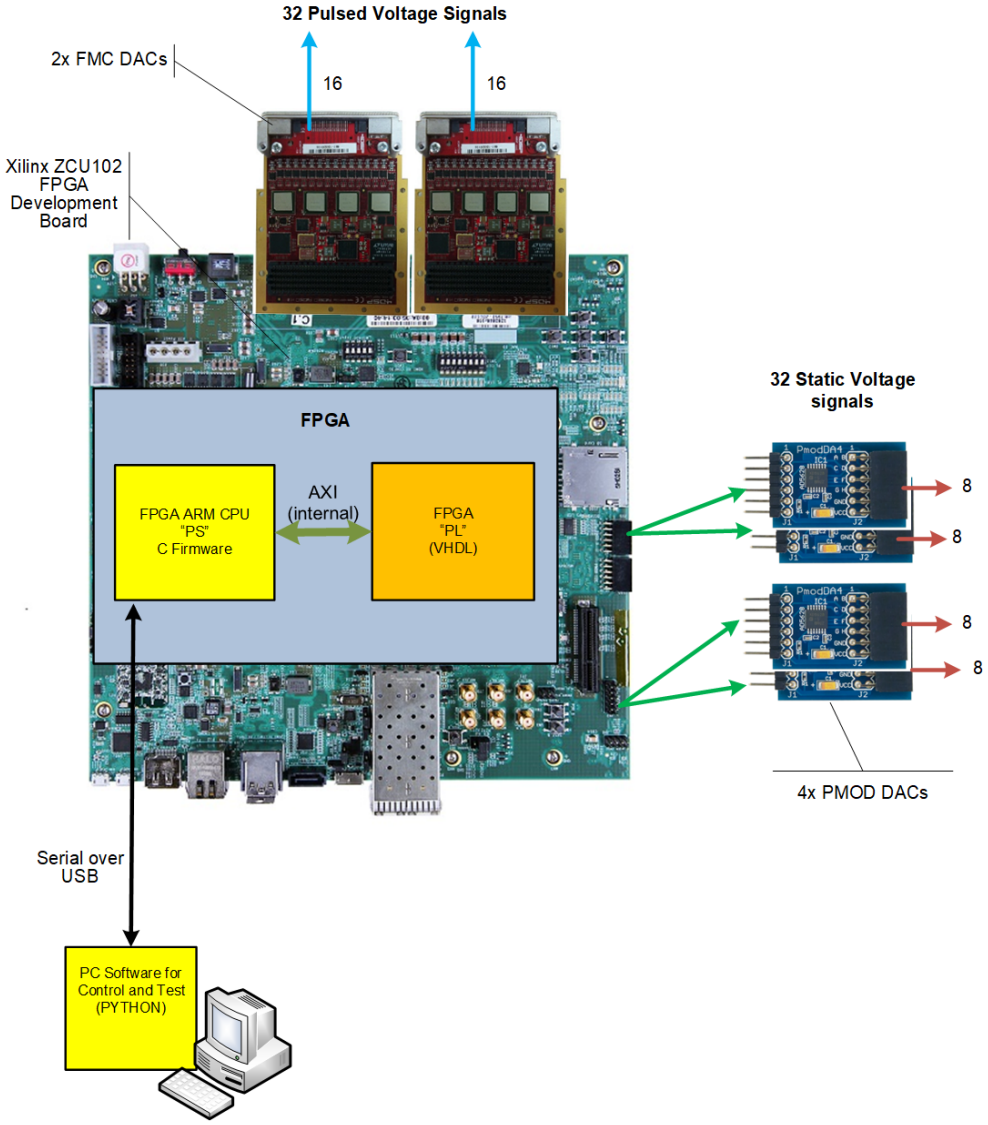

Figure 2.1: The FPGA board with necessary peripherals.

The PS is powered by a quad-core Arm® Cortex®-A53 processor, capable of running real-time processing applications. It incorporates multiple peripherals, such as DDR4 memory

for application storage and an Advanced eXtensible Interface (AXI) that connects the PS to other components, including the programmable logic. The PL, which is the FPGA itself, contains logic cells, flip-flops, and block RAMs (BRAM) [10]. It provides ample capacity for storing intermediate data.

This project leverages programmable logic to build custom hardware designs that serve the unique demands of quantum laser experiments. The FPGA's dedicated logic circuits and integrated high-speed interfaces enable quick, reliable communication with external modules. This capability maintains accurate control and precise timing. General-purpose I/O options, such as PMOD and FMC connectors, further simplify the experimental setup. These connectors offer flexible interfacing, allowing for both low-speed and high-speed connections that scale to the experiment's complexity without sacrificing performance.

In addition to the hardware features, a Python interface abstracts the low-level hardware details, streamlines testing, and integration with the FPGA platform. Indicated in Figure 2.1, a custom C-based program running in the processing system translates Python-generated commands into data that the FPGA can efficiently process. This layered approach bridges high-level control with low-level hardware performance, ensuring that experimental algorithms and rapid prototyping can be achieved.

2.3 Pulse Sequence

In high-fidelity quantum computing and atomic manipulation, pulsed sequences are pivotal for delivering precise, accurate, and stable outputs to photonic integrated circuits [4]. These

laser pulses (Figure 2.2) are designed with a controlled shape that consists of three distinct phases: rise, sustain, and fall. Each phase ensures quantum operations are performed reliably.

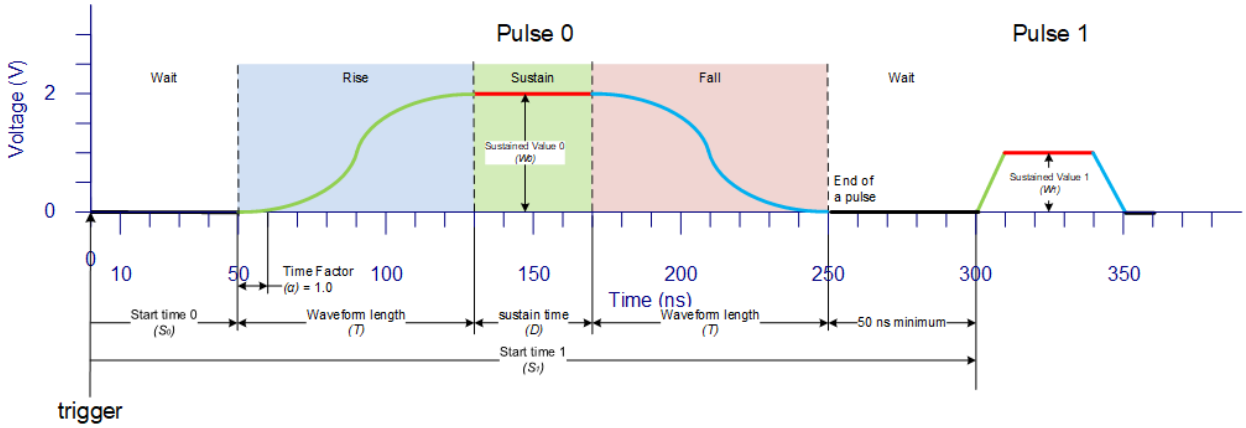

Figure 2.2: Example of the pulse sequence. Both the time and gain factors are set to 1

During the rise phase, the laser intensity increases to its target level. A well-controlled rise minimizes errors and ensures that the experiment operate exactly when needed. The sustain phase follows, during which the laser maintains a constant optical intensity. This steady state is paramount because any fluctuation can compromise the fidelity of quantum gates by introducing operational errors [3]. Finally, the fall phase allows the laser power to decrease smoothly. Maintaining symmetry between the rise and fall phases is critical for consistency and repeatability. In this design, the fall essentially becomes the mirror image of the rise. The rise is a programmable waveform pattern, while the sustain is a flat region with programmable length. User settings specify additional parameters such as start time, sustain time, and waveform length, as described in Figure 2.2. These configurations enable the module to adjust pulse characteristics programmatically without redefining the waveforms.

In many cases, a pulse is produced by scaling a common base waveform using programmable factors for both time and amplitude. However, when a pulse sequence includes pulses with different time and amplitude features, the configuration selects the distinct waveforms for each pulse. This approach ensures accurate and flexible control over the behavior of the entire sequence.

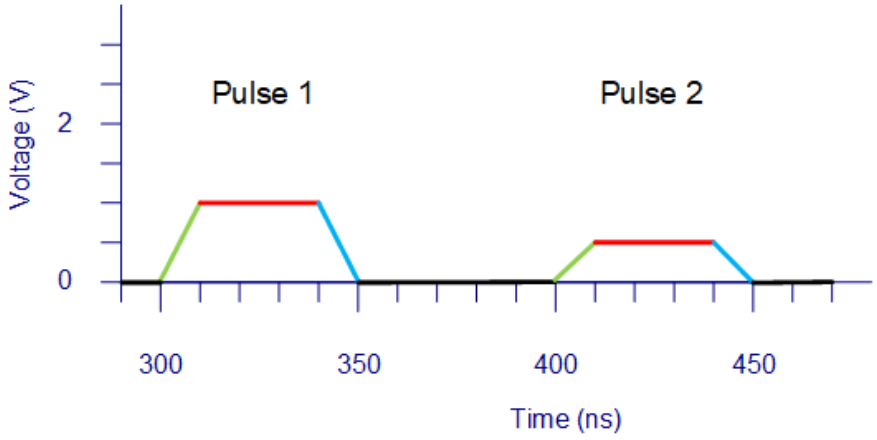

Figure 2.3: Two pulses with pulse 2 having half the amplitude of pulse 1.

For instance, the rising edge of pulse 0 in Figure 2.2 follows a curved profile while pulse 1 increases linearly. Since these profiles cannot be interconverted by scaling alone, the user must define two unique waveforms and assign each to the respective pulse configurations. Furthermore, suppose a subsequent pulse, such as pulse 2, is intended to have half the amplitude of pulse 1 while retaining similar waveform characteristics (Figure 2.3). In that case, the configuration should specify that pulse 2 uses the waveform of pulse 1 with a gain factor of 0.5. This structured configuration approach allows the module to adjust pulse characteristics without redefining fundamental waveforms, ensuring precise control over the

entire sequence's behavior.

Table 2.1: A single pulse's output at various time

Time	Phase	Output
0	Wait	0
	⋮	
$S_0 - 1$	Wait	0
S_0	Rise	$A_0 f_0(0)$
$S_0 + \alpha$	Rise	$A_0 f_0(\frac{\alpha}{T_0})$
	⋮	
$S_0 + T_0$	Rise	$A_0 f_0(\frac{T_0 - 1}{T_0})$
$S_0 + T_0 + 1$	Sustain	W_0
	⋮	
$S_0 + T_0 + D_0$	Sustain	W_0
$S_0 + T_0 + D_0 + 1$	Fall	$A_0 f_0(\frac{T_0 - 1}{T_0})$
	⋮	
$S_0 + T_0 + D_0 + T_0$	Fall	$A_0 f_0(0)$
$S_0 + T_0 + D_0 + T_0 + 1$	Wait	0

For an unscaled pulse 0 from Figure 2.2, with a waveform $f_0(x)$ (green line), can be generated using the equations in Table 2.1 at various times and phases. Here, S denotes the pulse start time, T is the waveform length, α is the time factor, D is the sustain time, A is the gain factor, and W is the output value during the sustain phase.

Chapter 3. Block Architecture

3.1 Functional Description

To fulfill the requirement to generate a pulse sequence, the pulse channel module uses Xilinx block RAM IPs, a state machine, and an interface to drive an external 16-bit DAC. One of the block RAMs is dedicated to the waveform table memory, where unique digital representations of pulses' waveforms are stored. A separate block RAM, known as the pulse definition table memory, holds configuration data for each pulse, including start time, sustain time, time and gain factors, waveform length, and a waveform identifier that links to the corresponding waveform stored in the waveform table memory. The state machine reads from both memories to select the appropriate waveform and configuration of a pulse, thereby driving the DAC to produce the desired pulse sequence. The area highlighted in purple in Figure 3.1 illustrates the relative placement of these two memories within the module, with further details discussed in later sections.

All waveform data and parameters are loaded into memory via a word-addressable interface, as highlighted by the green arrow in Figure 3.1. This interface supports both configuration and monitoring, allowing the module to select between the RAM blocks or perform real-time diagnostics and debugging. The design enables rapid reconfiguration and continuous system oversight, which assists in maintaining high performance during pulse sequence generation.

Error handling is integrated into the module. Dedicated error signals are generated during

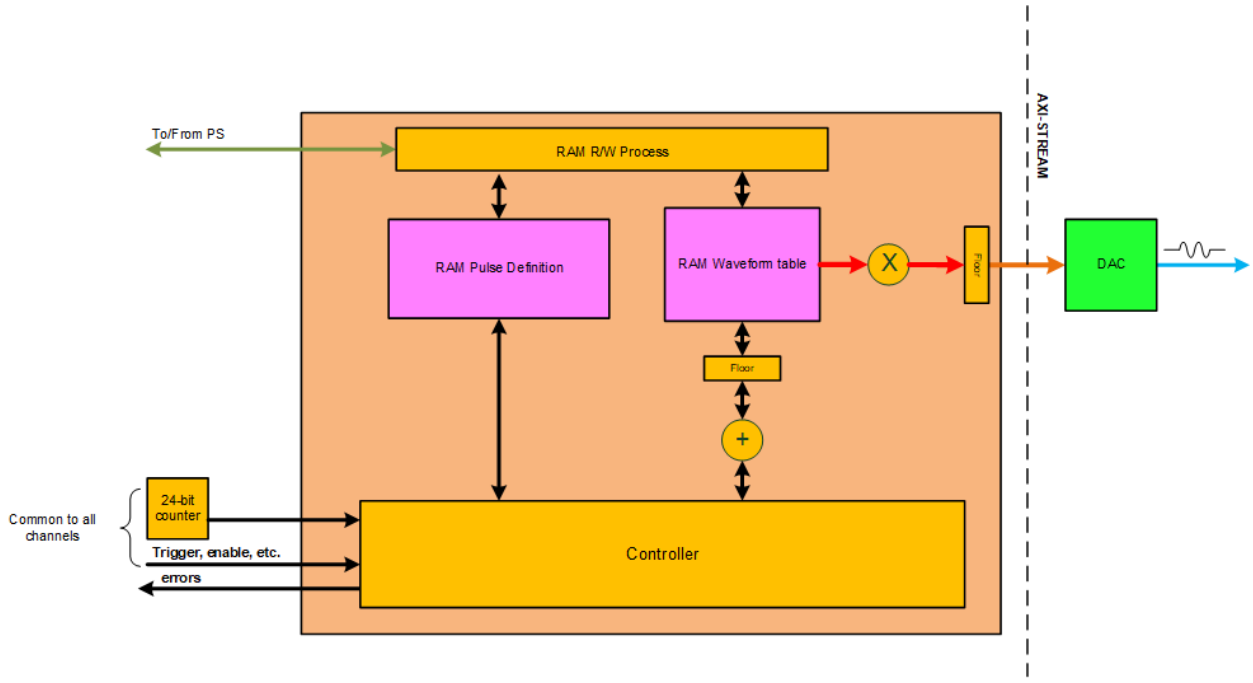

Figure 3.1: Block diagram of the pulse channel with two RAMs integrated.

pulse generation to detect anomalies and trigger immediate responses. Pipeline delays are carefully managed to ensure signal synchronization and maintain timing margins across the data path.

3.2 Setup

Waveform Table

The waveform table memory stores 16-bit values that define each pulse's shape. It is implemented as a dual-port RAM using the Xilinx Block Memory Generator [1]. One port accepts user read and write operations, while the other, designated for the pulse channel's

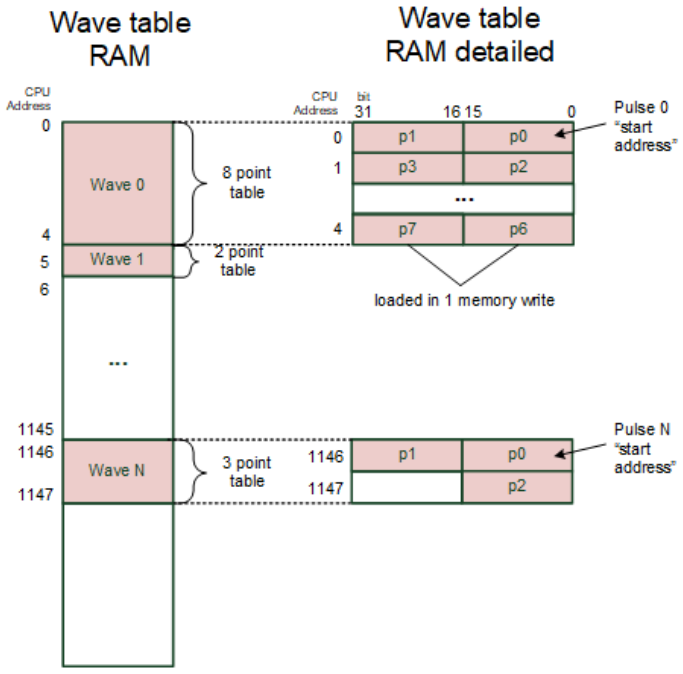

Figure 3.2: Memory layout for waveform table

internal controller (orange box in Figure 3.1), is read-only. As illustrated in Figure 3.2, users write two 16-bit values per memory access, while the controller reads one 16-bit value. This simultaneous access facilitates real-time monitoring and debugging of waveform values without interfering with pulse generation.

In addition, the waveform table memory employs dynamic allocation. Each pulse configuration includes a start address and a waveform length, which together uniquely identify a waveform in memory. This design allows pulses with a common waveform to share the same memory region, ensuring that only the space required for these characteristics is allocated.

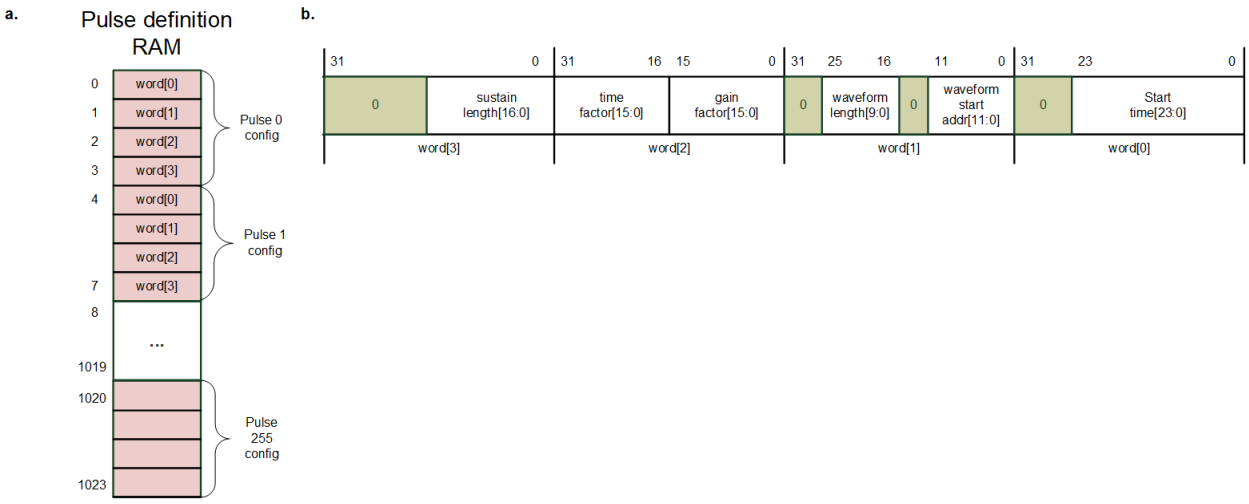

Figure 3.3: Pulse Definition RAM overall allocation. **a**, the overall allocation in the memory. **b**, specific bit-wise allocations of each pulse's parameter.

3.2.1 Pulse Definition

The pulse definition memory, analogous to the waveform table memory, is implemented using a Xilinx Dual-Port RAM. Unlike the waveform table, it adopts a static allocation scheme, whereby each user-defined pulse configuration is assigned four consecutive 32-bit words, forming a 128-bit entry (see Figure 3.3a). This organized structure minimizes addressing overhead and simplifies data retrieval. Figure Figure 3.3b illustrates the specific allocation of each pulse configuration in the pulse definition memory.

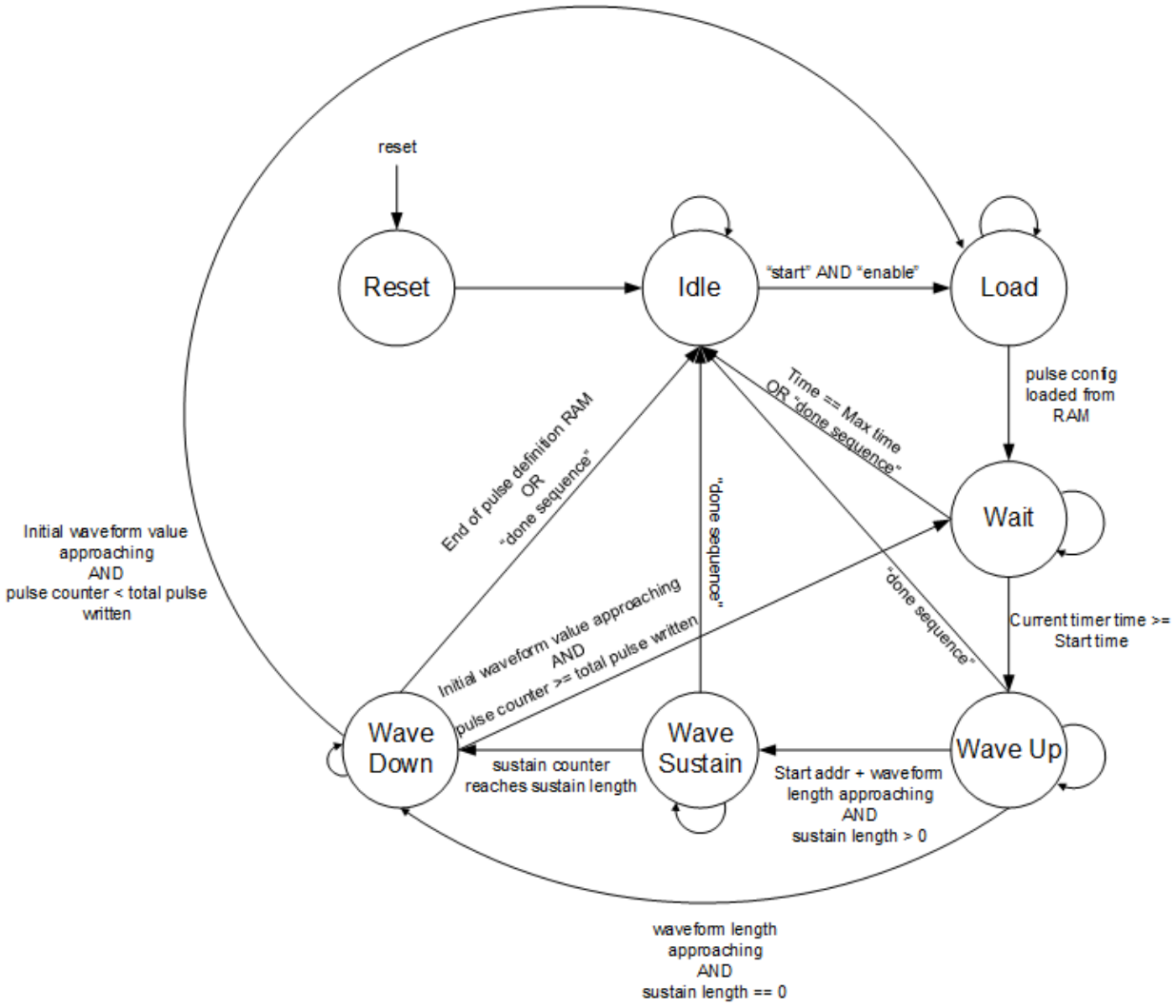

Figure 3.4: State Diagram of the Pulse Channel Module

3.2.2 Internal Controller

The internal controllers of the Pulse Channel module generate a precise pulse sequence by transitioning through a series of well-defined states, as illustrated in Figure 3.4. The module is activated when an external trigger is received and the module is enabled, as shown in the bottom left of Figure 3.1. Upon activation, it loads the required data from the pulse definition memory into a set of registers and then enters a waiting state until the 24-bit timer reaches the predetermined start time. This design guarantees precise pulse generation at the intended moments, ensuring seamless system synchronization.

When the timer reaches the start time, the module moves sequentially through the rise, sustain, and fall stages. During these stages, it carefully shapes the pulse by controlling both the waveform table address and its data output using the time (”+” sign in Figure 3.1) and gain (”x” sign in Figure 3.1) factor values. After the fall stage, the module evaluates whether additional pulses remain in the pulse definition memory. If there are more pulses, it immediately loads the next one. Otherwise, it waits for a ”done sequence” signal before going idle.

An internal pulse counter monitors the current entry in the pulse definition memory and increments each time a new pulse is loaded. Because each pulse definition spans four memory addresses, the counter increases by four with every update. Another counter tracks the number of clock cycles that the sustain phase has lasted following the rise stage. When this counter reaches the defined sustain length, the state machine transitions to the fall stage. This organized, stage-based design simplifies the control logic while enhancing timing

precision and signal integrity.

In parallel, a dedicated process manages memory read and write operations via the CPU interface, as indicated by the green arrows in Figure 3.1. Through this interface, users designate the target memory by including the appropriate identifier in the provided address. For the waveform table, each write operation transfers a 32-bit value, which together constitute one entry. This arrangement yields a total of 2048 available entries for user write accesses. In contrast, each pulse definition entry spans four 32-bit words. Therefore, it takes four consecutive write operations to complete one pulse definition entry. This clear division of memory operations not only ensures efficient and reliable data transfer but also simplifies the addressing scheme, maintaining data integrity and responsiveness in real-time FPGA applications.

3.3 Error Handling

The pulse channel is engineered to be highly resilient, capable of handling both typical user inputs and unusual or invalid parameters. It accounts for common values as well as edge cases that might otherwise disrupt system operation. While the software interface manages most user-induced errors, the built-in error registers and handling mechanisms serve as the final safeguard. This layered approach prevents hazardous inputs from causing system failures, ensuring that the design remains safe and reliable even under unexpected conditions.

To ensure performance, several potential cases must be carefully considered. The design guarantees that pulses begin only when the current time meets or exceeds their scheduled

Table 3.1: Error Registers

Register Bit	Error Details
0	Wave table memory overflow
1	Waveform size ≤ 1 (need minimum of 2 values)
2	Time factor bigger than the size of the rise
3	Gain factor > 1
4	Time factor < 1
5	Start time too early (minimum 50ns apart between pulses)
6	*Not yet assigned*
7	*Not yet assigned*

start. Special attention is given to the sustain length. For instance, if a pulse definition specifies a flat-top length of zero, the system immediately transitions to the fall stage. The module also keeps track of the total number of generated pulses using the internal pulse counter, which helps maintain proper sequence control. Moreover, if the start time of a subsequent pulse is earlier than that of the previous one, the pulse is omitted to preserve timing integrity.

Additionally, the design manages cases where a waveform exceeds the capacity of the waveform table memory. In such instances, the system either wraps the addresses correctly back into the valid range or triggers an error flag in an 8-bit register, with each bit defined in Table 3.1. When a specific error occurs, the system sets only the corresponding flag.

This clear mapping simplifies fault diagnosis and reduces ambiguity during troubleshooting. When an error is detected, the system stops waveform generation to prevent the further propagation of invalid data, thereby maintaining overall system integrity. Furthermore, the registers are designed to log and maintain these error events, aiding in prompt debugging and corrective measures. The error registers can be cleared with an external clear flag or by resetting the system. Some registers have no error details assigned yet, allowing for future expandability based on further testing results.

Chapter 4. Metrics For Evalutation

This section outlines a comprehensive test plan for the pulse channel module, designed to rigorously evaluate its functionality, performance, and error-handling capabilities. The test strategy is systematically divided into distinct phases. Initially, it focuses on controlled experiments using predefined pulse sequences, ensuring that the module writes and reads fixed parameters accurately and that the hardware faithfully produces the expected pulse patterns. Next, the plan expands to include tests with randomly generated pulse parameters, simulating real-world conditions and assessing the design's robustness against unpredictable inputs. Finally, it validates the module's error detection mechanisms by deliberately injecting invalid parameters, confirming that the associated error flag registers correctly identify and report faults. Together, these testing stages build confidence in the module's reliability and performance, ensuring it meets the stringent requirements of advanced digital systems.

4.1 Memory Access

One of the metrics to evaluate is the pulse channel block's ability to read from and write to both the pulse definition and waveform table memories. This functionality directly affects the pulse channel's overall functionality and reliability. By continuously monitoring this capability throughout various testing phases, one can identify potential issues early on. This early detection ensures that data processed within the module remains consistent and accurate. Adopting this proactive approach not only maintains system stability but also prevents data corruption, thereby enhancing the overall integrity and efficiency of the system.

4.2 Write Known Pulse

A fundamental step in verifying the functionality of the pulse channel module is to perform a preliminary test using two to three predetermined pulses with fixed parameters. This carefully controlled test is designed to evaluate whether the hardware can accurately receive and produce known values, ensuring that each pulse is generated exactly as specified. Moreover, this initial validation helps identify potential issues before the module is deployed in scenarios involving more complex or real-world signal patterns. It provides a clear benchmark for performance, aids in troubleshooting early-stage design flaws, and supports iterative refinement of the system. In doing so, the test not only ensures compliance with the required technical specifications but also instills confidence in the module's ability to operate reliably under varying conditions.

4.3 Controlled Randomized Test

After verifying that the hardware behaves as expected during initial tests that memory read and write operations execute correctly, a more comprehensive regression is initiated. In this phase, a software generator creates random pulse parameters and base rise values applicable to both the hardware design and its floating-point software equivalent model. This randomized input methodology simulates a wide range of real-world conditions, thereby enhancing the testing process in terms of efficiency and reliability.

The software model calculates a theoretical floating-point value for each randomly generated pulse, serving as the benchmark for comparisons. Since FPGA hardware uses integer-

based addresses and data, pulse parameter values are scaled and rounded down to the nearest integer before being written to memory or interfaced with peripherals, as shown in Figure 3.1. After conversion, hardware-produced values are compared to the modeled floating-point values. Discrepancies may arise due to rounding. To quantify these differences, an error metric is computed as the absolute difference between theoretical and actual values. Additionally, a delta value measures the difference between sequential hardware output values, providing insight into system behavior over time. Figure 6.1 illustrates the test results.

4.4 Check Error

The pulse channel module is equipped with error flag registers designed to detect and flag invalid pulse parameters. To confirm that these error detection mechanisms operate as intended, the test introduces invalid parameter values into the system. For example, configuring a pulse with a rise length of 1 and a step value of 2 is expected to simultaneously trigger the error flags in registers 1 and 2, as detailed in Table 3.1. Systematically verifying that the system accurately identifies and reports these errors ensures the pulse channel's capability to handle unforeseen or erroneous conditions effectively.

Chapter 5. System Integration

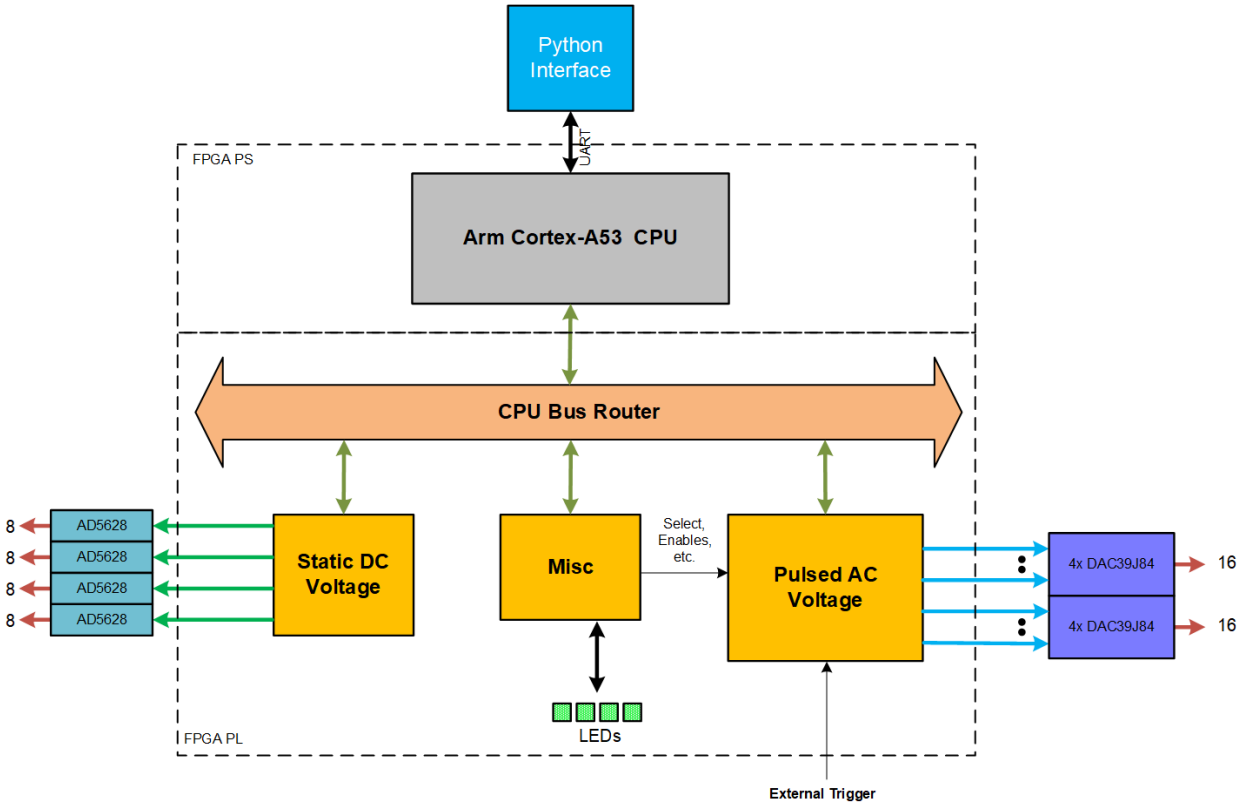

Figure 5.1: FPGA system architecture block diagram.

The pulse channel module is an integral component of an FPGA system designed to generate 32 pulsed AC voltage signals and 32 static voltage signals. This system leverages both the processing systems and the programmable logic of the ZCU102 architecture. A Python-based interface allows users to input specific information about the pulses and voltage signals. This interface translates user requests into custom commands, which are then sent to the processing system via the FPGA's UART interface.

The processing system's ARM-based CPU converts Python-generated commands into addressable instructions for the FPGA's blocks. Within the FPGA hardware, a bus router takes the byte-addressable data from the CPU and translates it into block-specific word-addressable data. The system comprises three distinct hardware blocks, each serving a separate function. The orange blocks in Figure 5.1 illustrate the names of these blocks, which will be discussed in detail in the subsequent sections. This architecture ensures efficient communication and data processing within the FPGA system, enhancing its overall performance and reliability.

5.1 Static DC Voltage Block

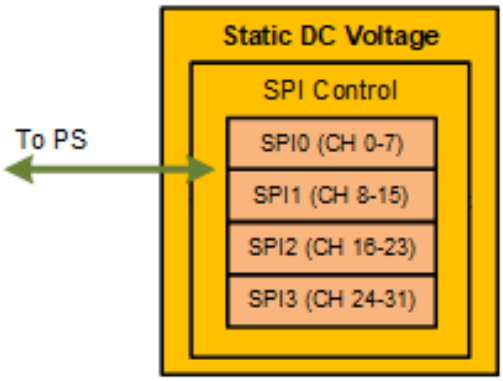

Figure 5.2: Block diagram for the DC voltage module

The DC block provides stable and programmable static voltage control using four Analog Device AD5286 PMOD DACs. Each DAC supports eight voltage channels, making a total of 32 channels, as illustrated in Figure 5.2. Communication between the DACs and the system occurs via the SPI interface. These SPI interfaces are implemented as separate modules, each managing the clock, data, and chip select signals for data transmission to the DACs.

Table 5.1: SPI DAC message format

31	28	24	22	20	19	8	7	0
0	Update DAC	0	Channel select		Data		0	

The DC block utilizes the lower five bits of the CPU address to select one of four SPI-based DACs. Part of the address is used in the SPI message for the chosen DAC, as detailed in Table 5.1. This address segment specifies the ”channel select” portion of the command, determining which of the eight channels on the selected DAC will be used. The SPI message also includes an ”update DAC” command to instruct the DAC to refresh the chosen channel's value, which is set in the ”data” portion of the message. The processing system provides a 12-bit positive integer value for the DACs, which is then converted to a voltage value as specified by:

$$V_{out} = V_{ref} \times \frac{D}{2^{12}} \tag{5.1}$$

Where V_{out} is the voltage output by the DAC, V_{ref} is the DAC's reference voltage, and D is the data from the processing system [2]. This design ensures precise and flexible static voltage control. Additionally, the module supports write operations to update DAC channels and control features such as internal reference voltage and power. It also supports read operations, which return status flags for each SPI transaction. The use of multiple DACs and independent SPI modules allows for efficient handling of numerous voltage channels, providing enhanced control and reliability.

5.2 Pulse AC Voltage Block

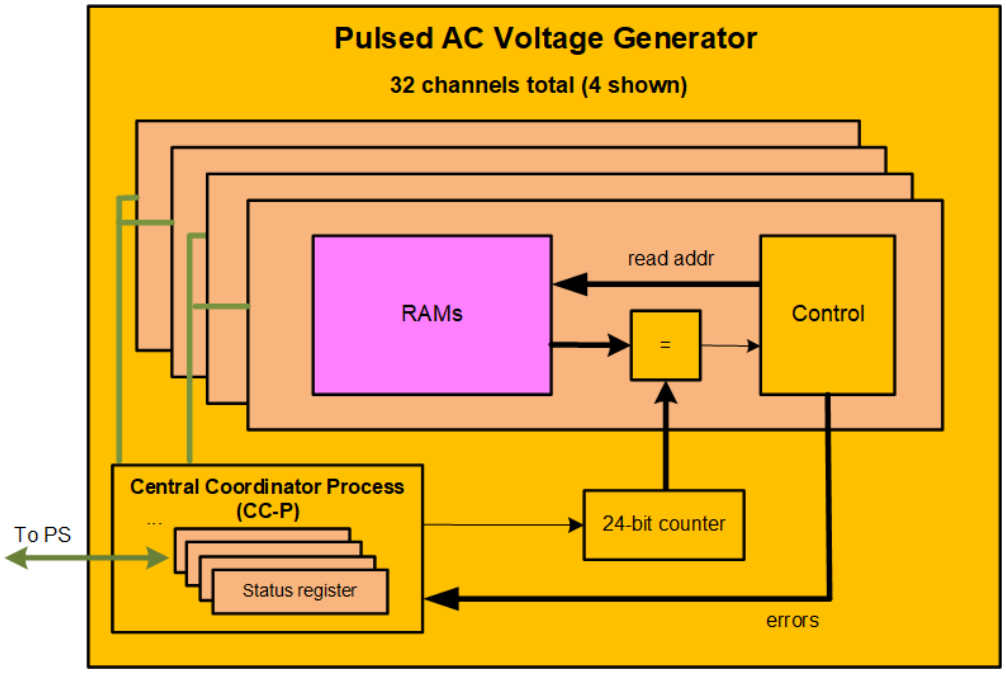

Figure 5.3: Block Diagram of the AC block.

The pulse generation module—also referred to as the AC block—is a synchronous multi-channel pulse generator designed to deliver precise control for laser operations on the photonic chips. Its architecture generates pulse sequences with a 10-nanosecond resolution across 32 channels, ensuring that each digital output is synchronized.

The AC block acts as the central coordinator for all pulse channels. A Central Coordinator Process (CC-P) manages both external control signals and error notifications from individual channels, ensuring reliable overall performance. Central to its operation is a 24-bit counter that functions as a common timer, as seen in Figure 5.3. Upon receiving an external trigger, this counter initiates the timing process uniformly for all channels. It continues until

it reaches either a user-defined sequence length or its maximum value, at which point it sends stop signals to halt pulse generation. This coordinated stopping mechanism maintains timing precision and prevents runaway pulses that could lead to data inconsistencies.

A series of internal registers underpin the module's functionality. These registers store key parameters such as the total sequence length, the active channel enable, and select for memory reading and writing operations. Values are loaded into these registers via the CPU interface, which streams configuration data from the processing system. The design supports simultaneous writing to multiple channels, a useful feature for updating pulse parameters in parallel. However, because the data bus is only 32 bits wide, the system limits read operations to one channel at a time.

The module also incorporates an addressing scheme that distinguishes between channel-specific memory and internal control registers. The lower address range is reserved for waveform data and pulse parameters, while the upper range is dedicated to registers for the CC-P. Registers for settings such as sequence length, channel enables, and channel selections are both readable and writable. This bidirectional access permits real-time control and monitoring of each channel, facilitating adjustments as operational conditions change. In contrast, dedicated read-only registers continuously convey status information from both the module and the individual pulse channels.

Error handling within the AC block is equally streamlined. As discussed, each pulse channel generates error signals that are collected as multi-bit vectors. The CC-P then transposes these into several 32-bit registers, with each register corresponding to a specific error

type defined in Table 3.1. Within a given register, each bit represents the status of one of the 32 channels. This systematic condensation of error information into a single-dimensional format simplifies monitoring and debugging processes, enabling engineers to quickly identify and address issues.

5.3 Miscellaneous

The miscellaneous block is a versatile control and status interface integrated into the system. It manages a variety of auxiliary functions that streamline control processes. This module organizes basic operations, such as managing system interfacing, monitoring status, and distributing control signals. Its primary role is to bridge CPU control with hardware functionality through several dedicated registers. These registers handle tasks such as version reporting, LED indication, and debug routing. A notable feature is its configurable LED control system. This system allows either PS or PL to drive status indicators based on the LED enable register settings, providing flexibility and accurate control. Isolating these auxiliary functions enhances maintainability, simplifies debugging, and promotes reusability. Beyond its fundamental capabilities, the miscellaneous block streamlines operations and supports a modular system architecture.

5.4 Software Control Interface

A dedicated Python package translates user-defined waveform values and pulse information into addresses and data formats for the hardware [7]. This abstraction bridges the gap between users with limited hardware design experience and the intricate nature of FPGA

architectures. By automating tasks such as memory management and data formatting, the system enables users to focus on refining their quantum control strategies rather than wrestling with hardware-level operations. This design philosophy enhances both usability and performance across the entire framework.

Table 5.2: Sample Format of the Waveform Record

Wave ID	262144	196612	327688
Waveform Values	0	0	0
	1	2	5
	3	5	6
	4	-	8
	-	-	12

The interface functions similarly to a streamlined web API, allowing users to submit waveform data and parameters intuitively. For every pulse entry, the software automatically locates the next available memory space, computes the starting address and length, and allocates the data accordingly, returning a unique waveform ID for straightforward reference. This methodical allocation ensures that inputs accumulate properly and that any error due to exceeding available memory is promptly flagged. Additionally, the system archives each waveform record in a dedicated file, providing traceability as detailed in Table 5.2. The first row for each column in the table represents a unique six-digit wave ID. The remaining rows are values for each waveform.

In a similar fashion, pulse parameter management is performed through dedicated func-

Table 5.3: Sample Format of the Pulse Parameter Record

Wave ID	Start Time	Scale Gain	Scale Address	Sustain
262144	5	0.98	1.2	5
196612	48	1.0	1.1	7
327688	100	1.0	1.0	0

tions that support both loading and retrieval. Unlike waveform data, which can vary in length and address, pulse parameters are stored in fixed groups of four values. Users provide key details such as the pulse's start time, time factor, gain factor, and sustain time, along with the relevant waveform ID. The interface then leverages this ID to fetch additional hardware parameters—such as the corresponding waveforms start address and length—converting the user inputs into a hardware-accepted format as outlined in Figure 3.3b. Corresponding records are then logged in a file following the format shown in Table 5.3.

After processing, the serial interface passes these hardware-ready values to a custom Python class that integrates them with subroutines designed for the FPGA. Communication with the embedded ARM CPU is maintained via a UART connection, with the third-party PYSerial package ensuring reliable data transfer. Commands transmitted by the Python class are interpreted by the PS to activate specific design blocks defined in Figure 5.1. This layered control strategy not only streamlines user interaction but also preserves the precision and high performance required by contemporary FPGA applications.

Chapter 6. Outcome and Result

6.1 Simulation Result

a.

```
# Simulation start
# generating random polynomials...
# Seed: 1751023766
# Generating pulse 0
# Generating pulse 1
# Generating pulse 2
# Generating pulse 3
# Generating pulse 4
# Generating pulse 5
# Generating pulse 6
# Generating pulse 7
# Generating pulse 8
# Generating pulse 9
# Generating pulse 10
# Done generating random polynomials
# Total Pulses: 10
# Load pulse RAM
# Pulse RAM loaded
# Load waveform RAM
# Waveform RAM Loaded
# Start the pulse outputs
# "pulse_0": {
#     "constants":    [0.167086, 0.006624],
#     "gain_factors": 0.179047,
#     "time_factors": 216.511719,
#     "size":         636.000000,
#     "status":       "PASS"
# },
# "pulse_1": {
#     "constants":    [0.753583, 0.713204],
#     "gain_factors": 0.056854,
#     "time_factors": 163.980469,
#     "size":         742.000000,
#     "status":       "PASS"
# },
# "pulse_2": {
#     "constants":    [0.118583, 0.509541],
#     "gain_factors": 0.919312,
#     "time_factors": 1.613281,
#     "size":         645.000000,
#     "status":       "PASS"
# },
# "pulse_3": {
#     "constants":    [0.525944, 0.696603],
#     "gain_factors": 0.582672,
#     "time_factors": 240.199219,
#     "size":         555.000000,
#     "status":       "ERROR",
# }
# Hardware error detected: 00100000
# Simulation done
```

b.

```
# Simulation start
# generating random polynomials...
# Seed: 1234
# Generating pulse 0
# Generating pulse 1
# Generating pulse 2
# Generating pulse 3
# Generating pulse 4
# Generating pulse 5
# Done generating random polynomials
# Total Pulses: 5
# Load pulse RAM
# Pulse RAM loaded
# Load waveform RAM
# Waveform RAM Loaded
# Start the pulse outputs
# "pulse_0": {
#     "constants":    [0.999591, 0.786095],
#     "gain_factors": 0.241608,
#     "time_factors": 235.277344,
#     "size":         293.000000,
#     "status":       "PASS"
# },
# "pulse_1": {
#     "constants":    [0.834332, 0.985331],
#     "gain_factors": 0.059753,
#     "time_factors": 213.710938,
#     "size":         938.000000,
#     "status":       "PASS"
# },
# "pulse_2": {
#     "constants":    [0.086698, 0.378265],
#     "gain_factors": 0.756226,
#     "time_factors": 112.011719,
#     "size":         872.000000,
#     "status":       "PASS"
# },
# "pulse_3": {
#     "constants":    [0.767287, 0.402101],
#     "gain_factors": 0.645905,
#     "time_factors": 129.070313,
#     "size":         806.000000,
#     "status":       "PASS"
# },
# "pulse_4": {
#     "constants":    [0.467824, 0.293503],
#     "gain_factors": 0.980957,
#     "time_factors": 151.343750,
#     "size":         898.000000,
#     "status":       "PASS"
# },
# Simulation done
```

Figure 6.1: Simulation result of **a.** Seed with incorrect parameters and **b.** Seed with normal parameters

To validate the functionality and ensure the reliability of the pulse channel module, a controlled, randomized regression test was designed and implemented. This test serves as a comprehensive evaluation of the module's performance across several key aspects. Specifically, it examines the module's ability to execute memory read and write operations efficiently, preserve data accuracy during these processes, and manage potential errors effectively. These assessments are the cornerstone of identifying and resolving any latent issues that could

compromise the module's functionality in practical applications. The results of this testing process provide significant insights into the module's behavior and confirm its expected performance. Figure 6.1 illustrates two simulation outcomes, both generated using Intel ModelSim, which visually validate the module's operation under the given test scenarios.

6.2 Board Result

The system is examined using Xilinx's Integrated Logic Analyzer (ILA) to validate signals from multiple modules, including the single pulse channel module.

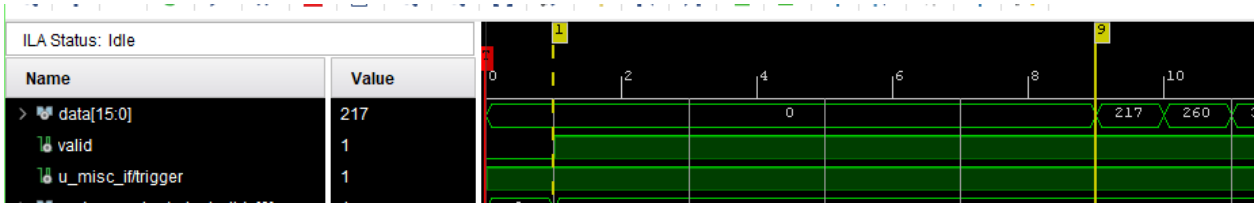

Figure 6.2: ILA monitored the result from the top level.

A 55-sample waveform, starting from a nonzero value at 50 ns, is transmitted to the hardware and captured by the ILA (see Figure 6.2). The first output appears after 8 cycles, with each cycle lasting 10 nanoseconds. This timing results in an 80-nanosecond delay, confirming that the pulse generation operates well within the sub-microsecond range required for quantum control hardware. Furthermore, ILA data indicate that data updates occur every 10 nanoseconds, highlighting the rapid switching performance required for high-speed applications. Additionally, an external oscilloscope recorded a pulse sequence in Figure 6.3, further verifying the overall functionality and behavior of the design.

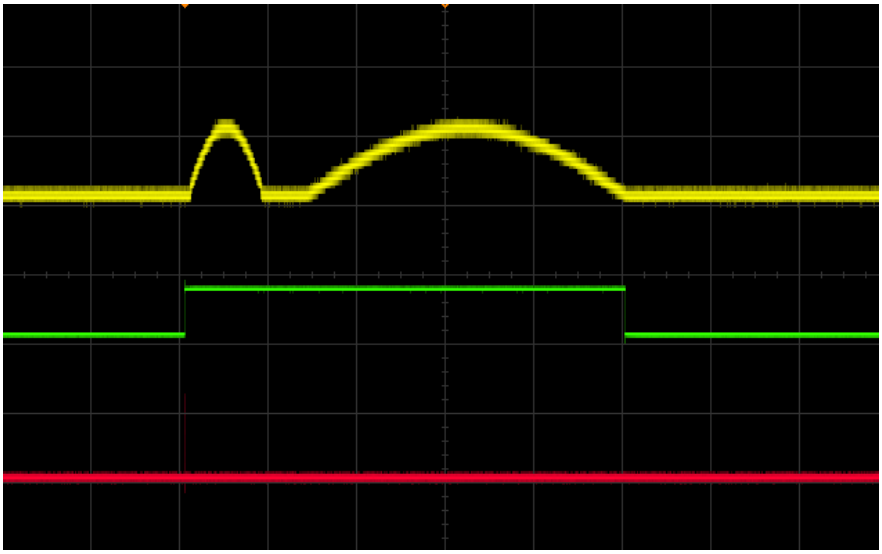

Figure 6.3: A single pulse sequence captured on an oscilloscope. The yellow line is the pulse sequence, the green line is the valid signal, and the red line is the trigger signal.

6.3 Resource Utilization

In FPGA design, efficient resource utilization is integral for ensuring a high-quality, reliable system. One of the most core aspects of this evaluation is timing analysis. After implementation, the Xilinx Vivado 2022.1 tool generates an in-depth timing report that details the minimum and maximum delay paths within the design. Generated reports in Figure 6.4 confirm that both setup and hold times meet the specified constraints.

This timing verification ensures that all synchronous elements operate correctly within the designated clock cycle, preventing data corruption and operational failures.

Table 6.1 shows that only a small fraction of the FPGA resources are used. This low usage proves the design's efficiency. Minimal resource usage reduces power consumption. It also

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             4.674ns  (required time - arrival time)
  Source:                 u_ps2/ps1_i/zynq_ultra_ps_e_0/U0/PS8_i/MAXIGP0ACLK
                            (rising edge-triggered cell PS8 clocked by clk_pl_0  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:
u_ps2/ps1_i/axi_smc/inst/s00_entry_pipeline/s00_si_converter/inst/gen_axilite_conv.axilite_conv_inst/ar_cnt_reg[4]/D
                            (rising edge-triggered cell FDRE clocked by clk_pl_0  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             clk_pl_0
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (clk_pl_0 rise@10.000ns - clk_pl_0 rise@0.000ns)
  Data Path Delay:        5.135ns  (logic 1.169ns (22.763%)  route 3.966ns (77.237%))
  Logic Levels:           6  (LUT4=1 LUT5=1 LUT6=4)
  Clock Path Skew:        -0.086ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    1.515ns = ( 11.515 - 10.000 )
    Source Clock Delay      (SCD):    1.695ns
    Clock Pessimism Removal (CPR):    0.094ns
  Clock Uncertainty:      0.130ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Discrete Jitter          (DJ):    0.250ns
    Phase Error              (PE):    0.000ns
  Clock Net Delay (Source):      1.532ns (routing 0.567ns, distribution 0.965ns)
  Clock Net Delay (Destination): 1.385ns (routing 0.510ns, distribution 0.875ns)
Min Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             0.004ns  (arrival time - required time)
  Source:                 u_dacs_pulse/ch_enables_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clk_pl_0  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            u_dacs_pulse/g_ch[1].ch_full.u_ch/enable_d1_reg/D
                            (rising edge-triggered cell FDRE clocked by clk_pl_0  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             clk_pl_0
  Path Type:              Hold (Min at Slow Process Corner)
  Requirement:            0.000ns  (clk_pl_0 rise@0.000ns - clk_pl_0 rise@0.000ns)
  Data Path Delay:        0.241ns  (logic 0.058ns (24.065%)  route 0.183ns (75.935%))
  Logic Levels:           0
  Clock Path Skew:        0.177ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    1.796ns
    Source Clock Delay      (SCD):    1.526ns
    Clock Pessimism Removal (CPR):    0.094ns
  Clock Net Delay (Source):      1.396ns (routing 0.510ns, distribution 0.886ns)
  Clock Net Delay (Destination): 1.633ns (routing 0.567ns, distribution 1.066ns)
```

Figure 6.4: Partial timing report from Vivado on the worst slack (top) and hold (bottom).

helps the FPGA maintain optimal timing and routing. This efficiency is ideal for scalable systems that may need future modifications. It validates the careful design decisions made during synthesis and optimization.

Table 6.1: Post-Implementation Resource Utilization Reported by Vivado

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.034	3	—	—
CLB Logic	0.018	30403	—	—
LUT as Logic	0.015	12671	274080	4.62
CARRY8	0.002	909	34260	2.65
Register	<0.001	12952	548160	2.36
LUT as Shift Register	<0.001	4	144000	<0.01
Others	0.000	567	—	—
Signals	0.029	24867	—	—
Block RAM	0.174	128	912	14.04
DSPs	<0.001	2	2520	0.08
I/O	0.014	36	328	10.98
PS8	2.472	1	—	—
Static Power	0.750			
PS Static	0.098			
PL Static	0.653			
Total	3.492			

Chapter 7. Discussion

7.1 Hardware Precision

The current FPGA-based control system for quantum laser control demonstrates proof-of-concept functionality and lays a foundation for future applications. However, test results reveal consistent discrepancies between theoretical and measured values. These errors stem primarily from the rounding mechanism. Figure 3.1 indicates that data and address inputs in the waveform table memory are floored to the nearest integer, which discards important fractional components and limits resolution.

Waveforms are generated from predefined tables that are externally computed and then loaded into the system. This method simplifies on-chip logic and resource management by avoiding complex dynamic computations. However, using a fixed 16-bit integer representation introduces quantization errors when multiple write operations produce a complete waveform. The resulting trade-off between simplicity and precision underscores the limitations of the current design. To improve accuracy, enhanced rounding methods, such as midpoint rounding or adaptive precision schemes, could reduce truncation errors. Adopting dynamic waveform generation lowers the data transfer volume. In this approach, only key parameters—amplitude, frequency, and phase—are specified. The system computes the complete waveform in real time. This method produces waveforms that more closely match their theoretical definitions. The reduced data transfer volume also lowers storage and communication overhead.

7.2 Hardware Error Handling

The system utilizes a basic error detection framework that effectively identifies invalid parameters from the user. When an error occurs, the system sets a status flag for the controlling processor to read. However, it relies on a passive notification system rather than active error recovery. This approach depends entirely on a dedicated software interface to resolve issues, which can lead to complications if the software is not properly implemented or if user parameter conversions fail.

The pulse channel's definition memory assumes waveform parameters are provided in a consecutive, incrementing order. The hardware lacks a built-in mechanism to verify or correct this order. The module directly maps the provided address to the memory, which can lead to data inconsistencies if users do not sequence their parameters correctly. Future development should introduce a hardware mechanism to enforce cumulative writes of pulse parameters.

Additionally, permitting users to write to any memory location enhances system flexibility but also introduces the risk of overlapping writes. Without rigorous external software control over memory address management, the probability of data overwrites increases. Addressing this vulnerability in future work at the hardware level helps to maintain system stability and reliability.

7.3 Interface Latency

Many user-level interfaces are implemented in Python on a separate PC, requiring UART serial writes to the hardware system. Each write transmits only 8 bits of data from the host to the FPGA, resulting in limited throughput and extended read and write times. Tests show that reading and writing an entire waveform and one pulse configuration each take an average of three to four seconds. These delays become significant bottlenecks, especially when the system needs to access memories multiple times during operation.

To mitigate this issue, sophisticated C firmware could replace much of Python's functionality in the future. This firmware would reside on Xilinx's processing system, fully leveraging Zynq's high-speed AXI interface to accelerate data transfers. Instead of relying on Python functions to relay user parameters, the processing system would accept input directly from a command-line interface. Users could manually type parameters or design scripts to automate their entry.

Chapter 8. Conclusion

In this thesis, a novel laser control system for trapped-ion quantum experiments is introduced and successfully tested, demonstrating a breakthrough design implemented on the Xilinx Zynq FPGA platform. The system integrates external digital-to-analog converters to achieve synchronous control over 32 laser channels with a minimum switching time of 10 ns for the precision required in quantum waveform generation. To ensure reliability and performance, a single pulse channel capable of generating the desired waveform is first developed and thoroughly evaluated. This prototype is then scaled to the full system through the incorporation of custom programmable logic blocks. These blocks deliver stable DC voltage control alongside high-frequency pulsed waveform generation, ensuring versatile and accurate signal manipulation that meets the stringent demands of quantum experiments. By leveraging the FPGA's built-in features, the design attains both high performance and throughput. This strategic resource utilization guarantees precise hardware timing and maximum efficiency, addressing the complex challenges of laser control in trapped-ion systems. Furthermore, a sophisticated user interface is developed to allow users to focus on designing experiments rather than managing low-level hardware configurations. Collectively, these integrated features enhance system responsiveness and scalability, setting the stage for future innovations in quantum computing hardware.

BIBLIOGRAPHY

[1] AMD Inc. *LogiCORE IP Block Memory Generator v8.4*, Dec 2023. PG058.

[2] Analog Devices. *AD5628/AD5648/AD5668: 12-/14-/16-Bit, I2C Voltage Output DACs*, 2020. Rev. I, September 2020.

[3] C. W. Hogle, D. Dominguez, M. Dong, A. Leenheer, H. J. McGuinness, B. P. Ruzic, M. Eichenfield, and D. Stick. High-fidelity trapped-ion qubit operations with scalable photonic modulators. *npj quantum information*, 9(1):74–6, 2023.

[4] Adrian J Menssen, Artur Hermans, Ian Christen, Thomas Propson, Chao Li, Andrew J Leenheer, Matthew Zimmermann, Mark Dong, Hugo Larocque, Hamza Raniwala, Gerald Gilbert, Matt Eichenfield, and Dirk R Englund. Scalable photonic integrated circuits for programmable control of atomic systems, 2022.

[5] Galan Moody, Volker J Sorger, Daniel J Blumenthal, Paul W Juodawlkis, William Loh, Cheryl Sorace-Agaskar, and et al. 2022 roadmap on integrated quantum photonics. *JPhys Photonics*, 4(1):12501–, 2022.

[6] University of Washington ACME Lab. Nano_qlaser. `https://github.com/uw-acme/NANO_QLASER/tree/eric_zcu102_all_pmods`, 2025. Accessed: 2025-05-29.

[7] University of Washington ACME Lab. Qlaser_zcu_python. `https://github.com/uw-acme/qlaser_zcu`, 2025. Accessed: 2025-05-29.

[8] Daniel T. Schussheim and Kurt Gibble. A many-channel fpga control system. *Review of Scientific Instruments*, 94(8):085101, 08 2023.

[9] A. Sitaram, G. K. Campbell, and A. Restelli. Programmable system on chip for controlling an atomic physics experiment. *Review of Scientific Instruments*, 92(5):055107, 05 2021.

[10] Xilinx Inc. *ZCU102 Evaluation Board User Guide*, v1.7 edition, February 2023. UG1182.